

SCAR: Smart Collision Avoiding Robot

SYSC 4907 4th year project: Final Report 2019/2020

Presented to Professor Howard Schwartz

Group 68

Shivanshi Sharma: 101037387

Adam Labelle: 101038735

Brian Ranjan Philip: 101018883

Shaviyo Marasinghe: 101019133

April 07, 2020

Letter of Transmittal

Team SCAR: Shivanshi Sharma, Shaviyo Marasinghe, Brian Phillip, Adam Labelle
Carleton University 4th Year Project – SCAR
1125 Colonel By Dr,
Ottawa, ON K1S 5B6

Tuesday, April 7, 2020

Dr. Howard Schwartz
Professor, Department of Systems and Computer of Engineering
Carleton University
1125 Colonel By Dr,
Ottawa, ON K1S 5B6

Dear Dr. Howard Schwartz and Dr. Mostafa Taha,

Our project group SCAR (Smart Collision Avoidance Robot) is writing to you to acknowledge our final report document which demonstrates our work delivered in our 4th year project. The project officially started on September 15th, 2019 and was completed on April 6th, 2020. The report discusses the details such as designing, implementing, and testing of various components and the entire system. It states the importance of learning how new technology works and the ample amounts of possibilities related to building a collision-avoidance robot. In our project, we attempted to engineer a prototype robotic car that can avoid obstacles while following real-time instructions for movements, as well as detect lanes.

The report shows our group's dedication to come as close to accomplishing our tasks, as we could. Despite of many obstacles and unexpected issues with hardware/software components, the team was able to come together to finish individual deliverables. At the end of the report, the conclusion states how all the team members have had a chance to grow individually, as well as develop interpersonal skills to learn team management and collaborate.

Lastly, we would like to thank you for your tremendous support and guidance throughout the project, as it helped our group reach our goal and stay motivated. Your valuable information and feedback inspired us to write this report and to expand our knowledge in this area. In addition, we would also like to thank Professor Mostafa Taha for his time he dedicated towards our presentation, as well as taking interest in our project.

Yours sincerely,

Team SCAR

Executive Summary

The objective of this research project was to research and engineer the steps that would need to be followed to create a self-driving robotic vehicle. The response to this motivation was to engineer a prototype robotic car capable of maneuvering itself through a lane on the ground, detect a collision that might be inevitable, and stop the collision from happening in a safe and timely manner. The first step to achieving this milestone was by maintaining a connection with a server through a Local area network (LAN). This server will be the brains of this operation that does the duty of guiding this wireless robot. This pair will be tasked to do the following: make and follow instructed movement, detect and avoid collisions, and detect and drive within physical lanes on a road. The engineering fields of Computer Systems and Software were the technical backgrounds of the team members that took part in contributing to the accomplishments of this robot. At the final demonstration of this robot, it could detect inevitable lane departure and would steer to avoid it. This robot could detect an inevitable collision that could occur if the same trajectory is followed and would take evasive action. Furthermore, all of this is done through wireless instructions that were being communicated over a network by a server that took note of the movements of the robot. This opens the door for more additions that can be added to the project like overhead tracking of the robots, and the integration of more robots in a road like network. Throughout this report, this robot vehicle will be referred to as SCAR; short for Smart Collision Avoiding Robot.

Table of Contents

Letter of Transmittal	2-3
Executive Summary	4
Introduction	10
1. The Engineering Project	11-16
1.1. Engineering Professionalism.....	11
1.2. Health and Safety.....	11
1.3. Project Management.....	12-13
1.3.1. <i>Communication</i>	12
1.3.2. <i>Project Planning</i>	12
1.3.3. <i>Version Control and Code sharing</i>	13
1.4. Individual Project Contributions.....	14
1.5. Individual Report Contributions.....	15-16
2. Background	17-32
2.1. Project Goal.....	17
2.2. Specifications.....	18
2.3. Microcontroller.....	19-22
2.3.1. <i>Raspberry Pi 4</i>	19-20
2.3.2. <i>Raspberry Pi 3b+</i>	21
2.3.3. <i>Raspberry Pi 3</i>	22
2.4. Hardware Kit.....	23

2.4.1. Chassis and Assembly components.....	24
2.4.2. Pi Camera.....	25
2.4.3. Motor Controllers.....	26-27
2.4.3.a. Initial K0073 driver.....	26
2.4.3.b. Post Presentation Motor Controller.....	27
2.4.4. DC Motors.....	28
2.4.5. Ultrasonic Sensor.....	29
2.4.6. Line Tracker.....	30
2.5. LAN Router	31
2.6. Computer used as server.....	32
3. Assembly.....	33-38
3.1. System Overview.....	33
3.2. Raspberry Pi Configuration.....	35-36
3.3. Post Oral Presentation Layout.....	37-39
3.3.1. Final Circuit Diagram.....	37
3.3.2. Detailed Circuit Diagram.....	38-39
4. Flow of Information.....	40-45
4.1. Sequence Diagram.....	40-45
4.1.1. Sequence Diagram of the Entire System	41-42
4.1.2. Sequence Diagram on the Raspberry Pi Side.....	43
4.1.3. Sequence Diagram on the Server Side	44-45
5. Client-End Programming.....	45-53
5.1. Motor Controls and Chip Use	46

5.1.1. <i>Motor Driver Chip (Dual L293D H-Bridge)</i>	47
5.1.2. <i>Motor Control Logic</i>	48-49
5.2. <i>Pulse Width Modulation (PWM)</i>	50
5.3. <i>Raspberry Pi Camera and Ultrasonic Sensor</i>	51
5.4. <i>Multithreading in The System</i>	52
6. Server-End Programming	54-73
6.1. <i>Initial Set up</i>	55
6.2. <i>Process for Detecting Path</i>	55
6.2.1. <i>Camera Calibration</i>	55-57
6.2.2. <i>Perspective Transformation and Warp Perspective</i>	58-59
6.3. <i>Failed Approach in Edge Detection</i>	60 - 62
6.4. <i>Lane Detection: Successful Model</i>	63
6.4.1. <i>Line Detection</i>	63
6.4.2. <i>Extracting Blue Color from the Picture</i>	64
6.4.3. <i>Extracting Lanes from the Picture</i>	65
6.4.4. <i>Sliding Window Algorithm</i>	66-72
6.5. <i>Latency of the Detection Algorithm</i>	73-74
Demonstration	75
Future Improvements	75
Conclusion	76-77
References	78-82
Appendices	83-84

List of Figures

- 1 Gantt chart the project was proposed and planned according to
- 2 Top down view of what was required of the car to be capable of doing
- 3 Using RAMspeed tool to measure read/write bW for 1MB blocks (in MPps
- 4 Using Python GPIO Zero library to measure switching rates of GPIO pins (in kHz)
- 5 Full K0073 assemble able kit by UCTRONICS that SCAR is based on
- 6 K0073 Chassis that was used for SCAR
- 7 Pi Camera V2 module
- 8 Texas Instruments L293D Dual H-bridge Motor Driver chip
- 9 Similar Geared DC Motor like this was used to move SCAR
- 10 HC-SR04 Ultrasonic sensor module used in SCAR
- 11 The power supply components of SCAR
- 12 D-Link DIR880L Model Router which established the LAN for SCAR vehicle
- 13 The computer used as a server in the system
- 14 Overview System Diagram of SCAR Project
- 15 The meu that is prompted when Raspi-config command is executed
- 16 Screenshot of VNC Viewer attempting to connect to the RPi aboard SCAR
- 17 The overall circuit diagrams
- 18 Figure showing detailed circuit diagram on the modelled car
- 19 Sequence diagram of the entire system
- 20 Sequence Diagram of the Pi side
- 21 Sequence Diagram of the Server Side

- 22 The L293D motor driver chip pinout
- 23 The setting of H-bridge pins to determine direction
- 24 Example of tank steering implemented in SCAR
- 25 Code snippet of a turn
- 26 Different PWM cycles
- 27 The method used to calculate distance
- 28 Threaded functionality of the system
- 29 The child thread created and how interrupts handles it
- 30 Radial Distortion caused due to light rays bending at various wavelengths as it hits the camera
- 31 Graphical Representation of radial and tangential distortion on a 2D plane
- 32 Chessboard images taken by the Pi camera for calibration
- 33 The initial image and code extracting region of interest
- 34 The perspective change in image
- 35 Decision making based on thresholds set by user
- 36 Plotted lines in Hough Line array
- 37 Extracted blue pixels from input image
- 38 Histogram for determining the starting point on the Sliding Window Algorithm
- 39 The core logic in the sliding window algorithm
- 40 Processing lanes lines to extract directional command to the motors
- 41 Only one lane line being detected
- 42 Properties of a second-degree concave down polynomial
- 43 The offset between the car and position where it starts capturing the image

44 The termination of program by sending command 7

45 The latency of the line algorithm system

List of Tables

1 Individual Project Contributions

2 Individual Report Contributions

3 Specifications table of the initial requirements from different project components

4 The latency of the line detection and command extraction algorithm

Introduction

Technology has taken a leap forward by creating highly intelligent autonomous devices, including vehicles. The idea of a self-driving car helps provide various benefits such as increased productivity, reduced labour and costs, as well as enhanced reliability. In addition, by reducing the number of fuel-operated vehicles on the road, the amount of harmful emissions being released can be reduced. As autonomous vehicles are becoming more common, there has been an increase in the demands for their usage. As the main goal of this technology remains to transport people in a safe and organised way from point A to point B, it is important to know the basic mechanism behind object detection. A driver-less vehicle which can help transport people around while being able to stay within lanes and detects obstacles has inspired our group to research and create a car capable of making autonomous decisions, which can demonstrate a range of benefits. This report will outline the technical and nontechnical steps and decisions that were taken by our group to make progress with respect to implementing this technology into a small-scale model using everyday electronics and software programming.

1 The Engineering Project

1.1 Engineering Professionalism

An engineer has an obligation to be a servant of public safety. During this project, precautions were taken to make sure no harm was brought along to any individuals involved with this research project. Due diligence was provided to the respectful individuals and organisations, and academic integrity was respected and practiced throughout the course of the project.

1.2 Health and Safety

The project was practised with health and safety in mind, and precautions were taken to ensure health and safety concerns are addressed. Safe handling boxes were used for the project contents such that unsafe exposure to sharp, unergonomic material would be minimized during transportation. This ensured to protect the car from adverse weather effects and minimize the chances short circuiting and injury from electric shock. During the second phase of the project, Exposed wiring was covered with tape to eliminate the exposures. Wires were colour coded to minimize mistakes during assembly that could lead to shorting and electric shock. Proper packaging practises were performed to keep the center of gravity low and between the wheelbase of the car. Thus, minimizing accidental tipping and uncontrolled driving behaviour from the car. Finally, the operation of the vehicle during the testing phase was done in an open environment without uncontrolled variables that could harm and injure individuals or property.

1.3 Project Management

1.3.1 Communication

During the time span of this project, the medium of communication we used was Facebook messenger. We found Facebook messenger to work best amongst ourselves due to the ease of use, reliable notifying system, multimedia and file sharing feature, and features to make plans, polls, and event notifications. Communication with the project supervisor was carried through emails and weekly meetings to discuss progress.

1.3.2 Project Planning

SCAR project planning was taken shape in a weekly basis. During the weekly meetings with the project supervisor, objectives for the upcoming week were discussed such that they were realistic and attainable but also would make significant progress towards the completion of the project. The following figure below shows the activity that was planned, what week the activity was planned for, the duration that was expected, and the actual start week and duration. In a percentile number, it shows how much of that activity was completed within that duration.



Figure 1: Gantt chart the project was proposed and planned according to

1.3.2 Version Control and Code sharing

Code sharing and version control for algorithms were done using Google Colab. Google Colab allows for in browser python programming with low need for configurations setting modifications. Colab has an easy to use user interface that made programming snippets shareable and easy to run on any computer with an internet connection. At the end of testing, we transferred all implemented code onto PyCharm to be executed on the computer. Software documentation, changing team roles, and operating software on the Pi were shared amongst the team using GitHub. Using a Git repository brought along the benefit of steady version control.

1.4 Individual Project Contributions

Table 1: The table shows each project team member's contribution to the project

Section	Shaviyo	Adam	Shivanshi	Brian
Chassis assembly with Breadboard & battery packs			√	√
Electrical wiring for the H-bridge			√	√
Power distribution and testing				√
Camera Calibration			√	
Testing GPIO pins for output logic to H-bridge				√
Ultrasonic sensor implementation and wiring				√
Functions for both the sensor and motors				√
TCP/IP connection between the Pi and server (bi-directional and synchronized)			√	√
Multi-Threading on Pi side of the system	√			√
Extracting region of interest, produce NumPy array				√
Thresholding colors to extract lane lines				√
Sliding Window Algorithm				√
Function to decide command, PWM				√
Splitting command on Pi, executed by the H-bridge			√	√
Event based interrupts to handle object detection				√

1.5 Individual Report Contributions

Table 2: This table addresses each team member's contribution to the various contents of the report

Section	Shaviyo	Adam	Shivanshi	Brian
Letter of Transmittal			√	
Executive Summary	√			
Introduction			√	
1) The Engineering Project	√			
2) Background				
Initial Plan	√			
Specification	√			
Microcontroller			√	
Hardware Kit	√			
LAN Router	√			
3) Assembly				
System Overview	√			
Raspberry Pi Configuration	√			
Post Oral Presentation Layout			√	√
4) Flow of Information				
UML Sequence Diagram				√
5) Client-end Programming				
Motor Driver Chip				√
Motor Control Logic				√
Pulse Width Modulation				√
Raspberry Pi Camera and Ultrasonic Sensor				√

Multi-threading				√
6) Server-end Programming				
Camera Calibration			√	
Perspective Transformation and Warp Perspective				√
Failed Model (Canny Edge Detection and Hough Transformation)				√
Extracting lane lines				√
Sliding Window Algorithm				√
Latency of server end				√
Demonstration				√
Future Improvements	√			
Conclusion			√	
Appendices			√	

2 Background

2.1 Project Goal

The first step to researching and understanding an autonomous vehicle is to create a small-scale model of this system. Ideally a model vehicle capable of detecting obstructions, maneuvering around them in a safe and decisive manner, all the while staying between a confined zone that would be the restricted area for this vehicle to travel in. In a broad sense it was required that we create a model vehicle that could get a sense of its location through a camera and would make necessary adjustments. The following figures are a baseline of what was set as the goals for this project moving forward.

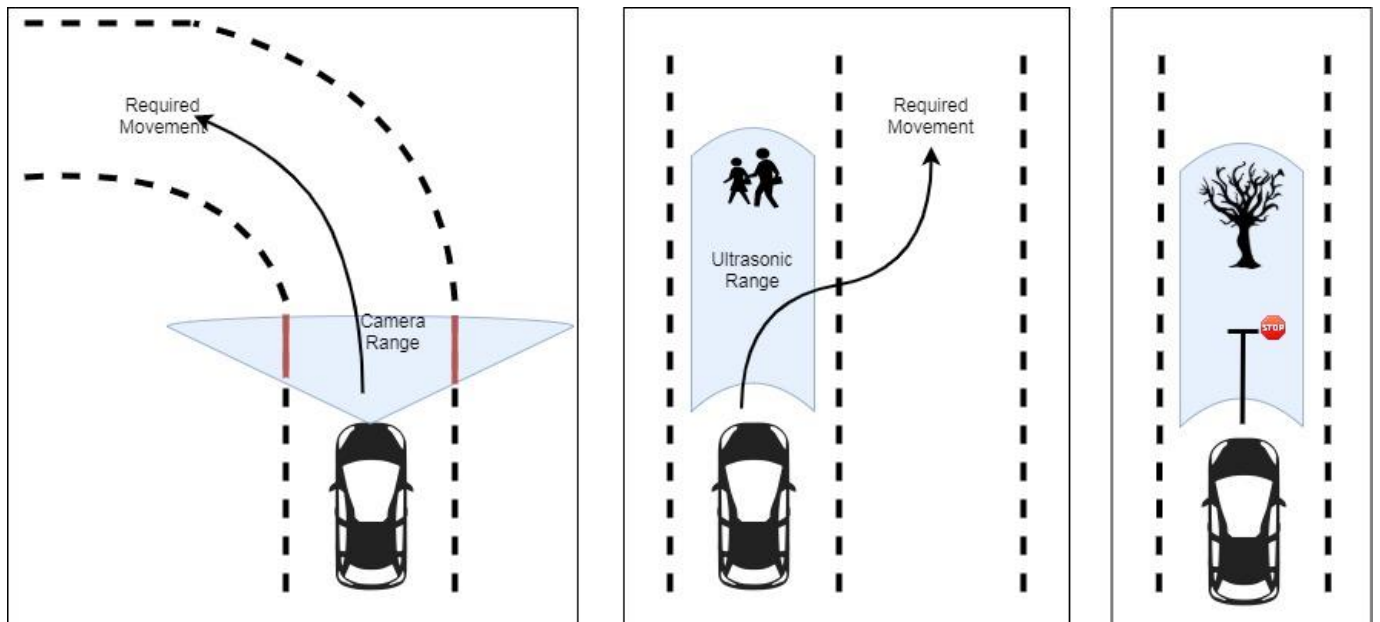


Figure 2: Top down view of what was required of the car to be capable of doing

2.2 Specifications

Firstly, it was required to understand what specification we required from the hardware of this project. It was important that the components were chosen to optimally use the skillsets of each of the members. It is also not recorded on the specifications table but expected that the components will be under the budget cap of 500 Canadian dollars.

Table 3: Specifications table of the initial requirements from different project components

Component	Specification
Movement	<ul style="list-style-type: none">• Bi-directional movement• Light duty• Should operate under the load of the car• Need to steer
Chassis	<ul style="list-style-type: none">• Should handle the weight of all components• Packaging requirements for all the components
Power Supply	<ul style="list-style-type: none">• Can power for the motors, sensors, and the microcontroller continuously for a reasonable amount of time• Light weight• Compact to help with packaging
Sensors	<ul style="list-style-type: none">• Camera with adequate video quality, compatibility with Raspberry Pi• Ultrasonic Sensor with adequate range
Microcontroller	<ul style="list-style-type: none">• WIFI Compatible• High processing capabilities to send video• Compact device for packaging purposes• Heat resistance

2.3 Microcontroller

2.3.1 Raspberry Pi 4

For starters, the group decided to purchase a Raspberry Pi 4 to be integrated as the main logic component of the project. The benefit of using this over a 3rd generation Raspberry Pi is that it offers a better performance and it consists of a faster processor with a clock speed of 1.5GHz, as compared to a 1.4GHz processor found on the previous model (Raspberry Pi 3B+) [1].

Additionally, it supports various RAM configurations varying from 1GB to 4GB, although for the project specifications, only the 1GB RAM was required.

Results from a few benchmarks tests from MagPi magazine show the vast amount of difference between the Memory Bandwidth(bW) Read/Write rates (Figure 3) and between toggling a GPIO pin on various Raspberry Pi models below (Figure 4) [2].

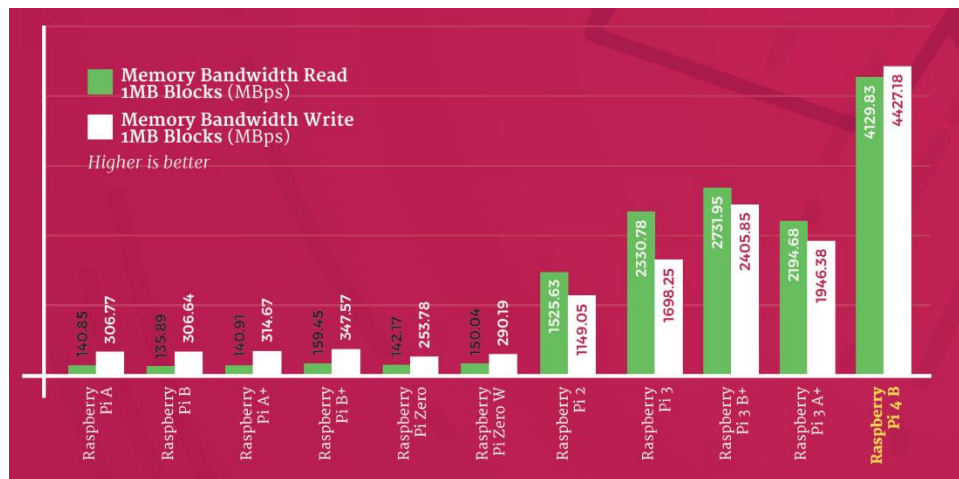


Figure 3: Using RAMspeed tool to measure read/write bW for 1MB blocks (in MPps) [2]

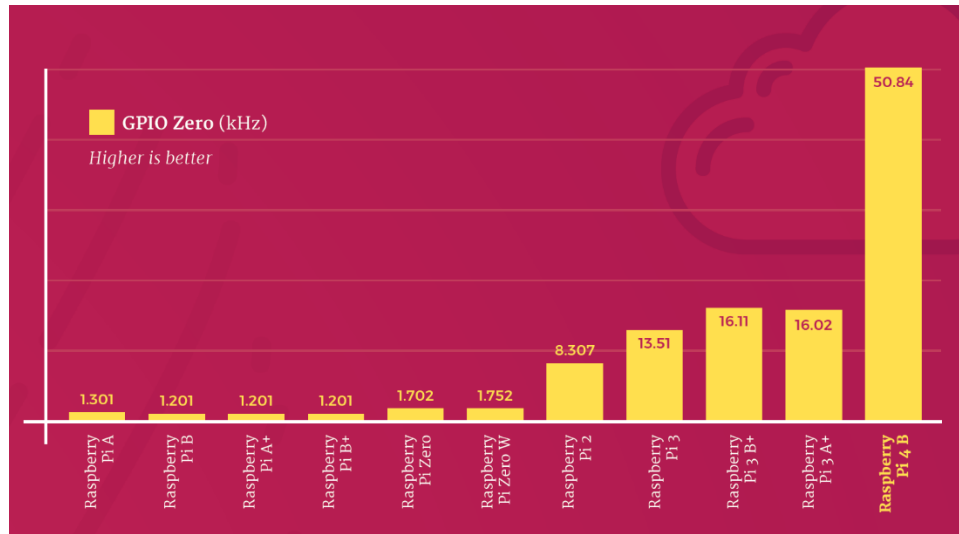


Figure 4: Using Python GPIO Zero library to measure switching rates of GPIO pins (in kHz) [2]

Although irrelevant to the project, the Raspberry Pi 4 also contains Bluetooth 5.0 which has an improved capacity, range, and speed. It's 2 HDMI ports allow the user to run two display motors concurrently. Additionally, it can run a 4K video at about 60 frames-per-second, making it efficient to run media playback with enhanced pixels [1]. These features would've overall helped increase the efficiency of our project by reducing delay times and increasing the time between switching various GPIO pins. In terms of the price, the newest model – Raspberry Pi 4, costs the same as the previous model but contains better features, so the value invested in this component was successfully retrieved.

2.3.2 Raspberry Pi 3b+

Unfortunately, there were compatibility issues that arose while trying to configure the motors using the UCTRONICS software. The software did not have a reliable working release for the operating system, the Raspbian Buster, used by the Raspberry Pi 4. Furthermore, it was impossible to retrofit the Raspberry Pi 4 with an older version of the operating system – Raspbian Stretch due to firmware attribute differences on the Pi 4. Hence, a lower generation Raspberry Pi was needed.

Luckily, the group was able to return the Raspberry Pi 4 and purchase a new Raspberry Pi 3B+. Using the new piece of hardware, most of the compatibility issues were solved and the project continued as desired. The new Pi worked smoothly in terms of connections with the sensors, GPIO pins, and the client/server. As a matter of fact, the overall delays and a restricted throughput still allowed our system to run, and the price of the hardware was exactly the same. Despite of having a reduced number of USB ports and pins (such as SPI and UART) [3], the Raspberry Pi 3B+ integrated perfectly with our system and the interface combined well with the UCTRONICS software. Moreover, having an HDMI port rather than a micro-HDMI port [3] added a slight convenience to the project.

2.3.3 Raspberry Pi 3 B

Another unexpected technical issue arose when the Raspberry Pi 3b+ started overheating within 5 minutes of usage. After a while, it became impossible to test various codes and components as the Pi was no longer capable of running for a long period of time. This issue delayed some timelines and milestones as no tests were implemented. Fortuitously, a friend offered a helping hand by lending their old generation pi – Raspberry Pi 3b. The reason as to why the old Pi started overheating is unknown. It is said to overheat when it is pushed beyond its factory limitations or is provided with a heavy load [4].

As compared to the Raspberry Pi 3b+, the Raspberry Pi 3b also consists of a 1GB LPDDR2 RAM and the same number of ports [5]. Yet, the Raspberry Pi 3b does not include the feature of ‘Power over Ethernet’ (PoE), and the Broadcom is 0.2 GHz slower [5]. The PoE feature was not used in the project as we used a WiFi connection to connect to the Pi. In terms of performance, there is not a great boost visible in real-time situations, so, the Pi did not hinder the project’s performance by a great factor.

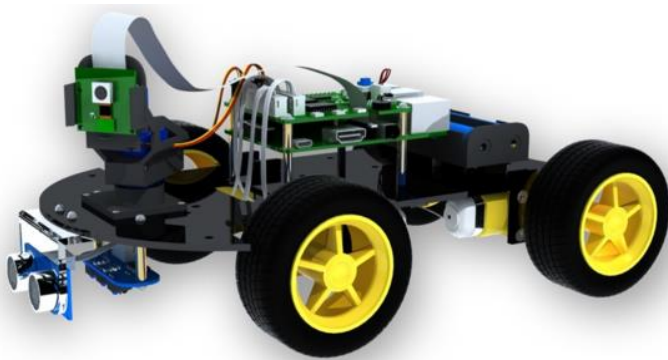
The major physical parts of SCAR were taken from the reverse engineering process of similar vehicle that was purchased. This robot was the UCTRONICS K0073. This car arrived as a self-assemble kit, that we extensively reverse engineered to come up with the final iteration for SCAR. One of the main reasons K0073 was chosen was that this Kit contained hardware suitable to most of the requirements that were outlined in the specifications table in Table 3. The following sections will look at the components of K0073, represented in Figure 5 below, that were used in the final prototype model of SCAR.



Figure 5: Full K0073 assemble able kit by UCTRONICS that SCAR is based on [6]

2.4.1 Chassis and Assembly components

K0073 Chassis components that were borrowed over are an ABS plastic structural frame board, 4 wheels, 4 direct current motors, pair of lithium-ion batteries, mounts for sensors and computers, and fasteners. Figure 6 shows a brief description and an image of the overall assembly components. The structural frame was light weight and sturdy with ample space for planned components. The frame is sold with predrilled holes and extra fasteners to give the user some freedom in the assembly of the car.



Model	K0073
Platform	Raspberry Pi
Release Date	8/1/2018
Sales Channel	Amazon, uctronics.com

Figure 6: K0073 Chassis that was used for SCAR [7]

2.4.2 Pi Camera

Pi Camera V2 was also provided with the K0073 kit. The Pi Camera was chosen for SCAR project because of its compatibility with all the Raspberry Pi microcontroller models. It connects to the RPi through a 15cm ribbon cable, as shown in Figure 7, and the RPi microcontrollers contain the drivers need to operate the Pi Camera module. This camera has an 8 Megapixel sensor and it is capable of 720p video at 60 frames per second.

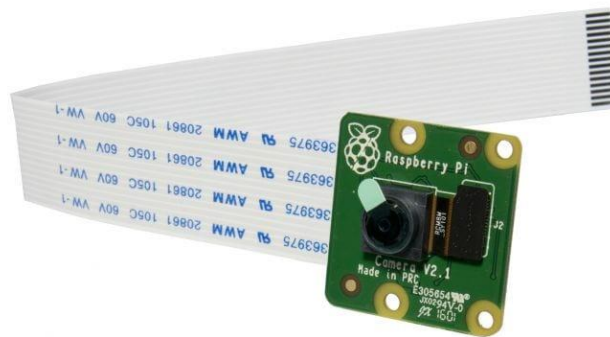


Figure 7: Pi Camera V2 module [8]

2.4.3 Motor Controllers

2.4.3.a Initial K0073 driver

K0073 robot was also delivered with a driver board to connect the hardware components to the Raspberry Pi using GPIO pins in a safe manner. This board played a key role in this robot as it was the intermediary between the RPi, Power Supply, and the other hardware devices being used. It did this through an integrated motor driver chip. This chip was a Texas Instruments L293DD motor driver chip. It is a push-pull four channel motor driver capable of 600mA of peak output current in each channel [9]. This was more than adequate for the ceiling current of 250mA that the geared DC motors would operate under [10]. Unfortunately, due to the closed-source nature of K0073, much of the software and pin programming documentation was unavailable to base SCAR upon this GPIO board. Therefore, another approach had to be made regarding motor controlling and intermediating sensors.

2.4.3.b Post Presentation Motor Controller

After the oral presentation of this project with the project supervisor, a radical change of approach was chosen regarding the motor design philosophy SCAR would be following. Starting with the phasing out process of the K0073 driver board. This board was taken out of use due to the complication that arrived with reverse engineering a piece of hardware that did not contain any documentation. Instead, two L293D push-pull two channel chips were used to control the movement of SCAR. As seen in Figure 8, the motor has 16 pins which are used to mount onto a breadboard.

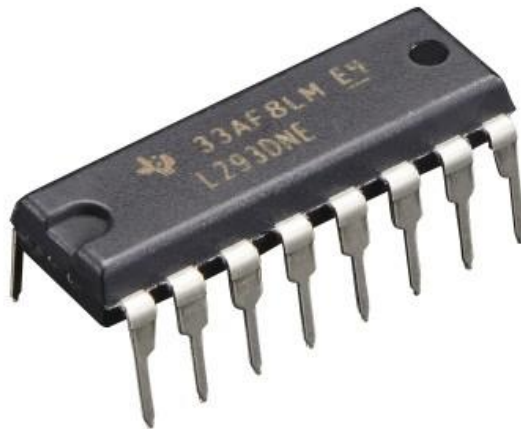


Figure 8: Texas Instruments L293D Dual H-bridge Motor Driver chip [11]

This motor driver was a 2-channel chip with 600mA peak output current per channel [12]. Since we had four motors, two of L293D chips were resourced in the final SCAR prototype model. The wiring and integration of this chip will be discussed in detail in later sections of this report.

2.4.4 DC Motors

The DC motors (Figure 9) that were provided with the K0073 kit were a set of TT motors geared 1:48. These motors were bi-directional with speed variability at between 3-6 volts. They have torque figures between 0.15-0.60Nm [10]. These motors proved themselves to be the movement components of SCAR due to their performance with respect to size and power consumption.



Figure 9: Similar Geared DC Motor like this was used to move SCAR [10]

2.4.5 Ultrasonic Sensor

An ultrasonic sensor was also resourced for SCAR. This sensor was the HC-SR04 model ultrasonic sensor, as can be seen in Figure 10. The module contains an ultrasonic transmitter and a receiver (trigger and echo) that the user can utilize to get a working range of 2-400cm with $\pm 3\text{mm}$ accuracy. When the trigger pin is activated, this module will produce an 8-cycle sonic burst with the transmitter, and the receiver will listen for an echo [13]. The application in scar is a similar system to forward collision radar detection that most modern cars come with as a safety feature.



Figure 10: HC-SR04 Ultrasonic sensor module used in SCAR [13]

2.4.6 Power Supply

One of the main requirements of SCAR was to be wireless and mobile, therefore, having adequate power during operational time was a necessity. The SCAR model's power supply consisted of two 2200mAh 18650 Lithium Ion batteries, and one 10400mAh USB power bank manufactured by Black Web, as displayed in the Figure 11 below.



Figure 11: The power supply components of SCAR [14] [15]

The L-Ion batteries supplied 3.7V to the motor controller system to power the DC Motors. The Black Web power bank was used to provide 5V power to the Raspberry Pi, Ultrasonic sensor, and Logic power for the motor controller system. The power bank was not ideal for SCAR as it inherited much of the weight and space within the vehicle. In terms of power, SCAR was limited by the L-Ion batteries (Usage was sufficient), so in terms of efficiency it would have been reasonable to use a smaller and lighter power bank that would provide similar usage to the L-Ion batteries.

2.5 LAN Router

A Network was required for the SCAR vehicle and the server computer to communicate. For modeling purposes, a router was acquired and a local area network (LAN) was established through it. The router that was used to do this was a D-Link DIR-880L model router (Figure 12). This router supports dual-band wireless connection which together will provide up to 1.9 GB per second of wireless speeds [16]. This router was not sourced especially for this project but it's a device that was available and adapted for use in the SCAR project.



Figure 12: D-Link DIR880L Model Router which established the LAN for SCAR vehicle and server [16]

2.6 Computer used as Server



Figure 13: The computer used as a server in the system

Figure 13 above shows the computer used to perform the image processing and command extraction in the system. It is equipped with an Intel Core i5-9300H, 8GB of RAM, 512GB SSD, and a NVIDIA GTX 1650 graphics card. All latency in the algorithm was calculated on this model and may be subject to change on other devices.

3 Assembly

3.1 System Overview

The following (Figure 14) a System-level diagram of the various systems that were integrated into the use of SCAR Project.

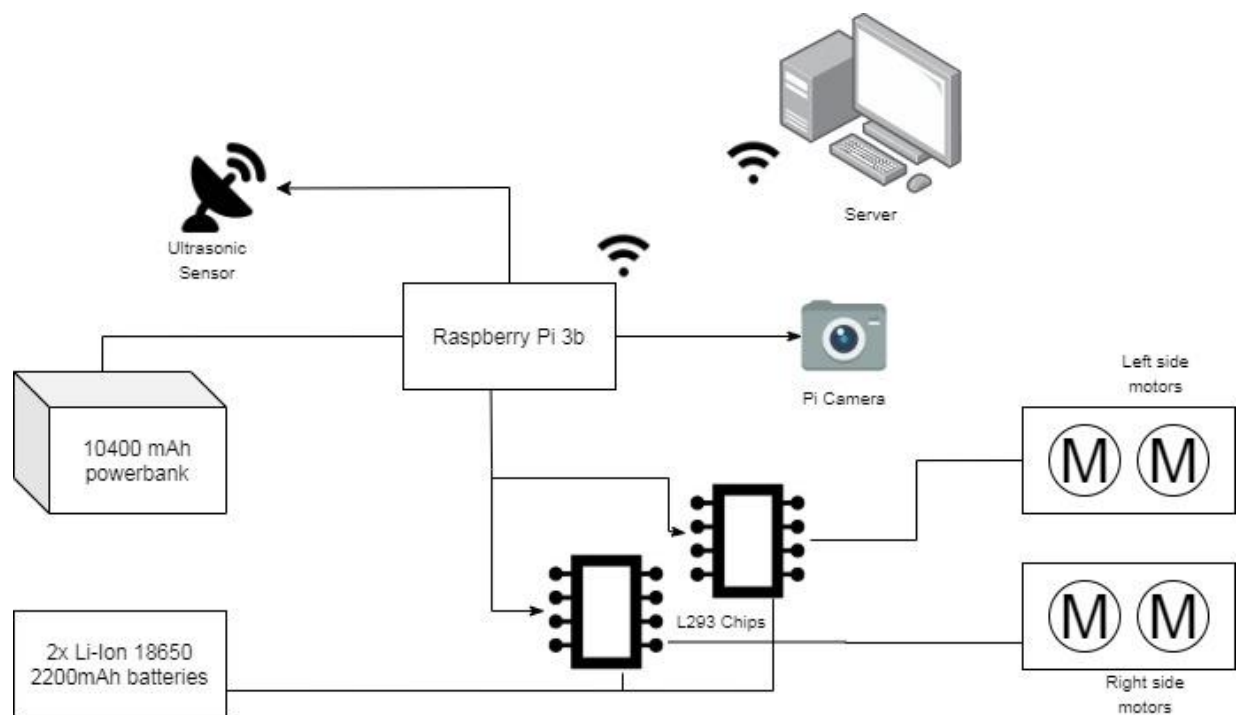


Figure 14: Overview System Diagram of SCAR Project

SCAR's movement was manipulated by four bidirectional DC motors that steered SCAR in a tank-steering style. The sensory information was fed to SCAR through a Camera and an Ultrasonic sensor. The information was sent to the server-end device that made the decisions for the robot through a wireless Local Area Network and a response was sent back to SCAR on the same network. The entire system is split into their sections.

3.2 Raspberry Pi Configuration

3.2.1 Raspi-Config

The Raspbian operating systems are a Linux family operating system based on the Debian distribution. Therefore, the appropriate Linux commands can be followed through the computer's terminal to configure it. However, the Raspbian Operating systems come preinstalled with the Raspi-Config tool that can be used to configure the computer. This feature lets the operator modify bootup settings, security settings, camera settings, network settings, etc. on their Raspberry Pi. Initiating Raspi-config was the first step that we took to configure the Raspberry Pi after bootup. This was done by the following Linux Command.

```
sudo raspi-config
```

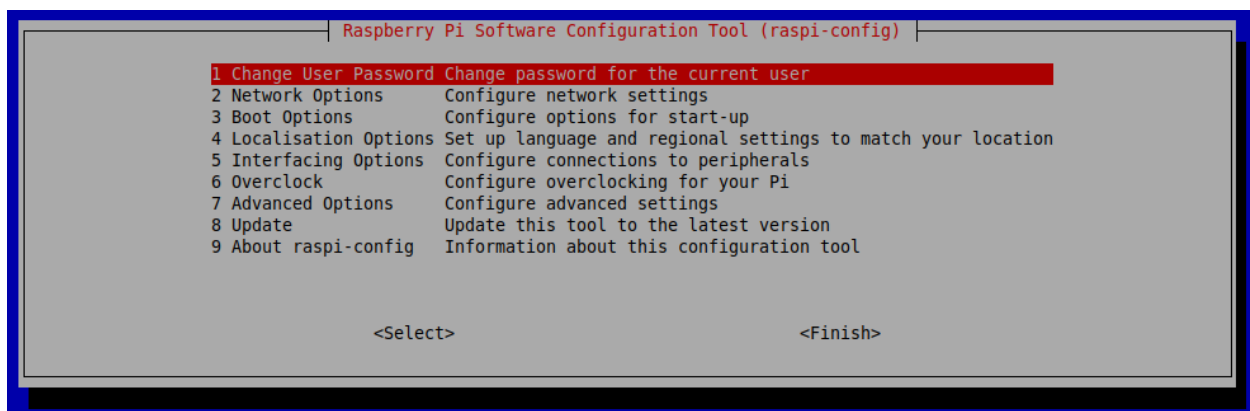


Figure 15: The menu that is prompted when Raspi-config command is executed [17]

Figure 15 above shows the settings that can be accessed through Raspi-Config tool. The components that were configured through this tool for SCAR were the following: enabling Secure Shell (SSH), connecting to Wi-Fi network, and enabling the use of the Pi Camera on the RPi.

3.2.2 Remote Command of Raspberry Pi

SCAR is a mobile device, and the Raspberry Pi aboard it would always also be mobile. That meant, there needed to be a way to access the computer aboard SCAR remotely. Using the aforementioned configuration tool, one of the settings that was enabled on the Raspberry Pi was the SSH feature. With this feature, the access to the Raspberry Pi was made available by connecting it to the host computer and using its keyboard, mouse, and the monitor to manipulate the RPi. To do this, the host computer needed a program capable of Vital Network Computing (VNC). For SCAR's use, VNC Viewer program was chosen.

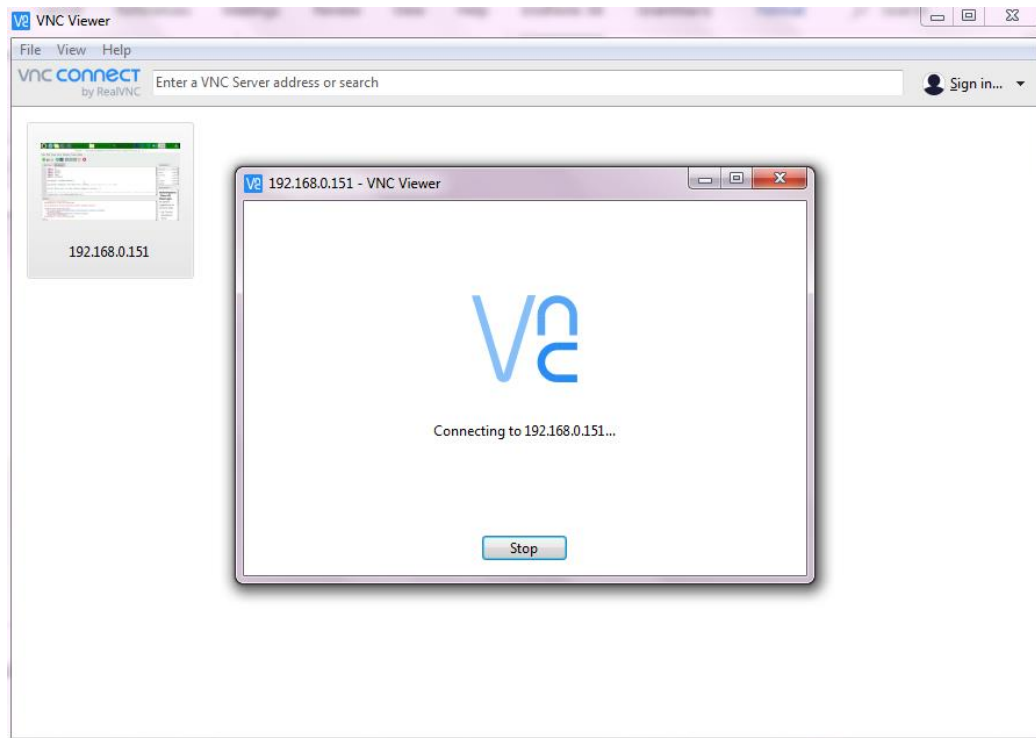


Figure 16: Screenshot of VNC Viewer attempting to connect to the RPi aboard SCAR

VNC Viewer required the IP of the Raspberry Pi to initialize the remote connection. This was easily acquirable through the network that the Raspberry Pi and the Host computer automatically connected to.

3.3 Post Oral Presentation Layout

3.3.1 Final circuit diagram

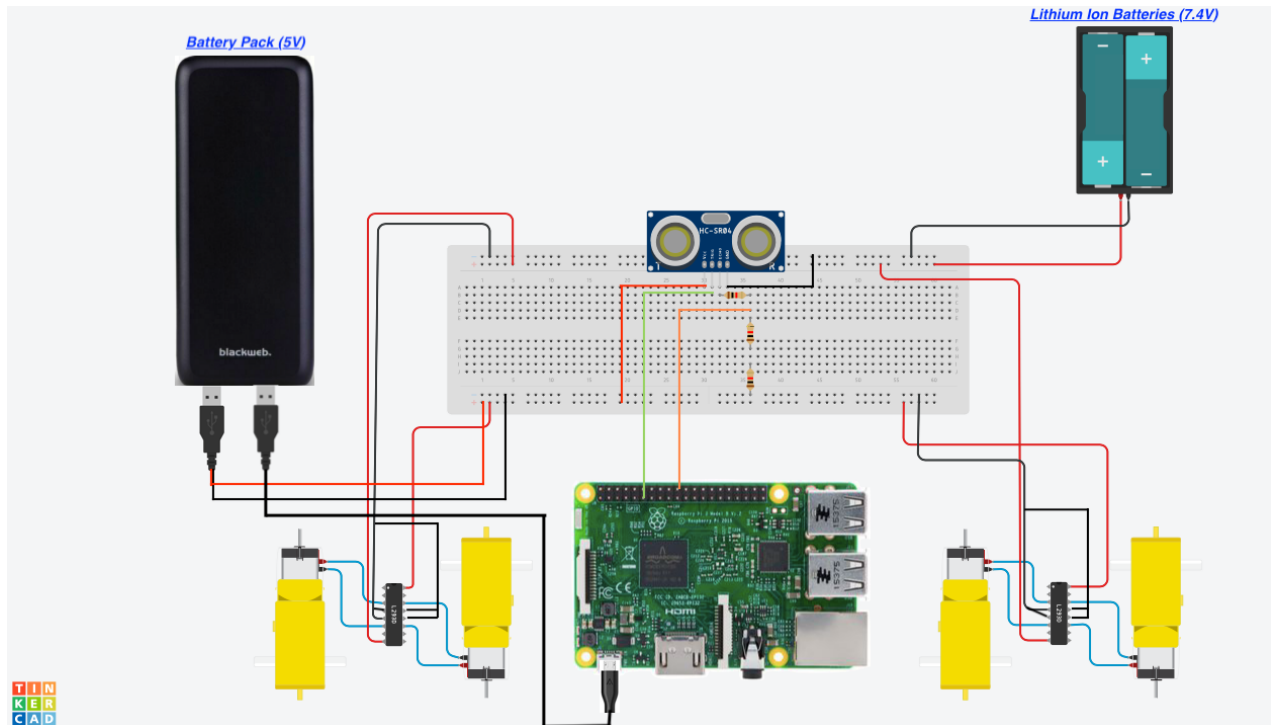


Figure 17: The overall circuit diagram

The figure #? shows an overview of the final components used in the project, along with the basic connections. The two H-bridges on the bottom are mounted on the breadboard in reality but are shown separately on the diagram for more clarity. The battery pack supplies power of 5V to the Raspberry Pi 3b and the ultrasonic sensor, while two Lithium Ion batteries in series provide a total of 7.4V, which powers the motors. A more detailed configuration of how each pin on each H-Bridge is connected to the Raspberry Pi is shown below in the detailed circuit diagram.

3.3.2 Detailed circuit diagram

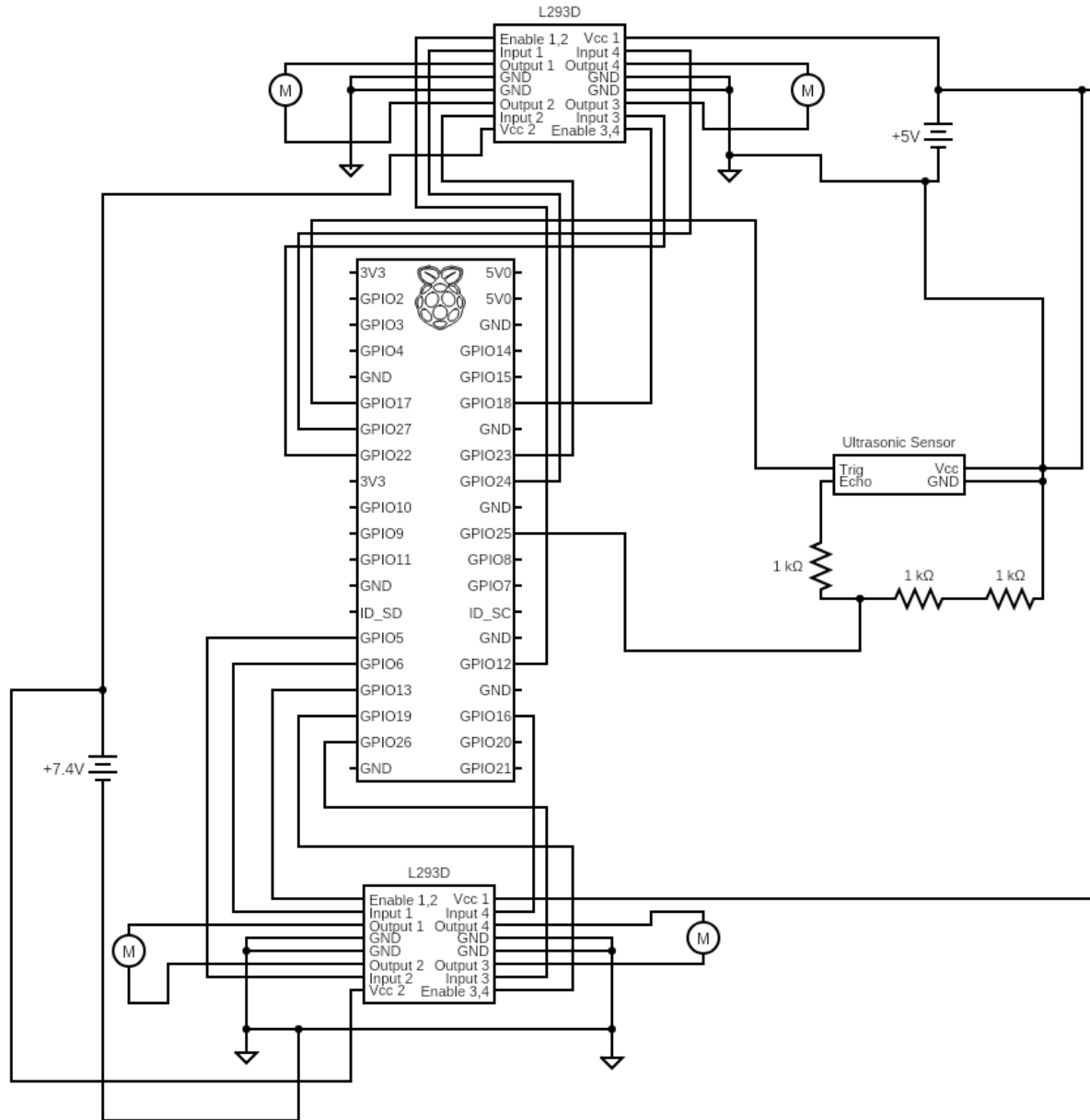


Figure 18: Figure showing detailed circuit diagram on the modelled car

Figure 18 shows the detailed circuit diagram of the modelled car system implemented for the project. It shows how each of the motors is connected to the L293D dual H-bridge. Since a dual H-bridge was implemented, each side of the H-bridge was used for a single motor. There were two battery packs integrated in the system for the functionality of the motors. The H-bridge uses the 'Vcc 2' port to power the motors, and in order to rotate these motors, a higher voltage (7.4V) was required. The enable pin and the input pins for the H-bridge were connected to the GPIO pins on the Raspberry Pi to be programmed by the user. Each motor has its own PWM making functionality of the modelled car diverse. This was also repeated for the rear wheels using another H-bridge. The battery pack also powered the H-bridge at 'Vcc 1', which serves as the power required to run the circuitry of the motor driver. The second port of the battery pack was used to power the Raspberry Pi. Finally, the ultrasonic sensor was connected to the 5V battery pack. To avoid damaging the GPIO pins on the Raspberry Pi's, a voltage divider using resistors was created to reduce the voltage read-in on the echo pin.

Appendix A and Appendix B, attached to the end of the report, show the pinout configuration of the Raspberry Pi 3B as well as the L293D H-Bridge.

4 Flow of Information

The following subsections show the flow of information between the server and client. Details about any technical implementation of the system is discussed in later sections of the report.

4.1 Sequence Diagrams

The following figures represent the sequence of events in the entire system, and within just the Raspberry Pi respectively.

4.1.1 Sequence Diagram of the entire system

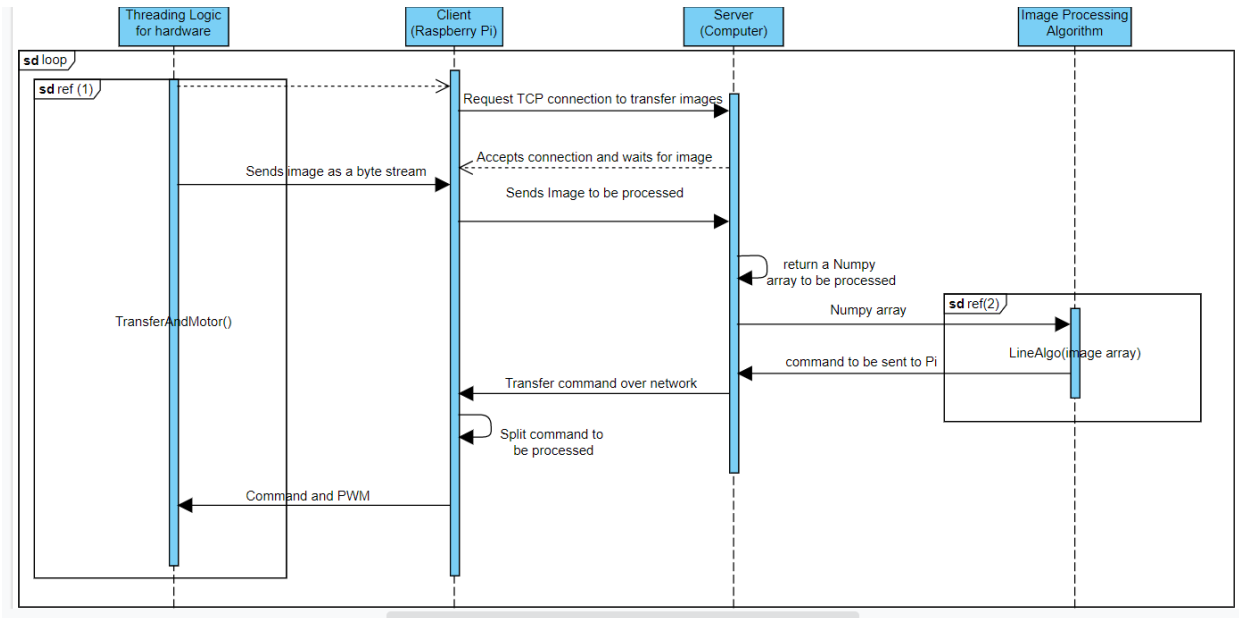


Figure 19: Sequence diagram of the entire system

Figure 19 shows the sequence diagram of the entire system. The figure is further broken into ref (1) and ref (2), as Figure 20 and Figure 21, which shows the logic implemented on the Raspberry Pi and the server-side algorithm. The system is in an infinite loop broken by 2 possibilities. The first possibility is the server detects no lines in the image and terminates the program by sending command 7. The other possibility is using a keyboard interrupt on the raspberry pi.

The sequence starts with the raspberry pi requesting a Transmission Control Protocol (TCP) connection to the server to start transferring an image for processing. Once established, the image is sent as a byte stream over the TCP/IP protocol. The server receives the image and puts it into a NumPy array to be passed as a parameter to the main function (elaborated in Figure 3). After this the image is processed, which is discussed in a subsequent section in the report, and a command to move the motors with the PWM is returned to the main function to be transferred over the network. After the command is received, the client program executes the command by sending signals using General Purpose Input Output (GPIO) pins on the raspberry pi.

4.1.2 Sequence Diagram on the Raspberry Pi Side

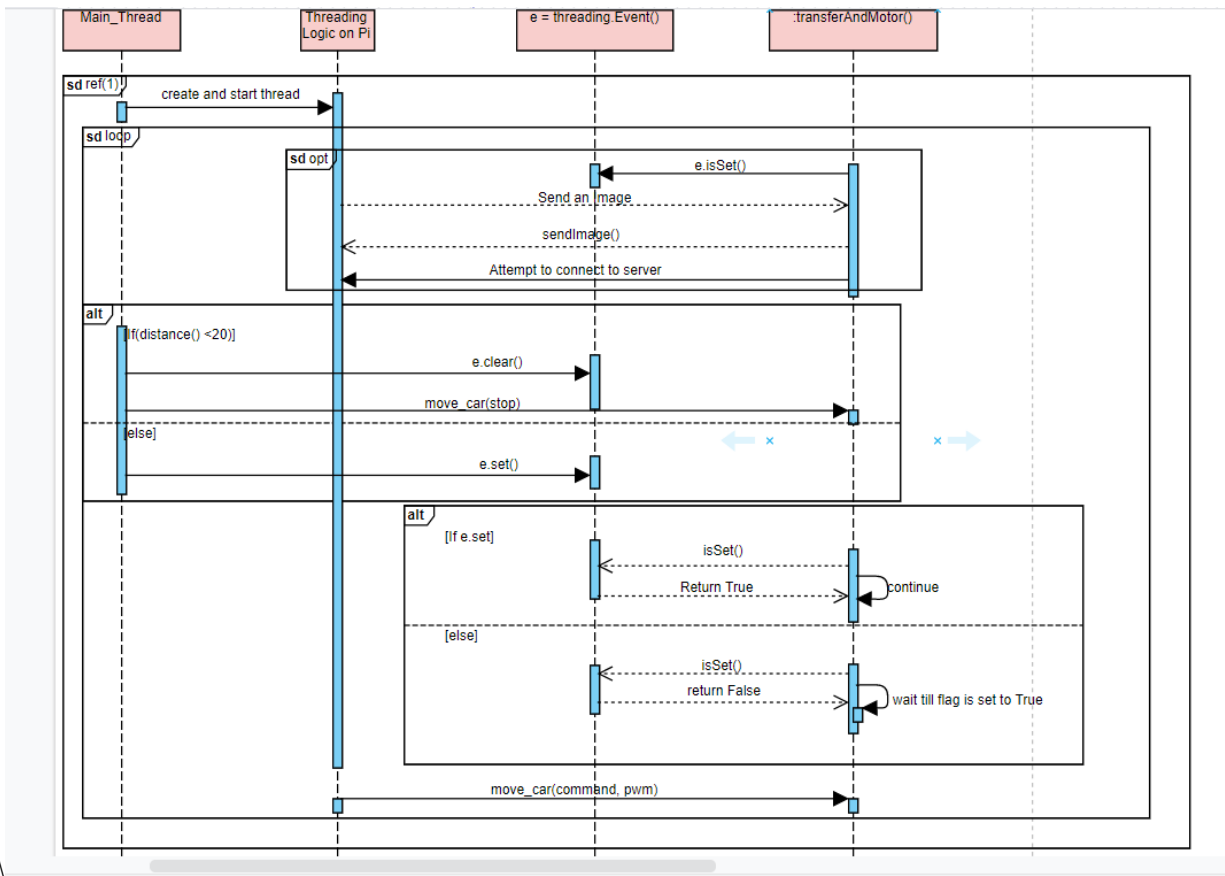


Figure 20: Sequence Diagram of the Pi side

Figure 20 shows the logic implemented on the raspberry pi. Two threads run concurrently in this system. The main thread which is executing the ultrasonic sensor code and a child thread called `transferAndMotor()` thread running the TCP/IP connection along with the motor function. The sequence begins by creating a thread and starting it. This thread takes care of communication with the server. It exists in a loop constantly polling a flag defined as “e”. If the flag is set to “True” it continues the normal process of sending an image and receiving a command. Else if the flag is “False” it essentially pauses until it sets back to “True”. The flag is controlled by the ultrasonic sensor constantly checking for objects less than 20 centimeters away. A stop function is executed if an object is detected in the range. Else the flag is set to true and the command sent from the server end is executed.

4.1.3 Sequence Diagram on the Server Side:

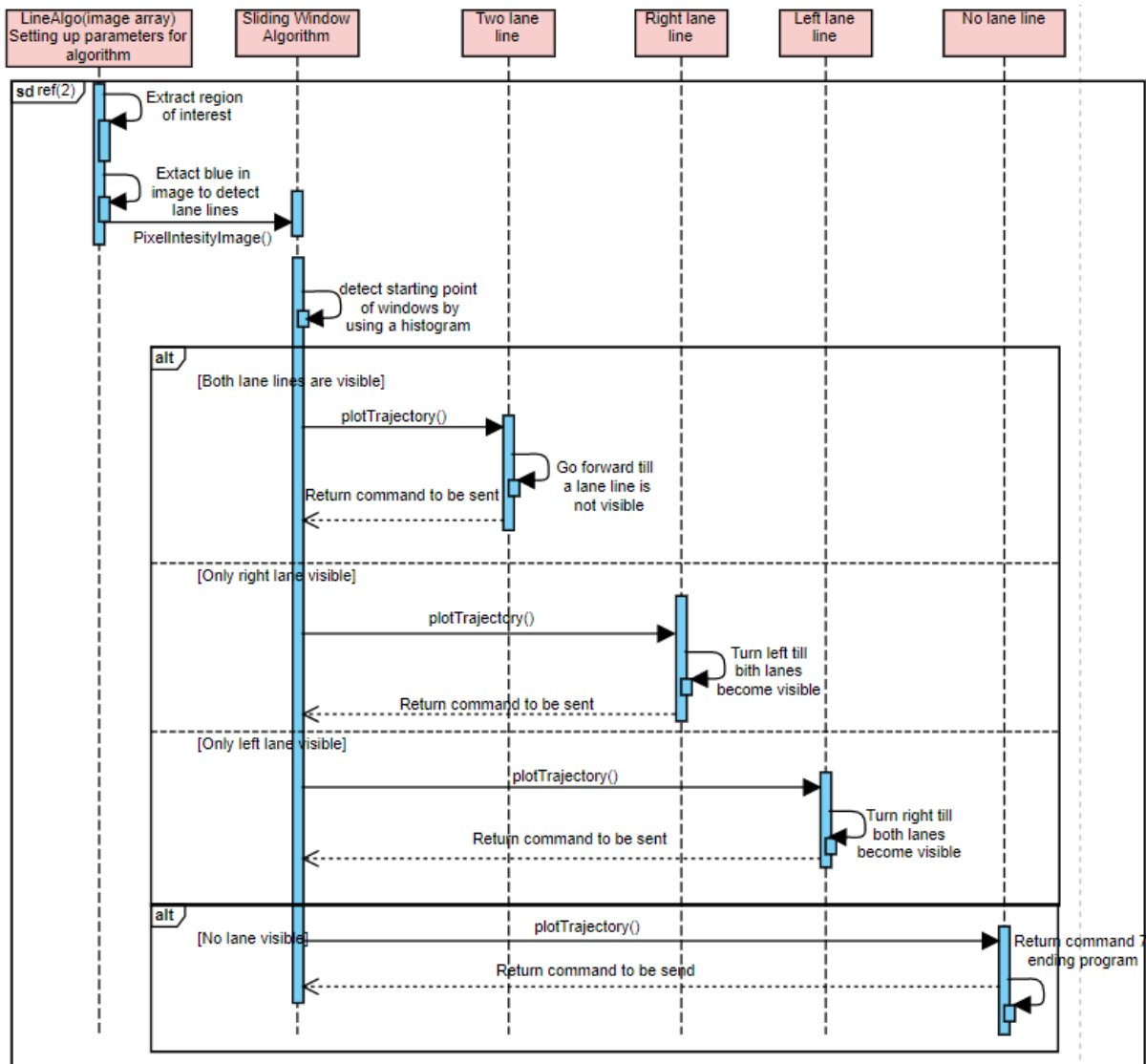


Figure 21: Sequence Diagram of the Server Side

Figure 21 shows the flow of information on the server end. After receiving a NumPy array of the image, the image is cropped and reshaped to only plot the region of interest. A few more manipulations occur before the blue tape lane lines have been detected - this has been talked about at a later section in this report. Once this is done, a histogram is plotted to determine the starting position of the sliding window algorithm. The algorithm considers four conditions. If two lane lines are detected, give a forward command. Secondly if only one lane line is detected, the algorithm computes if it is a left lane line, giving a right command. Thirdly, if it detects a right lane line, the algorithm gives a left command. Finally, if no lane lines are visible, the algorithm returns command 7, terminating both the client and server programs.

5 Client-End Programming

The client end represents the modelled car, as described in the previous section, along with the microcontroller mounted on the chassis. Pictures of the model car is shown in Appendix A for further clarifications. The Raspberry Pi is at the heart of the modelled car as it controls the signals sent to the H-bridge motor driver, inputs from the server through a network, and finally the output input of the ultrasonic sensor.

5.1 Motor Controls and Chip Used

The following subsections discuss the implementation of the L293D Dual H-bridge and the ways the pins were programmed to get a desired output.

5.1.1 Motor Driver Chip (Dual L293D H-Bridge)

A pair of the dual L293D chip is what controls all 4 motors. Figure 22 below shows the pinout of the L293D dual H-bridge and figure 18 shows how it is connected to the Pi and the respective motors. The Raspberry Pi pinout, as seen in Appendix A, shows the available pins that can be used. Since the hardware PWM was given on the Raspberry Pi, it was used to program the 4 enable pins on the H-bridge (2 enable pins per H-bridge). PWM and its utilization in the project is talked about in a later section of this report. The dual H-bridge can be viewed by cutting the chip vertically - each side of this chip controls an independent motor. Since this means each motor has 2 half H-bridges, it can be used to control the motor speed as a bi-directional motor [18]. Using the command `GPIO.setmode(GPIO.board)`, the GPIO pins on the Raspberry Pi can be configured similarly to the circuit diagram in figure #.

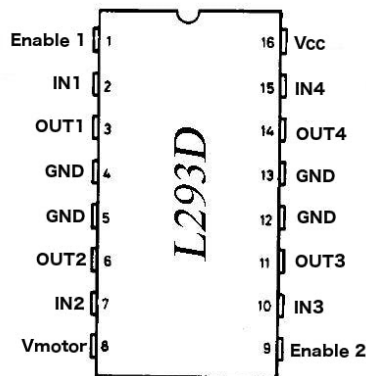


Figure 22: The L293D motor driver chip pinout [18]

5.1.2 Motor Control Logic

The code snippet shown in figure 23 below shows the setting of input pins of the dual H-bridge. As seen, depending on which pin is set as high or low, the motors movement would either be forward or backward. Similar functions were implemented for the other 3 motors on the car.

```
| def front_right():  
|     GPIO.output(29, True)  
|     GPIO.output(31, False) # Motor 1 move forward (front right)  
|  
|  
|  
|  
| def front_right_reverse():  
|     GPIO.output(29, False)  
|     GPIO.output(31, True) # Motor 1 move backward (front right)  
|
```

Figure 23: The setting of H-bridge pins to determine direction

The directional movement of the car is based on the same idea as the way tanks move. Figure 24 below shows the basic instruction on how to move the modelled vehicle. For example, to move right we turn the left front and rear wheels forward, and the right front and rear backward.

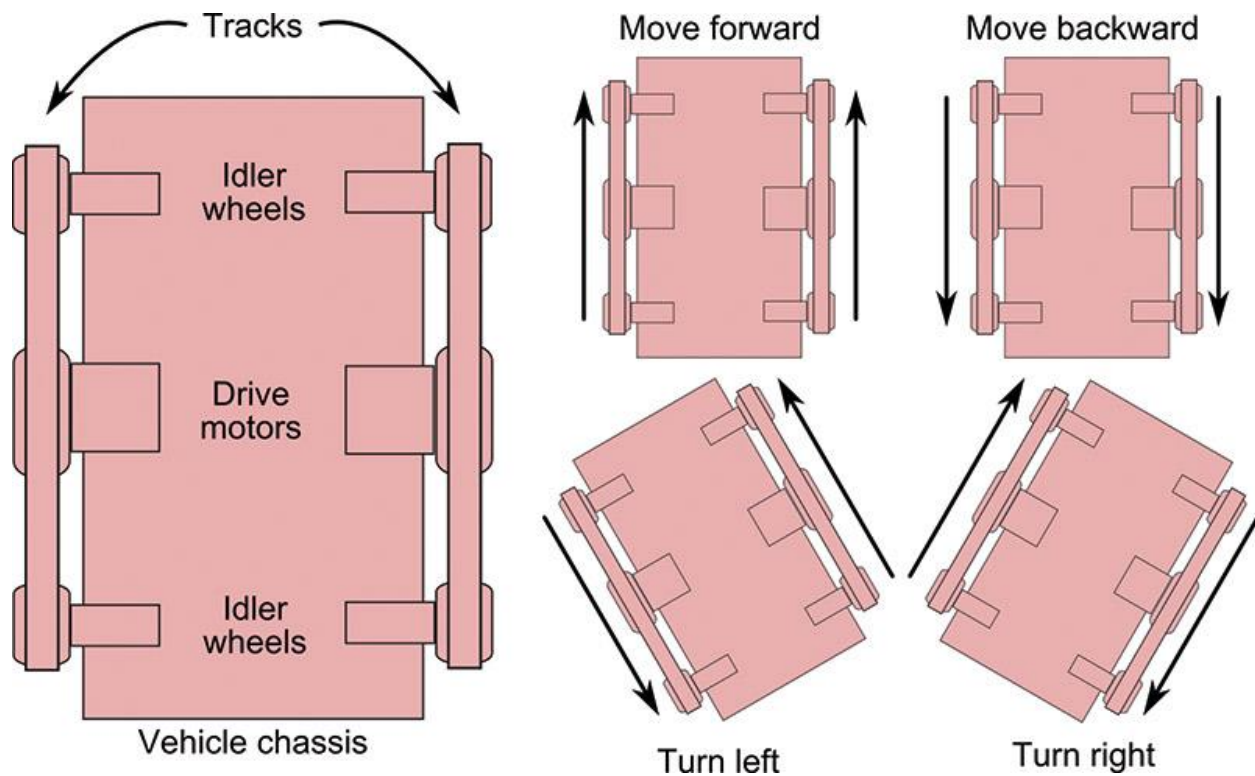


Figure 24: Example of tank steering implemented in SCAR [19]

In the above example of moving right, the code snippet in figure 25 illustrates how this is carried out by the modelled car. Since we do not have the same setup as the way a tank wheel is set up, it was found that the rear right wheel had to move 1.4 times in order to make the right turn. This number was achieved by trial and error to result in an acceptable turning radius.

```

if direction == 2: # right
    front_left();
    duty_cycle(2, speed)
    rear_left();
    duty_cycle(3, speed)
    front_right_reverse();
    duty_cycle(1, speed)
    rear_right_reverse();
    duty_cycle(4, speed * 1.4)

```

Figure 25: Code snippet of a turn

5.2 Pulse Width Modulation (PWM)

Pulse Width Modulation is directly related to the speed of each motor. The idea of PWM is to toggle between on and off on the enable pin of the H-bridge fast enough to imitate speed. The PWM scale is on a percentage scale, defining the frequency of the on/off signal being set to the enable pin. Figure 26 below shows different duty cycles and how they are broken from being high and low. The model car weighed about 900 grams which was heavy. This was mainly due to the size of the battery pack used. Hence, the car motors struggled to move with any duty cycle less than 43%. The remainder of the project uses a set predefined PWM of 50% for all commands to ensure reliable movement of the modelled car.

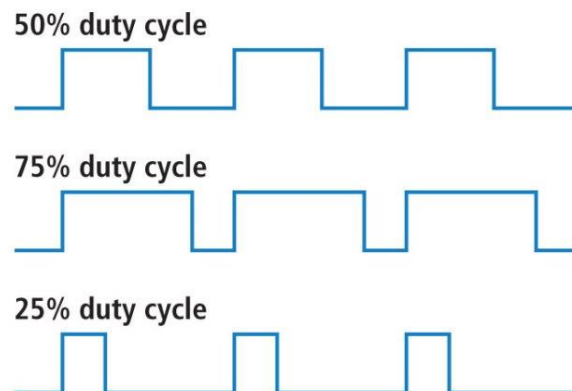


Figure 26: Different PWM cycles [20]

5.3 Raspberry Pi Camera and Ultrasonic Sensor

The Raspberry Pi Camera as mentioned before serves the purpose of our systems eye. The resolution required for the system was 500×480 pixels. This was determined by taking a picture and defining acceptable parameters. Since the lane lines used in the environment fit in this resolution it was ideal to use this as the parameters. An obstacle faced while capturing a picture was that the camera had a small delay before it could retake another picture. This delay was found to be about 0.15 seconds. To ensure proper and systematic arrival of pictures, another 0.1 second delay was injected into the code to account for any fluctuations in taking the image. An explanation of the round time latency is discussed in a later section of the report.

The ultrasonic sensor as previously mentioned is a key component in the system to avoid collision with any objects directly in front of the modelled car. Human beings can only hear frequencies in the “Acoustic Range”. The ultrasonic sensor uses “Ultrasound” which is a frequency higher than that of the Acoustic Range. As mentioned previously the trigger module sends these ultrasounds. The echo module then waits for the sound to reflect off an object and then detects it. The time taken for the sound wave to travel from the trigger module to the echo module is what is used to calculate distance [21]. The code snippet shown in figure 27 below shows the simple implementation of the ultrasonic sensor. It sends a sound wave for 0.00001 seconds and then starts a timer. As soon as the echo pin becomes high the timer stops. Mathematical manipulation is then conducted to extract the distance in centimeters.

```

def distance():
    GPIO.output(trig, True)
    time.sleep(0.00001)
    GPIO.output(trig, False)

    while GPIO.input(echo) == 0:
        pulse_start = time.time()

    while GPIO.input(echo) == 1:
        pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    d = pulse_duration * 17150
    d = round(d, 2)
    return d

```

Figure 27: The method used to calculate distance

For the purpose of this system, the distance () method was called repeatedly with a 0.1 second delay between calls. The system would check if an object is within 20 cm of the car. If it is, the car stops all motors and runs an interrupt which is talked about in the next section.

5.4 Multi-threading in the System

This section discusses the implementation of two threads and how they function. The first thread which is in the main program is the ultrasonic thread. This thread constantly checks the ultrasonic sensor to make sure there are no objects within 20 cm of the model car. Figure 28 below shows a code snippet outlining the main thread. Since python has no predefined way to implement an interrupt, the system used an event to outline changes in the system. This created an event driven interrupt program where “e=threading.Event()” defines the event.

```
if __name__ == '__main__':
    e = threading.Event()
    t = Thread(name = 'Motor',target=transferAndMotor, args=(e,) )
    t.start()
    try:
        while (true):
            dist = distance()
            time.sleep(0.2)
            if dist < 20:
                e.clear()
                stop()
                disable_pwm()
                dist = distance()
            else:
                e.set()
    except KeyboardInterrupt:
        t.join()
```

Figure 28: Threaded functionality of the system

The ultrasonic sensor is in control of the functionality of the system. If an object is detected the system sets the event flag to false by doing “e.clear()”. If no object is detected the ultrasonic thread sets the event flag to true and continues polling for objects. The other thread in the sequence deals with sending the image and receiving a command to execute. The command received is then split and the directional command is executed. This thread only runs “while e.set” is true shown in figure 29 below, hence creating the interrupt-based threading system.

```
try:
    while e.set:

        sendImage()
        bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
        disable_pwm()

        message = bytesAddressPair[0]
        address = bytesAddressPair[1]

        clientMsg = message.decode()
        print(clientMsg)
        clientIP = "Client IP Address:{}".format(address)
```

Figure 29: The child thread and how interrupts handles it

6 Server-end Programming

6.1 Initial Set up

After receiving the image, before extracting any commands for the motor direction, the image needs to be processed. This is done by converting the image into a 3-dimensional array, since the image is in RGB. After this, the image can be manipulated and processed as desired. The environment was set up with blue tape indicating the lane lines.

6.2 Process for Detecting Path

6.2.1 Camera Calibration

Camera calibration is used to determine the location of the camera, measure the size of objects in the camera, and fix the lens distortion. Distortion is caused from the curved shape of the lens, and in some cases, due to the background light hitting the camera. In our project, the Raspberry Pi camera has a frame of 500 by 480 pixels. There are two kinds of distortion present: radial and tangential distortion [22]:

- Radial Distortion occurs when the image appears to be curved at the edges or at the optical center, due to the bending of light curves at various angles [22]. The figure below shows the difference in a negative radial distortion and a positive radial distortion. The negative radial distortion appears to be a convex mirror, while the positive radial distortion seems like a concave mirror.

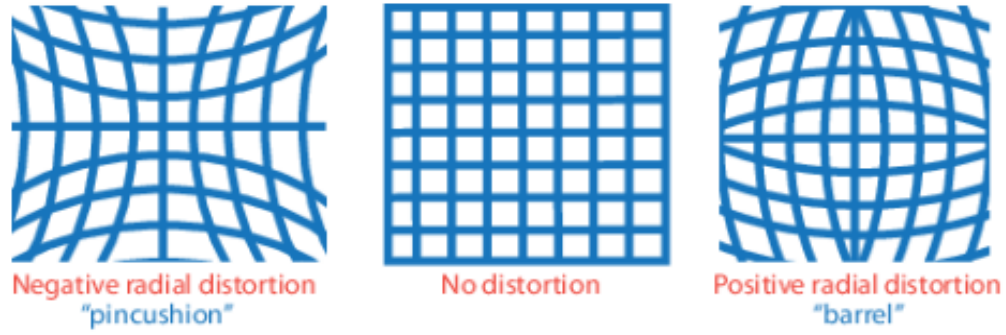


Figure 30: Radial Distortion caused due to light rays bending at various wavelengths as it hits the camera [22]

- Tangential Distortion takes place when the camera lens is not directly parallel to the image plane and some areas on the image appear to be closer than they are. [23]

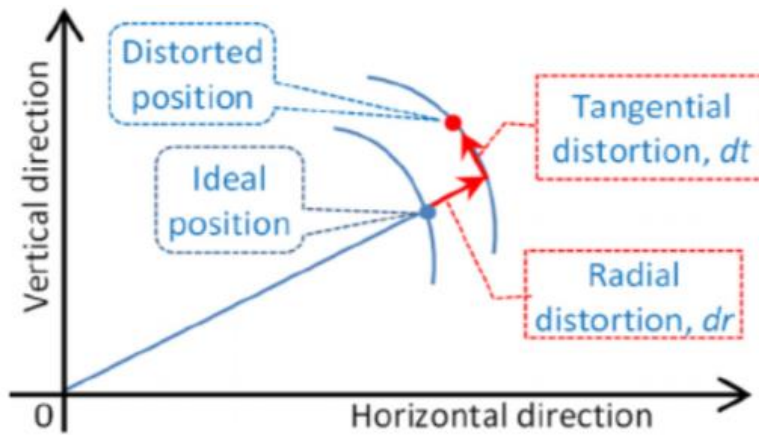


Figure 31: Graphical Representation of radial and tangential distortion on a 2D plane.

As seen in the Figure 31 above, distortion displaces the ideal position, resulting in an incorrect or a distorted image. A camera matrix is a 4-by-3 matrix representing the camera parameters, that helps map a real-world frame onto an image plane. To undistort the image various parameters are required so the calibration algorithm can calculate the camera matrix. Firstly, 5 distortion coefficients are needed – k_1 , k_2 , k_3 (radial distortion coefficients) p_1 , and p_2 (tangential distortion coefficients) [22]. Next, extrinsic parameters, such as rotation and translation vectors,

are used to translate to the camera's coordinates system. Lastly, intrinsic parameters like focal length, optical center, etc. are used to map the camera coordinates onto the image plane.

To achieve camera calibration in our project, we used a picture of a chess board. The idea behind this was that the chess board is a well-defined figure and helps calibrate the camera as the pictures can be captured from a variety of angles. Figure 32 below shows a few examples of the various pictures that were taken to train our algorithm.



Figure 32: Chessboard images taken by the Pi camera for calibration

Once the pictures have been taken, they are fed into the algorithm to find the various parameters. Then, the distorted pictures are automatically selected and are undistorted, using a remapping function. This helps calibrate the camera, so the points are projected properly.

6.2.2 Perspective Transformation and Warp Perspective

A perspective transformation is an essential component in the way the server computed lane lines in the image. Looking at an image, as depicted in figure 33 below, the system views the curves with an undesired vantage point since it does not give good information about the path. It would be ideal if the image was taken from above to provide a “birds eye view” of the image [24]. The `getPerspectiveTransform()` and `warpPerspective()` function, supported by cv2 libraries, help attain this change in perspective. Then converting it to an image seen in figure # to create an aerial view. It can then be assumed that the lane lines are on a flat 2D surface and can simply draw a curve using the `polyfit` function over the lane lines. The first argument of the perspective transformation function is the source points, which is a set of 4 non-collinear points. The second argument that the perspective transformation takes is the destination points which are the 4 corners of the original image. It then returns a perspective matrix transformation of the vectors [25]. This matrix is then used by `warpPerspective()` to apply the perspective transformation to an image as seen in figure 34.

```
[3] img = mpimg.imread("1.jpeg")
print(type(img))
src = np.float32([[0, 100], [500, 100], [500, 300], [0, 300]])
dst = np.array([[0, img.shape[0]], [0, 0], [500, 0], [500, img.shape[0]]], np.float32)
def perspTransform(src, dst):
    M = cv2.getPerspectiveTransform(src, dst) # The transformation matrix
    Minv = cv2.getPerspectiveTransform(dst, src) # Inverse transformation

    return (M, Minv)

(M, Minv) = perspTransform(src, dst)
plt.imshow(img, cmap='gray')
plt.show()
```

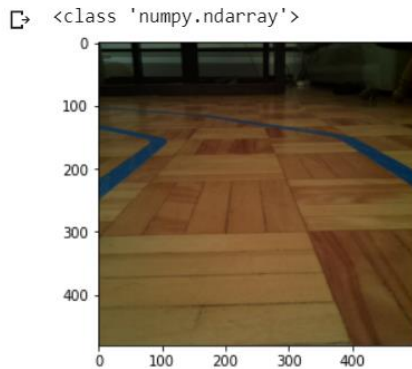


Figure 33: The initial image and code extracting region of interest

Warp Perspective

```
result = cv2.warpPerspective(img, M, (500, 480), flags=cv2.INTER_LINEAR)
rotated = imutils.rotate(result, 270)
edges = cv2.Canny(rotated, 20, 100)
plt.imshow(rotated, cmap='gray')
plt.show()
```

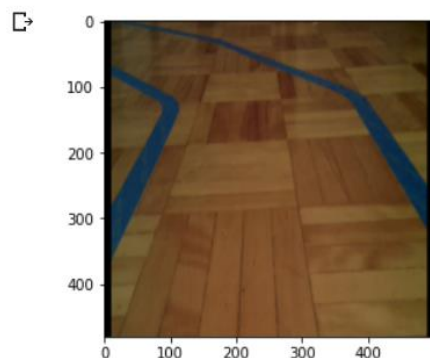


Figure 34: The perspective change in image

6.3 Failed approach in line detection: Canny Edge detection and Hough

Transformation

Initially, the approach was to use canny edge detection and Hough transformation to draw the lines seen in an image. The canny edge detection algorithm provided in the cv2 library has four main stages of detecting edges. First a Sobel kernel is used to perform a spatial gradient measurement on an image emphasizing regions of high spatial frequency that relates to edges [26]. Secondly, a logic of non-maximum suppression is performed that traverses through every pixel in the image, checking if the pixel is the local maximum about pixels. Then, if it is not a local maximum the pixel is set to 0 (suppressed). Finally, Hysteresis Thresholding, which is the parameter used in the function, is used to determine if they are a line or not. As seen in figure 35 below, if a pixel is above the maximum threshold, it is for sure a line edge. Anything below minimum threshold is discarded. In between the maximum and minimum thresholds, any edge detected will be an edge line based on if the edge connects to an edge above the maximum threshold. If it does, the algorithm accepts it as an edge. If not, it discards it and sets the pixels to 0 [27].

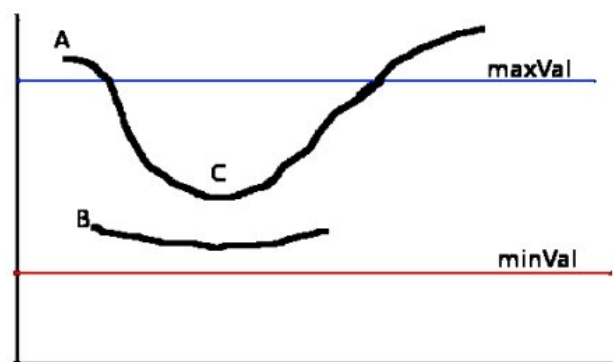


Figure 35: Decision making based on thresholds set by user

After the edges have been detected, lines need to be drawn to calculate the equation of the line to then finally extract a command to send back to the Raspberry Pi. Hough Transformation can be used to detect any shape, given it can be represented as a mathematical expression. Since detecting lines is the main goal here, we apply the Hough Line transform method. The idea is to create a 2D array of the rows denoting the r value and columns the θ (theta) representing the equation of the line. The size of the array can vary to accommodate for precision. This array is populated and returned [28]. The HoughLine function in the cv2 library provides an implementation of this logic. In fact, HoughLinesP, which is the probabilistic Hough line transform is also a function in the cv2 library which is an optimized version of HoughLine [29] With the help of canny edge detection and Hough lines, we can then plot the array of lines to visualize this.

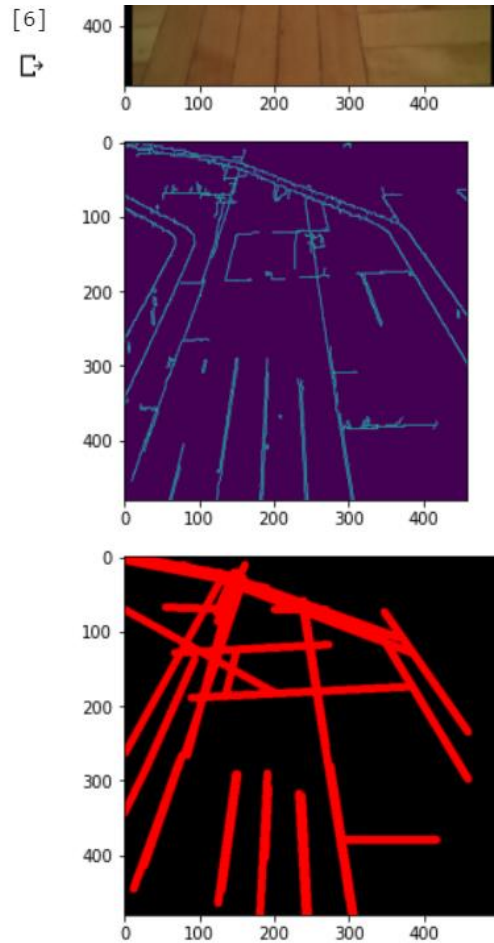


Figure 36: Plotted lines in Hough Line array

As figure 36 above shows this is not the desired output that was expected. The environment the lanes were set up in had lined floors. The canny edge detection algorithm picked these up as edges. Changing the thresholds in the canny edge detection resulted in the lane lines disappearing or picking up more of the lines on the floor. Another concern faced during evaluating this method of detecting lines was that it was specific to a line. Hence, curved roads resulted in an approximation of multiple linear lines. Due to these reasons a different approach was used in detection of lane lines and subsequently extracting directional commands for the modelled car.

6.4 Lane Detection: Successful Model

This section discusses the successful model that was implemented and run for the demo of the project.

6.4.1 Line Detection:

The detection and extraction of lane lines can be broken into two sections. This approach uses the pixel intensity of the image in an algorithm called sliding window algorithm.

6.4.2 Extracting Blue Color from Picture

After a bird's eye view of the image is processed, the image can be treated like a 2D plane with lane lines on it as previously mentioned. Since the lane lines are blue, we can extract only those sets of pixels that represent blue. First, we need to convert the RGB format into HSV since light reflecting off the ground and other nearby objects may affect the extraction processes. After this an upper and lower threshold is set obtaining a range for blue in HSV format. Then using `cv2.inRange` in the `cv2` library we can simply extract pixels of just this range as seen in figure 37 below.

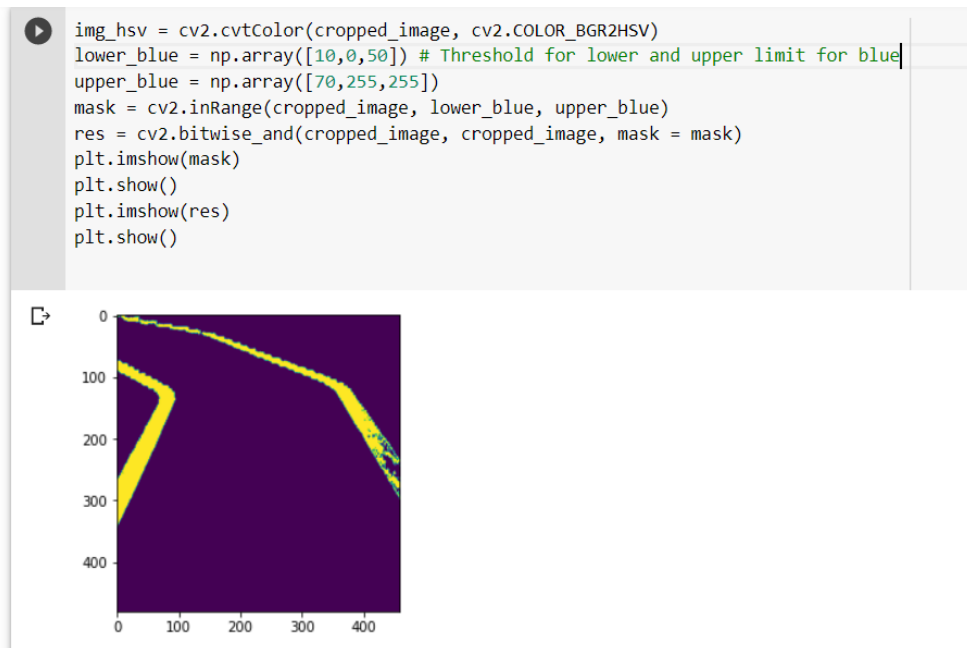


Figure 37: Extracted blue pixels from input image

6.4.3 Extracting Lanes from the Picture

After isolating just the lane lines we need to identify where and how many lines in the image there are, The sliding window algorithm is the key component in the design of this system as it identifies which lane line is seen in the image. The algorithm works in a simple but powerful way. It takes in the isolated lane lines as seen above in figure 37 and plots a pixel intensity histogram. The histogram tells the sliding window algorithm where in the image is the highest pixels located. The histogram tells the sliding window algorithm where in the image is the highest pixels located. This is used to determine the starting point of the algorithm. Figure 38 below visualizes this concept. The image is cropped showing only the bottom half and then a histogram on it is plotted. Figure 38 shows high pixel intensity at the right and left corner on the x-axis of the image.

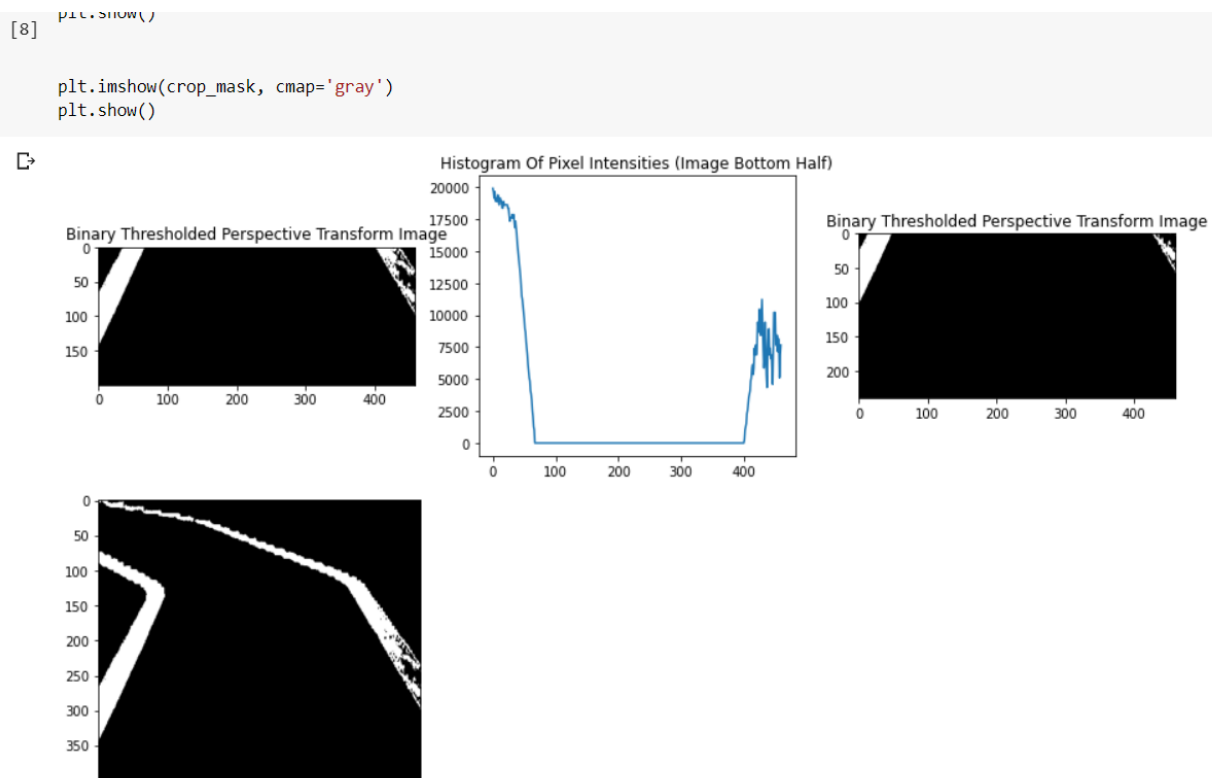


Figure 38: Histogram for determining the starting point on the Sliding Window Algorithm

6.4.4 Sliding Window Algorithm

The sliding window algorithm forms a window that starts at the starting point outlined by the histogram above and then slides in order to capture different portions of the line [30]. This is at the bottom of the image. It repositions itself as the pixels of the image deviates from the center of the window. The below snippet of code in figure 39 shows this logic. It checks if the window indices are within the images. This is then put into a list of indices for both the left and right lane line. Then moves the window according to a minimum threshold. Number of pixels recognized is greater than a threshold of 50 the algorithm recognizes this and re-centers using the mean of all the pixels in the right and left lane line list. This process is repeated until the window hits the top of the image.

```
left_line.windows.append([(win_xleft_low,win_y_low),(win_xleft_high,win_y_high)])
right_line.windows.append([(win_xright_low,win_y_low),(win_xright_high,win_y_high)])

good_left_indices = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
(nonzeroy >= win_y_low) & (nonzeroy < win_y_high)).nonzero()[0]
good_right_indices = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
(nonzeroy >= win_y_low) & (nonzeroy < win_y_high)).nonzero()[0]

# Append these indices to the lists
left_lane_indices.append(good_left_inds)
right_lane_indices.append(good_right_inds)

# If found > minpix pixels, recenter next window on their mean position
if len(good_left_indices) > minpix:
    leftx_current = np.int(np.mean(nonzeroy[good_left_indices]))
if len(good_right_indices) > minpix:
    rightx_current = np.int(np.mean(nonzeroy[good_right_indices]))
```

Figure 39: The core logic in the sliding window algorithm

Finally, a list of indices for the left and right lines is saved and used in the next part of the algorithm. The algorithm simply uses the polyfit function to draw a second-degree polynomial using the indices saved in the previous step. The final product of the processed image looks like the figure 40 below. Finding the average between the left and right lane lines, a center line can be plotted as well.

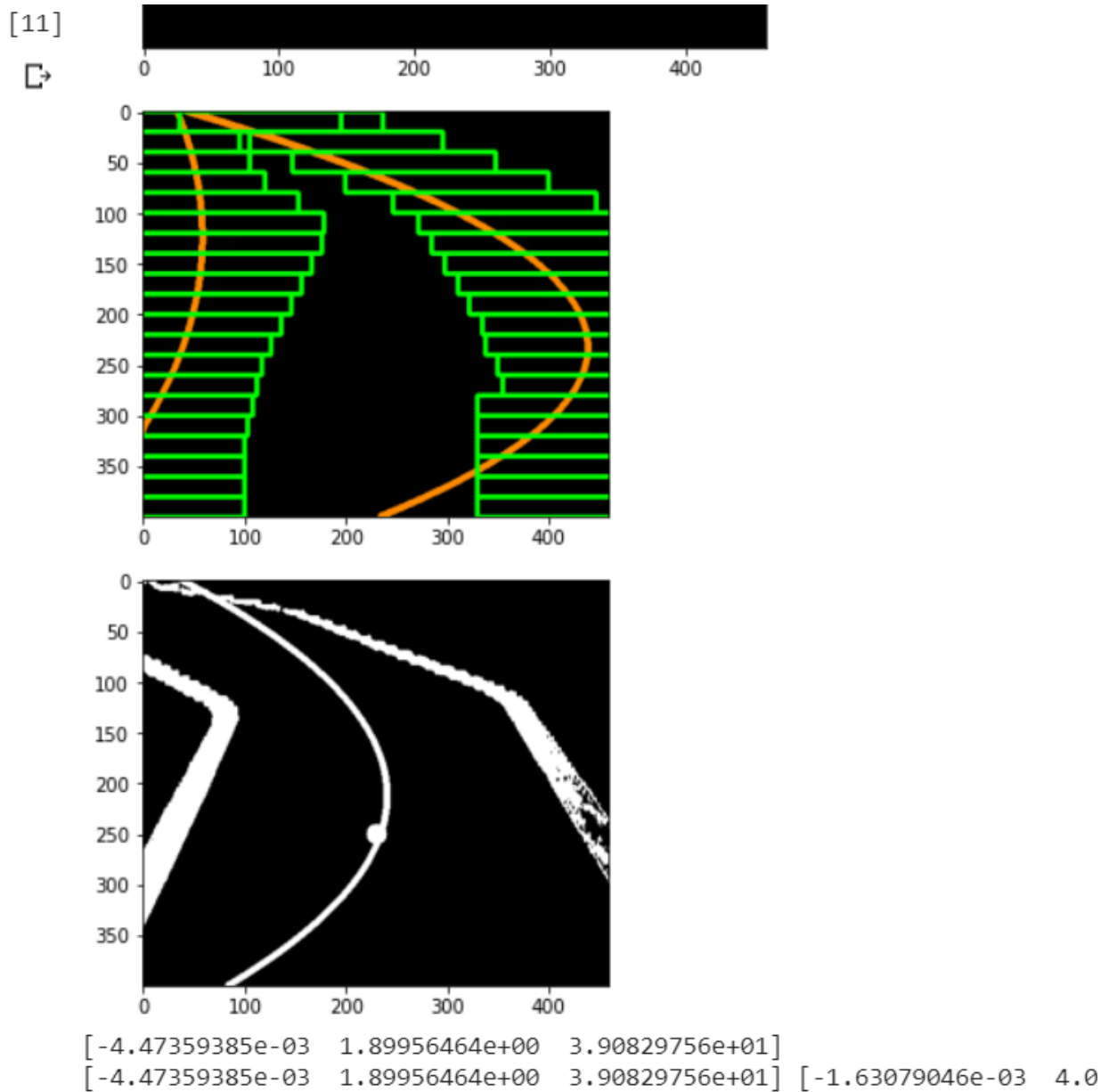


Figure 40: Processing lanes lines to extract directional command to the motors.

As seen previously in the sequence diagram of the system, there are three possible routes for the algorithm to take. One where it detects two distinct lines as seen in the figure above. The second possible prediction is if the picture received has only one visible lane line. If this is the case the left and right windows have the same starting position and they result with the same polynomial. Figure 41 shows the result if only a left lane is detected.

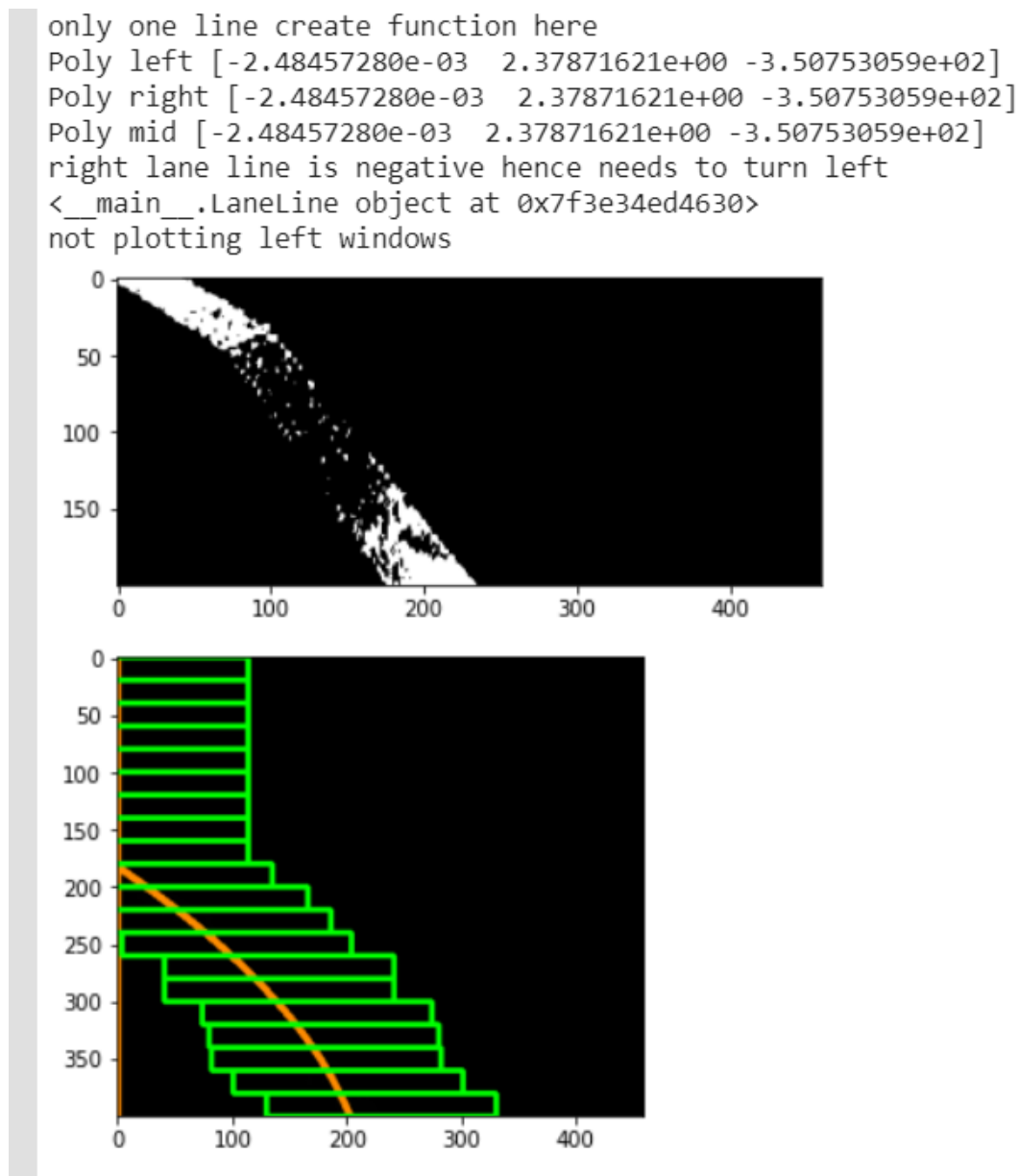


Figure 41: Only one lane line being detected

If only one lane line is detected the algorithm needs to decide if the right lane line has been detected or the left lane line has been detected. In figure 42 Below, bottom left section shows a similar shape to what is observed in the output of the sliding window algorithm in figure 41 above. It is important to note that if we flip the curve seen in figure 41 about the y-axis a left lane would be observed like that of figure 42 on the bottom right box. We can derive two key points from this relationship. Firstly, both left and right lane lines would result in a negative polynomial, i.e. the highest power has a negative coefficient. To distinguish between them a simple function is called upon to check if the function is decreasing or increasing. As seen in figure 42 below taking the first derivative of the function notifies the system if the function is either an increasing or decreasing function. If it is decreasing as seen in figure # above, then it can be confirmed to be the right lane line. Otherwise we can assume the lane is a left lane line. This was done by extracting the subsection of the function plotted on figure 41 above and determined if it had an increasing or decreasing slope for that interval.





	$f'(x) < 0$	$0 < f'(x)$
$0 < f''(x)$	 <p>The function f is decreasing, while the rate itself is increasing. In this case the curve $y = f(x)$ is concave up.</p>	 <p>The function f is increasing, while the rate itself is increasing. In this case the curve $y = f(x)$ is concave up.</p>
$f''(x) < 0$	 <p>The function f is decreasing, while the rate itself is decreasing. In this case the curve $y = f(x)$ is concave down.</p>	 <p>The function f is increasing, while the rate itself is decreasing. In this case the curve $y = f(x)$ is concave down.</p>

Figure 42: Properties of a second-degree concave down polynomial [31]

The algorithm also stores the previous command sent to the Raspberry Pi in order to position itself correctly before it executes the turn. The algorithm records the command sent to the Raspberry Pi. Assume the previous command sent was a forward command, and the next image shows only one lane line. The algorithm uses this information to continue to move forward and set a Boolean called “offset_done” to true. The next image would show a steeper curve like that of figure 41 above. This then checks if the offset_done variable is set to true. If it is then it carries out the left or right command as explained previously. This helps in determining the offset seen in figure 43 below showing an offset of about 20 centimeters.

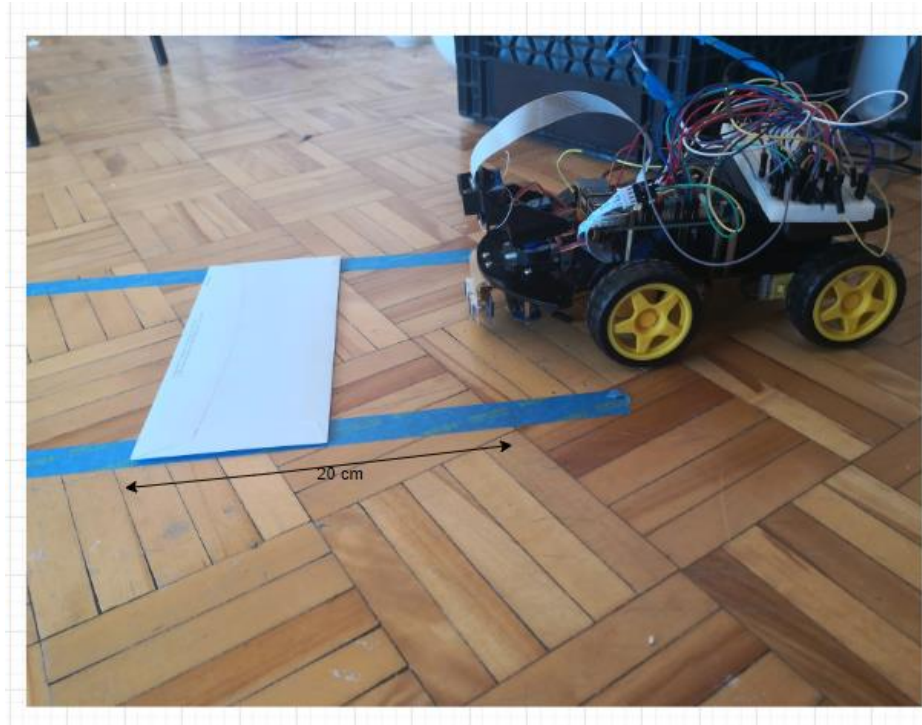
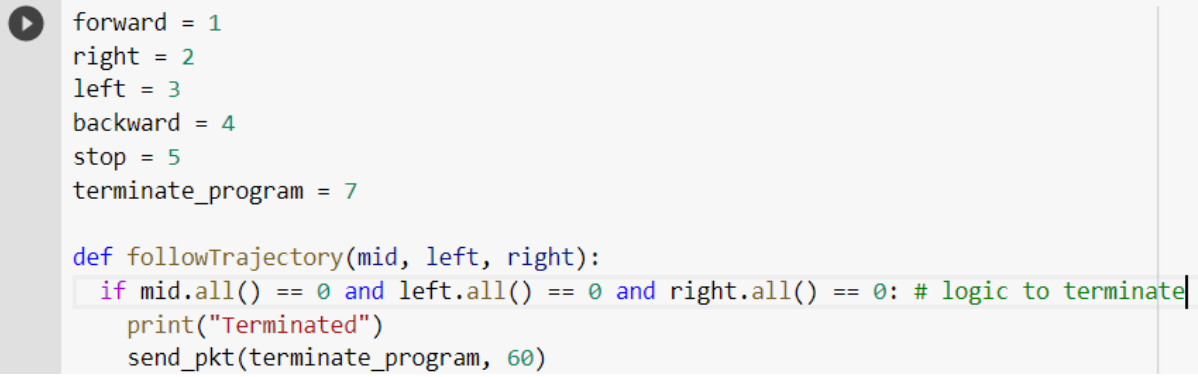


Figure 43: The offset between the car and position where it starts capturing the image

Finally, the last alternative path for the algorithm to take is if there are no lane lines visible in the picture. This would result in the list of coordinates stored to be zeros resulting in no lane lines. For the purpose of this project it was decided that this situation would be the end sequence of the program. As shown in figure 44 below a `terminate_program` command is sent if no lane lines have been detected. An alternative approach would be to start moving the modelled car in a circular manner until it detects lane lines and can continue the sequence.



```
▶ forward = 1
right = 2
left = 3
backward = 4
stop = 5
terminate_program = 7

def followTrajectory(mid, left, right):
    if mid.all() == 0 and left.all() == 0 and right.all() == 0: # logic to terminate
        print("Terminated")
        send_pkt(terminate_program, 60)
```

Figure 44: The termination of program by sending command 7

6.5 Latency of the detection algorithm

It is important to understand the latency of the algorithm as this provides essential feedback into implementing how often we process a frame. Table 4 Below shows test runs on multiple pictures with lanes to estimate the average time taken by the server to obtain a command.

Trial number	Latency (in seconds)
1	0.03789854049682617
2	0.0219419002532959
3	0.019946575164794922
4	0.020944595336914062
5	0.019945621490478516
6	0.019946813583374023
7	0.019933223724365234
8	0.019947528839111328
9	0.02293872833251953
10	0.01994633674621582
	Average = 0.022339

Table 4: The latency of the line detection and command extraction algorithm

The table was plotted on figure 45 below for a better visual perspective on the latency.

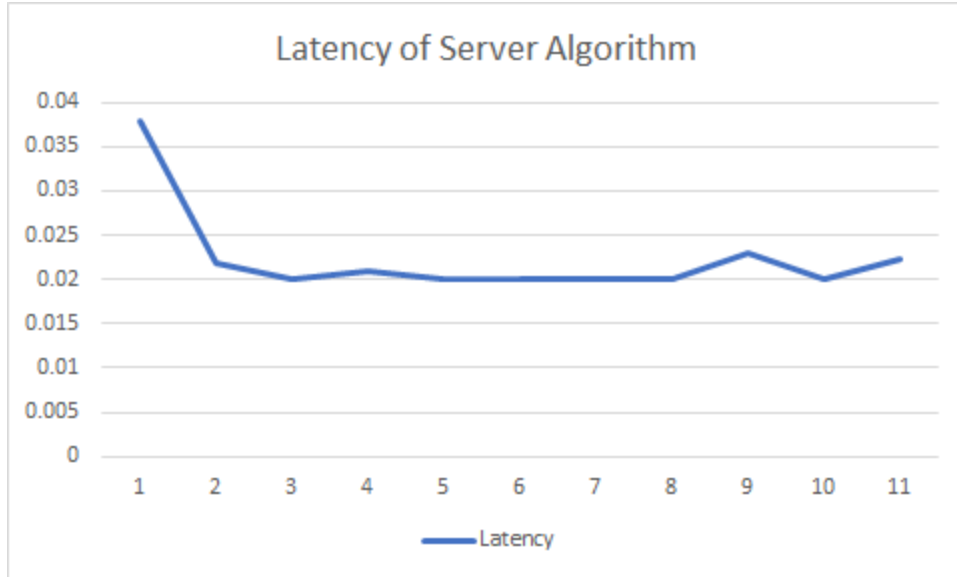


Figure 45: The latency of the line algorithm system

As mentioned in the previous section of this report the raspberry pi needs about 0.25 seconds to send a picture with the desired resolution to the server end of the system. Hence, this was the threshold used to determine the duration of how often we send a frame to the server. We assume the latency for transferring data over the network is negligible and the time taken to execute a command on the Raspberry Pi is instantaneous. All latencies included the system finishes an entire cycle of receiving a picture, processing it, extracting a command, and finally sending the command back to the pi for execution takes about 0.27-0.3 seconds. Hence each command is executed for about 0.3 seconds at a certain PWM outlined in the client-end section.

Demonstration

Due to the cancellation of the Fourth-Year poster fair caused by the COVID-19 pandemic, the demonstration of Smart Collision Avoiding Robot was performed virtually. The following link is a Google Slides demonstration that contains the videos of SCAR operating.

<https://drive.google.com/open?id=1VRhcEXFqSbSOX8QJP-czHu4c wdHIN5BN>

Future Improvements

Without the limiting variable of time and a budget cap, the following would be some challenges to be accomplished for future of this project:

- Addition of a lane changing feature: Using the ultrasonic sensor and the panning function on the camera mount to check the blind spot.
- Create a more robust system that follows trajectories outlined by the sliding window algorithm.
- Create a simulation for speed control using PWM and weight as a constant variable. This will outline simulated speeds to implement into the system for a better flow of movement of the modelled car.
- Implement a Bird's-eye view camera so that location tracking of SCAR can be done.
- Implement a grid network that SCAR can use to get to a specific location
- Integration of additional robots into the system and have them operate together in harmony creating a much more complex system.

Conclusion

With the school academic year coming to an end, our project team was able to successfully finish the project. Our idea was to design a replica of an autonomous car and as the project commenced, the group decided to incorporate the idea of building a robot which could avoid collision, and integrated lane detection in our project. As a result, the project did not continue as initially planned, as many different milestones and unpredicted mishaps emerged. For example, the overheating of our main logic component – the Raspberry Pi 3b+.

To summarize, the project group designed SCAR – a Smart Collision Avoidance Robot, using a Raspberry Pi 3b, an ultrasonic sensor, two H-bridges to control the motors, and a robot chassis, which included motors, batteries, and additional hardware. The robot operates over a client-server code, which gives instructions regarding movement. It uses the sliding window algorithm once the lines on the path have been detected, to guide the car within the lanes. The car is successfully able to stay within the lanes, as well as stop when it detects an object using the ultrasonic sensor. Each frame from the Raspberry Pi's camera is fed to the algorithm to help the car maintain its distance from each lane and continue to follow its path.

Given the current situations due to the pandemic, some events related to the 4th year project did not take place. This included the poster fair and a final demonstration of the project working. This also interfered with the project as the group was unable to collaborate to test various components of the project, as in the final stage, hardware was required to implement the tests and logic. The group would like to give our team member Brian Philip a special shout out for investing a huge amount of time into merging all components and testing the overall working of the project

To conclude, as an aspiring group of students belonging to the Faculty of Engineering and Design, this project helped us research into a diverse range of topics. Our project included researching about the lane-detection technology and real-time communications using networking. Not only did these topics help us build a robot, but it gave us an insight into the world of Computer Vision and Robotics. It was a great opportunity to be able to increase knowledge in a field that is growing and can be applied to several engineering problems in the future.

References

- [1] L. Hattersley, "Raspberry Pi 4 vs Raspberry Pi 3B+ — The MagPi magazine", *The MagPi magazine*, 2019. [Online]. Available: <https://magpi.raspberrypi.org/articles/raspberry-pi-4-vs-raspberry-pi-3b-plus>. [Accessed: 07- Apr- 2020].
- [2] R. Zwetsloot, "Raspberry Pi 4 specs and benchmarks — The MagPi magazine", *The MagPi magazine*, 2019. [Online]. Available: <https://magpi.raspberrypi.org/articles/raspberry-pi-4-specs-benchmarks>. [Accessed: 07- Apr- 2020].
- [3] N. Heath, "Raspberry Pi 4 Model B review: This board really can replace your PC", *TechRepublic*, 2019. [Online]. Available: <https://www.techrepublic.com/article/raspberry-pi-4-model-b-review-this-board-really-can-replace-your-pc/>. [Accessed: 07- Apr- 2020].
- [4] B. Crawford, "How Hot Is Too Hot?", *Raspberry Pi Stack Exchange*, 2019. [Online]. Available: <https://raspberrypi.stackexchange.com/questions/82618/how-hot-is-too-hot>. [Accessed: 07- Apr- 2020].
- [5] J. Hildenbrand, "Raspberry Pi 3 Model B vs. 3 B+: Which should you buy?", *Android Central*, 2019. [Online]. Available: <https://www.androidcentral.com/raspberry-pi-3-model-b-vs-3-b>. [Accessed: 07- Apr- 2020].
- [6] UCTRONICS, *UCTRONICS K0073 Kit*. 2020: <https://www.uctronics.com/media/catalog/product/cache/1/image/1500x1500/9df78eab33525d08d6e5fb8d27136e95/7/3/73.jpg> [Accessed: 07-Apr-2020]

- [7] UCTRONICS, *UCTRONICS K0073 Chassis*. 2020:
<https://www.uctronics.com/wiki/images/thumb/6/6b/Header1.1.2.png/1000px-Header1.1.2.png> [Accessed: 07-Apr-2020]
- [8] SolarRobotcs, *Raspberry Pi Camera V2*. 2020: https://cdn.solarbotics.com/wp-content/uploads/52090-img_0934-1.jpg [Accessed: 07-Apr-2020]
- [9] "L293D Diode", *Mouser.ca*, 2003. [Online]. Available:
<https://www.mouser.ca/datasheet/2/389/cd00000059-1795435.pdf>. [Accessed: 07- Apr-2020].
- [10] A. Industries, "DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC", *Adafruit.com*. [Online]. Available: <https://www.adafruit.com/product/3777>. [Accessed: 07- Apr- 2020].
- [11] Texas Instruments, *L293D*. 2020:
<https://www.robotshop.com/media/catalog/product/cache/image/400x400/9df78eab33525d08d6e5fb8d27136e95/6/0/600-ma-dual-h-bridge-motor-driver-dc-steppers-l293d.jpg>
[Accessed: 07-Apr-2020]
- [12] "L293x Quadruple Half-H Drivers", 2016. [Online]. Available:
<http://www.ti.com/lit/ds/symlink/l293.pdf>. [Accessed: 07- Apr- 2020].
- [13] "Ultrasonic Ranging Module HC - SR04", *Cdn.sparkfun.com*. [Online]. Available:
<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>. [Accessed: 07-Apr- 2020].
- [14] SAMSUNG, *16850 Batteries*. 2020: https://cdn11.bigcommerce.com/s-272hagu0yx/images/stencil/original/products/116/433/20S_1__45457.1573630720.jpg?c=2&imbypass=on&imbypass=on [Accessed: 07-Apr-2020].

- [15] Walmart, *Black Web Power Bank*, 2020: https://i5.walmartimages.com/asr/675d197b-b540-4196-9d0f-01149a5d3374_1.dab075626a01df18d15c3a35fdffeda7.jpeg?odnWidth=undefined&odnHeight=undefined&odnBg=ffffff [Accessed: 07-Apr-2020].
- [16] "D-Link Technical Support", *Support.dlink.ca*, 2020. [Online]. Available: <https://support.dlink.ca/ProductInfo.aspx?m=DIR-880L>. [Accessed: 07- Apr- 2020].
- [17] "raspi-config - Raspberry Pi Documentation", *Raspberrypi.org*, 2020. [Online]. Available: <https://www.raspberrypi.org/documentation/configuration/raspi-config.md>. [Accessed: 07- Apr- 2020].
- [18] "L293D Dual H-Bridge Motor Driver - ProtoSupplies", *ProtoSupplies*, 2020. [Online]. Available: <https://protosupplies.com/product/dual-h-bridge-motor-driver-l293d/>. [Accessed: 07- Apr- 2020].
- [19] G. McComb, "Ways to Move Your Robot," *Servo Magazine*. [Online]. Available: https://www.servomagazine.com/magazine/article/may2014_McComb. [Accessed: 07-Apr-2020].
- [20] JordanDee, *Pulse Width Modulation*. [Online]. Available: <https://learn.sparkfun.com/tutorials/pulse-width-modulation/all>. [Accessed: 07-Apr-2020].
- [21] P. Hut, "HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi," *The Pi Hut*, 22-Oct-2013. [Online]. Available: <https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi>. [Accessed: 07-Apr-2020].

- [22] "Camera Calibrator," *What Is Camera Calibration? - MATLAB & Simulink*. [Online]. Available: <https://www.mathworks.com/help/vision/ug/camera-calibration.html>. [Accessed: 07-Apr-2020].
- [23] 2020. [Online]. Available: https://www.researchgate.net/publication/302555906_Development_of_the_local_magnification_method_for_quantitative_evaluation_of_endoscope_geometric_distortion. [Accessed: 07- Apr- 2020].
- [24] Kemfic, "Curved Lane Detection", *Hackster.io*, 2020. [Online]. Available: <https://www.hackster.io/kemfic/curved-lane-detection-34f771>. [Accessed: 07- Apr- 2020].
- [25] "Operations on Arrays — OpenCV 2.4.13.7 documentation", *Docs.opencv.org*, 2020. [Online]. Available: [https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#void%20perspectiveTransform\(InputArray%20src,%20OutputArray%20dst,%20InputArray%20m\)](https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#void%20perspectiveTransform(InputArray%20src,%20OutputArray%20dst,%20InputArray%20m)). [Accessed: 07- Apr- 2020].
- [26] "Feature Detectors - Sobel Edge Detector", *Homepages.inf.ed.ac.uk*, 2003. [Online]. Available: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>. [Accessed: 07- Apr- 2020].
- [27] "Canny Edge Detection — OpenCV-Python Tutorials 1 documentation", *Opencv-python-tutroals.readthedocs.io*, 2020. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html. [Accessed: 07- Apr- 2020].

- [28] "Line detection in python with OpenCV | Houghline method - GeeksforGeeks", *GeeksforGeeks*, 2020. [Online]. Available: <https://www.geeksforgeeks.org/line-detection-python-opencv-houghline-method/>. [Accessed: 07- Apr- 2020].
- [29] "Hough Line Transform — OpenCV 2.4.13.7 documentation", *Docs.opencv.org*, 2020. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html. [Accessed: 07- Apr- 2020].
- [30] J. Moore, "An Introduction to Sliding Window Algorithms", *gitconnected.com*, 2019. [Online]. Available: <https://levelup.gitconnected.com/an-introduction-to-sliding-window-algorithms-5533c4fe1cc7>. [Accessed: 07- Apr- 2020].
- [31] "Concavity", *Ximera.osu.edu*, 2020. [Online]. Available: <https://ximera.osu.edu/mooculus/calculus1/master/higherOrderDerivativesAndGraphs/digInConcavity>. [Accessed: 07- Apr- 2020].
- [32] "GPIO," GPIO - Raspberry Pi Documentation. [Online]. Available: <https://www.raspberrypi.org/documentation/usage/gpio/>. [Accessed: 07-Apr-2020].

Appendices

Appendix A

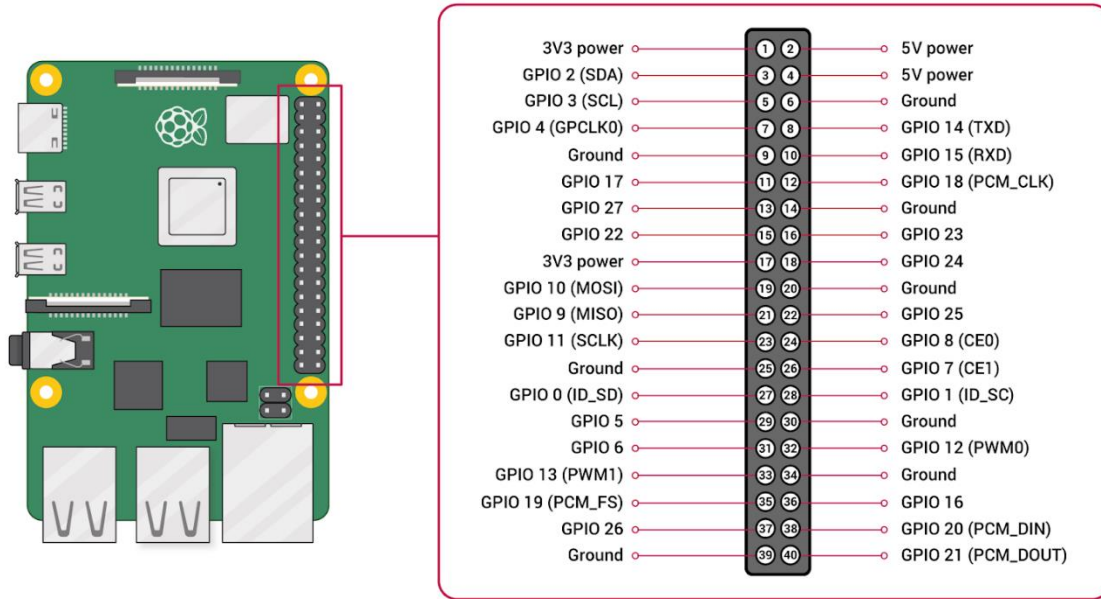
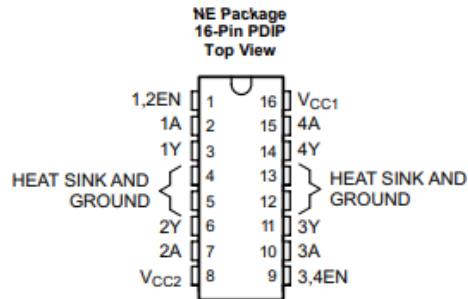


Figure 46: A detailed pinout configuration of the Raspberry Pi 3b [32]

Appendix B

5 Pin Configuration and Functions



Pin Functions

PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to printed-circuit-board ground plane with multiple solid vias
V _{CC1}	16	—	5-V supply for internal logic translation
V _{CC2}	8	—	Power VCC for drivers 4.5 V to 36 V

Figure 47: Detailed Pin Configuration of the L293D H-Bridge [12].

More details of the L293D H-Bridge are provided in this datasheet:

<http://www.ti.com/lit/ds/symlink/l293.pdf>