



University of Wales Trinity Saint David
School of Engineering

Major Project Report

Michael Anderson
P151178

21/02/2020

Major Project Report

M6X00417

Kerry Tudor

Dedicated to my Dad

He was so proud that I was accepted into university, but was taken before I could attend.

Be safe on the road.

Declaration of Authenticity

I, Michael Anderson, declare this dissertation is the result of my original research and development. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Acknowledgements

I would like to thank Kerry Tudor for her continued support and guidance throughout this project, pushing me to gain the best performance and validation methods for the development of this automated system. Her skills for report writing and structure has helped me greatly.

Leo May was courteous enough to provide the wide-angle fisheye lens that was used for this project, free of charge. Without him, I would have had to spend another £10 on an already expensive project.

Danny Richardson, for his fountain of logic which helped shape decisions taken for progression.

Mark Fall and Paul Davis have provided both their advice and immense support throughout this project and my career at university.

And finally, my mom, Gayle and my cousin Monique for enduring the proofreading process of this report.

Abstract

The safety of the worlds' roads are paramount to both the governing bodies of every country and vehicle manufacturers. Automated Driving Assistance Systems (ADAS) have been developed by numerous companies to improve the safety of vehicles around the world. By removing the individual from behind the wheel and replacing it with an Artificial Intelligence (AI), you remove the human error that occurs in the process which is responsible for the vast majority of road traffic collisions.

This report details the steps taken to achieve a working ADAS with use of a 3D printed ego vehicle, Raspberry Pi single board computer to control the on-board motors and MATLABs Automated Driving toolbox to perceive the ego vehicles surrounding environment. The project undertook many iterations of development to ascertain the best performance, using the simplest method possible to create a system that can be mass produced and implemented as a cheap but effective solution to a problem that is only getting worse with the increasing number of incompetent drivers.

There are a lot of videos and photos of this project in development. If you would like to view this material, please access the following OneDrive folder:

[OneDrive Link](#)

Contents

1) Introduction	1
1.1. Aims of the Project.....	2
2) Background Research.....	3
2.1. Current Applications of Autonomy in Road Vehicles	3
2.2. MATLAB's Automated Driving Toolbox	7
2.3. Raspberry PI.....	9
2.4. Potential Sensors.....	10
3) Methodology	12
3.1. Virtual Code Validation	12
3.2. Physical Testing Procedures.....	15
3.2.1. Solid Road Lane Markers	15
3.2.2. Dashed Road Lane Markers	24
3.2.3. Different Mounts And Positions For The Pi Camera	27
3.2.4. Varying the Lens Used On The Pi Camera	30
3.2.5. Development Of Stopping Procedures	32
4) Discussion	44
5) Conclusion	49
6) Further Work.....	50
7) References	51
8) Appendix.....	64
8.1. Appendix A – Code & Vehicle Construction	64
8.1.1. Ego Vehicle Construction.....	64
8.1.2. Code Construction	88
8.2. Appendix B – Object Detection	100
8.3. Appendix C – Parts List	111
8.4. Appendix D – Samsung INR18650-30Q Lithium-ion Cell Specifications.....	113
8.5. Appendix E – MATLAB Script To Operate The Ego Vehicle.....	115

Table of Figures

Figure 1 – Sensor Coverage And Implementation; Tesla (Tesla Motors, 2019) (Left) & Waymo (Waymo LLC, 2019) (Right)	4
Figure 2 – Bosch Automated Valet Parking (Robert Bosch GmbH, 2019) (Left) & Audi AICon Concept (Audi, 2017) (Right).....	4
Figure 3 – Fiat Chrysler Jeep Cherokee (Lazarra, 2015) (Left), Charlie Miller & Chris Valasek Responsible For The Hacking (Wired, 2015) (Right)	5
Figure 4 – Stop Signs That Have Confused An Autonomous Vehicles Neural Network (Highfield, 2017).....	6
Figure 5 – MATLAB Automated Driving Functionality; Detecting Lane Markers (Left) & Compiling Sensor Detections Data Onto A Bird's Eye Plot (Right) (Mathworks, 2017)	7
Figure 6 – Applications Of MATLABs Automated Driving Toolbox; Caterpillars Deep-Learning Infrastructure (Mianzo, 2017) (Top) & Scania's Automated Emergency Braking System (Andersson, 2016) (Bottom)	8
Figure 7 – R/C 3D Printed Vehicle Used As A Base Vehicle For This Project (Banana_Science, 2018) (Left) & Raspberry Pi And Arduino Laptop (Electronics For Everyone, 2017) (Right) ..	9
Figure 8 – Operation Of An Infrared Sensor (Burnett, 2017) (Left), & Of A Lidar Sensor (Schweber, 2019) (Right)	10
Figure 9 – Operation Of A Radar Sensor (Masinde Muliro University of Science and Technology, 2019) (Left), & Of An Ultrasonic Sensor (Seeed Studio, 2020) (Right)	11
Figure 10 – Screenshot Of Every Function Within A Separate Figure During The Validation Process.....	12
Figure 11 – Varying The Level Of Sensitivity Within The "BW" Image; Too Low (Left), Optimal (Middle) & Too High (Right).....	13
Figure 12 – Calculating Steering Angle Command Path Using The Stanley Method	14
Figure 13 – Physical Representations Of A Straight Road (Left) And Curved Road (Right). ..	15
Figure 14 – Measured Angles Of Each Section Of The Curved Road	16
Figure 15 – Wide Lane Road (Left) & Narrow Road (Right)	16
Figure 16 – First Attempt At Physical Testing Using Vertical Camera Mount.....	16
Figure 17 - Changes To The Sensitivity Factor; Too Low (Left), Optimal (Middle) & Too High (Right)	17
Figure 18 - Effect Of Changing The Minimum ROI Setting On The Bird's Eye View; 0m (Left) & 0.25m (Right).....	18
Figure 19 – Effect Of Changing The Maximum ROI Setting On The Bird's Eye View; 1m (Left), 0.8m (Middle) & 0.6m (Right)	18

Figure 20 – Effect Of Changing The Maximum ROI Setting When Navigating A Corner; ‘0.6m’ Causes Issues (Top) & ‘0.4m’ Proving More Efficient (Bottom)	19
Figure 21 – Region Of Interest For Use With Converting An Image To A Birds' Eye View (Mathworks, 2019)	20
Figure 22 – Code to Increase/Decrease the Maximum ROI X-axis Value.....	21
Figure 23 – Effect Of The Automatically Adjusting The Maximum ROI X-axis Distance	21
Figure 24 – Code to Increase/Decrease the Minimum ROI X-axis Value.....	22
Figure 25 – Effect Of The Automatically Adjusting The Minimum ROI X-axis Distance	22
Figure 26 – Code To Reset The Parabolic Lane Boundary Lengths.....	23
Figure 27 – Effect Of Changing The Lane Marker Size Within The Code, While Keeping The Same Sensitivity Value Of 0.35; 0.01m (Left), 0.013m (Middle) & 0.016m (Right).....	23
Figure 28 – Longitudinal Lane Markers Dimensions As Given By Department Of Transport; 40mph Or Less (Left) & Greater Than 40mph (Right) (Department of Transport, 2018)	24
Figure 29 – Scaled Down Representation Of A Dashed Line UK Road	25
Figure 30 – Inaccuracy Of Parabolic Lane Boundaries Without Smaller Lines	26
Figure 31 – Optimised Performance And Stability On The Dashed Lines.....	26
Figure 32 – Physical Testing Using A Camera Mount With 0° To The Vertical; Static Position (Left) & Raised Rear (Right).....	27
Figure 33 – Displaying The Process Of Creating Multiple Reference Planes To Gain The Desired Angle For The Camera Mount And The Numerical Order In Which They Were Created	28
Figure 34 – Various Iterations Of Camera Mount Designs; Initial Vertical Mount 20mm Shank (Left), 11.25° Pitch 20mm Shank (Middle Left), 11,25° Pitch 40mm Shank (Middle Right), 22.5° Pitch 40mm shank (Right).....	28
Figure 35 – Effect From Changing The Pitch Of The Camera Mounting; 11.25° (Left) & 22.5° (Right)	29
Figure 36 – Comparison Of Standard Lens (Left) Against Fisheye Lens (Right) At The Same Point Along A Corner	31
Figure 37 – Vehicle Stopping Test; Measurement Using A Tape Measure (Left), Onboard View Of The Vehicle Annotated With Distance Calculated By The Infrared Sensor (Right).....	32
Figure 38 – Closest Approach To Empty Box When Attempting To Stop; Tape Measure (Left), Onboard Image (Right)	33
Figure 39 – Collision With Empty Box When Stopping; Tape Measure (Left), Onboard Image (Right).....	34
Figure 40 – ‘If’ Statement To Perform The ‘Stop’ Procedure	34
Figure 41 – “Two Second Rule” Demonstration Which Would Be Applied To The Ego Vehicle (Driving Test Tips, 2020).....	35

Figure 42 – Lag Experienced While Operating the System; First Frame Sensor Detected Object As A Threat (Left), First Frame Camera ‘Caught Up’ With The Object (Right).....	36
Figure 43 – Lag Experienced While Operating the System; First Frame Sensor Stopped Detecting Object As A Threat (Left), First Frame Camera ‘Caught Up’ With The Object (Right)	36
Figure 44 – Small Code To Determine An Adequate Acceleration/Deceleration Rate	37
Figure 45 – Code To Determine The Value For "Drive" To Operate The Electric Motor	38
Figure 46 – The Fluctuation Of FPS On The Same Hardware During Operation, High (Left), Low (Right)	38
Figure 47 – Emergency Stop Procedure With The Code	39
Figure 48 – Code To Accelerate The Ego Vehicle Should The System Deem It Safe To Proceed	40
Figure 49 – Tennis Ball Rolling In Front Of The Ego Vehicle, Detected By Sensor And Determined As A Danger (Left), Still In Shot Of Camera And Potential Danger (Right)	40
Figure 50 – Code To Determine Whether There Is Potential Of Further Danger	41
Figure 51 – Distances Annotated Onto The Pi Camera Live Feed Before Setting Specific Values For Each Protocol	43
Figure 52 – 3D Virtual Simulation To Assess The Performance Of An Automated System (Lozé, 2019).....	44
Figure 53 – Sensor Coverage On A Ego Vehicle; Windscreen Mounted Camera Blind Spots (Felix, 2019) (Left) & Shadows Can Be Seen Within A LiDar Sensor Sweep (Scott Drew Pendleton, 2017) (Right)	45
Figure 54 – Visual Representation Of Tesla Vehicles Sensor Range (Tesla Motors, 2019)	46
Figure 55 – Cyber security On Automated Vehicles; Vulnerable Systems (Bedigian, 2018) (Left), Threat Security (Continental, 2020) (Right)	47
Figure 56 – Fatal Crash In Arizona; Dashcam Last Moment Before Impact (Levin, 2018) (Left), News Coverage (Gartenberg, 2018) (Right).....	48
Figure 57 – Imported '.stl' File As A Graphics Object.....	64
Figure 58 – Using The Graphics Object As A Reference Point To Recreate The Model.....	65
Figure 59 – Completed CAD Model Of The Ego Vehicle (Left), ...	65
Figure 60 – Ascertaining Measurements For The Raspberry Pi Board	66
Figure 61 – Displaying The Camera Mount At The Front Of The Vehicle	66
Figure 62 – Battery Case Assembly; Sides Separate (Left) & Sides Inserted (Right)	67
Figure 63 – Different Iterations Of The Rear Bumper; Original (Left) & Modified (Right).....	67
Figure 64 - Fisheye Lens; Attached (Top Left), Motor's R/C Battery Pack (Top Right), Camera Lens Vibration (Bottom)	68

Figure 65 – Cable Ties Use To Secure The Battery In Place, Attached (Left), Removed (Middle), Battery Wire (Right).....	68
Figure 66 – Mount Created To Securely Hold The Fisheye Lens, CAD model (Left), Assembled (Right).....	69
Figure 67 – View From The Fisheye Lens With New TPU Mount (Left), Updated CAD Model Of The Ego Vehicle After All Modifications (Right)	69
Figure 68 – Weights Used To Apply Downforce To The front Wheels Of The Ego Vehicle, In place (Left), Attached Using Insulation Tape (Right)	70
Figure 69 – Connections To Servo Motors; Steering Servo (Left) & Driving Motor (Right)...	71
Figure 70 – Servo Motors Connected To The Raspberry Pi's GPIO Pins; Steering Motor (Left) & Driving Motor (Right).....	71
Figure 71 – Both Motors Connected To The Raspberry Pi GPIO Pins	72
Figure 72 – MATLAB Code To Establish Connection To The Raspberry Pi Board, Camera, Sensor & Motors	72
Figure 73 – Soldering Of The Leaf Springs (Left) & Soldering To The Power Bank Board (Right).....	73
Figure 74 – Soldering Of The Battery Pack; Positive (Left) & Negative (Right).....	74
Figure 75 – Battery Pack Connected To A Variable Load; 1A (Left) & 2A (Right).....	74
Figure 76 –The Battery Pack Connected To The Raspberry Pi Displaying The Current Draw; 1 Minute After Connection (Left) & 5 Minutes After Connection (Right)	75
Figure 77 – Wiring Layout For Connecting The MCP3008 ADC To the Raspberry Pi (Mathworks, 2020)	76
Figure 78 – Read Data From Infrared Sensor and Convert That Value Into A Voltage, Then Into A Distance.....	78
Figure 79 – Graph Depicting The Voltage Output Based On The Relevant Object Distance (Sharp, 2006).....	79
Figure 80 – Attempts At Gaining A Polynomial Equation To Calculate The Distance Using The Infrared Sensor; 9 th Degree Polynomial (Left), 8 th Degree Polynomial (Right)	79
Figure 81 – Testing The Accuracy Of The Infrared Sensor; Object At 400mm Away (Left), Closest Object Could Get To The Sensor (110mm) Before Distance Increased Again (Right)	80
Figure 82 – Wiring The Ultrasonic Sensor Into The Raspberry Pi (Raspberry Pi tutorials, 2019) (Left), Both Ultrasonic And Infrared Sensor Connected To The Pi Using A Breadboard (Right)	81
Figure 83 – Python Script To Operate The Ultrasonic Sensor Through The Raspberry Pi (Raspberry Pi tutorials, 2019).....	83

Figure 84 – Infrared Sensor Mount; CAD Model (Left), Front View (Middle), Side View (Right)	84
Figure 85 – Top View Once Assembled (Middle) Side View (Right)	84
Figure 86 –Tidy For The Sensor Wires (Left), Heat Shrink To Prevent Perceptive System Being Misled (Right)	85
Figure 87 – Shortening All Of The Wires For A More Aesthetically Pleasing Feel; Original Setup (Left), New Placement (Middle), Wires Shortened (Right)	85
Figure 88 – Infrared Sensor Mount; Overhanging The Bumper (Left), New Design (Right)	86
Figure 89 – Aftermath Of Devastating Autonomous Vehicle Collision With Fire Door That Appeared Out Of Nowhere	86
Figure 90 – New Bumper vs. Original (Left), New Bumper Design With Infrared sensor Mount Repositioned	86
Figure 91 – Effect Of Connecting The Infrared Sensor Mount To The Steering Rack; Static (Top), Steering Left (Bottom Left), Steering Right (Bottom Right)	87
Figure 92 – MATLAB Code To Create 'monoCamera' Object	88
Figure 93 – MATLAB Code To Initiate Deployable Video Player	89
Figure 94 – MATLAB Code To Begin The 'while' Loop And Take A Snapshot From The Pi Camera	89
Figure 95 – MATLAB Code To Define ROI & Image Size For Bird's Eye View	89
Figure 96 – MATLAB Code To Create A Bird's Eye View And Transformed Image	90
Figure 97 – Original Camera Image (Left) & Transformed Bird's Eye View (Right)	90
Figure 98 – Bird's Eye Image Converted To Greyscale	90
Figure 99 – Bird's Eye View Transformed Into A Binary Black & White Image	91
Figure 100 – Black & White Bird's Eye View Displaying 'Noise'	91
Figure 101 – MATLAB Code For Retrieving Lane Marker Points And The Conversion To Vehicle Coordinates And Parabolic Lane Boundaries	92
Figure 102 – MATLAB Code To Computation Of Boundary Model	92
Figure 103 – Defining Left & Right Lanes	93
Figure 104 – 'yOut' Variable Plotted To Determine Accuracy	93
Figure 105 – Left & Right Lanes Annotated Onto The Original Bird's Eye View	94
Figure 106 – Parabolic Lanes Annotated Onto The Original Cameras' Image	94
Figure 107 – MATLAB Code To Associate Lanes For Steering Control	95
Figure 108 – Plot Displaying Lane Boundaries & Ego Vehicle Path	95
Figure 109 – Redefining The Value Of Steer From The Previous Iteration	96
Figure 110 – Comparison Showing the Effect Of Taking Different "yCentre" Values; Closer To The Ego Vehicle (Top), Then Further Away From The Ego Vehicle (Right)	97

Figure 111 – MATLAB Code For Steering Control Operation to Calculation & Steering Value Log.....	98
Figure 112 – MATLAB Code For Steering Control Operation to Neutralise Adverse Steering Values.....	98
Figure 113 – Comparison of Accuracy (Mean Average Precision (mAP)) vs Training Time In milliseconds (GPU Time) (Hui, 2018)	101
Figure 114 – Applying Ground Truth Using The Image Labeller Application Within MATLAB	101
Figure 115 – Basic R-CNN Training On A CPU With 16 Training Images	102
Figure 116 – Images Used Within The R-CNN Detector Annotated With The Predicted Labels	102
Figure 117 – Images The R-CNN Had Been Trained On	103
Figure 118 – Images The R-CNN Had Never Seen Before	103
Figure 119 – Images The R-CNN Has Seen (Top), Images the R-CNN Has Never Seen Before (Bottom)	104
Figure 120 – Comparison Of Computation Time Between Original Size And 227 x 227 Resized Images.....	104
Figure 121 – Comparison Of Computation Time Between Original Size and 480 x 640 Resized Images.....	105
Figure 122 – Detections Of Images The Network Has Never Seen Before Using 480 x 640 Image Size.....	105
Figure 123 – Detections Of Images The Network Has Never Seen Before Using 227 x 227 Image Size.....	106
Figure 124 – Computation Time Of 227 x 227 Resized Images Ran Through The R-CNN Repeatedly.....	106
Figure 125 – Initial Images Captured With The Raspberry Pi Camera And Processed Through the Detector	107
Figure 126 – Images Captured With the Raspberry Pi Camera And Process Through The R-CNN.....	107
Figure 127 – Live Feed Of Images From The Raspberry Pi Camera Passed Through To The Detector	108
Figure 128 – Detections Of Images By The Faster R-CNN Detector From Online Sources And Pi Camera With Their Computation Time	109
Figure 129 – Comparison Of Images Processed Through Both The Basic R-CNN And The Faster R-CNN Detectors	110
Figure 130 – Samsung INR18650-30Q Lithium-ion Cell Specifications	113
Figure 131 – Samsung INR18650-30Q Lithium-ion Cell Specifications	114

1) Introduction

In today's society, the safety of drivers and passengers within all vehicles on the road has been made one of the main concerns for all manufacturers, alongside emissions and fuel economy.

In recent decades, numerous safety features have drastically enhanced and become standard even on the most basic vehicles, meaning that the main contributor to road accidents and collisions are down to human error. 1.35 million people are killed each year in vehicles on the road worldwide and 94% of crashes involve human error in the US. (World Health Organization, 2018)

Now, the only safety feature that can be implemented onto a vehicle, that would drastically reduce the likelihood of otherwise easily avoidable collisions, is automated driving. By removing the human aspect of the vehicle and letting a computer process the information at an accelerated rate, the roads around the world would become a much safer place.

This goal has been at the forefront of the automotive industry for at least 10 years, by a diverse range of companies, however it still hasn't quite gained the widespread momentum to become included on every vehicle, as perhaps anticipated. This could be due to many factors; the cost of implementing the technology and developing the system, people's opinions and lack of trust with an Artificial Intelligence (AI) managing the vehicle, and the general public consensus of not feeling comfortable with not being in direct control.

Although the concept is there, the programming, practicality and infrastructure of a driver-less system hasn't been fully established to incorporate every scenario that a typical driver would encounter on a day to day basis, so the most standardised level of autonomy in current vehicles is level 2, where Tesla and Waymo are level 3. It is hard to programme and incorporate a system to deal with every situation, as a slight change in the vehicles surroundings could drastically change the appropriate action the vehicle would need to undertake to preserve the safety of the ego vehicles' occupants, surrounding vehicles and pedestrians.

Therefore, the purpose of this project is to develop and test the simplest method of programming a MATLAB code to accurately control the hardware of a scaled down vehicle, that if successful, could be adapted for use within a life-sized vehicle.

1.1. Aims of the Project

- Research autonomous control of vehicles in development and currently used.
- Develop CAD model of ego vehicle.
- Assemble 3D printed components for ego vehicle.
- Connect all components to the Raspberry Pi.
- Write a MATLAB script to control the electronic functions of ego vehicle.
- Test the control of ego vehicle.
- Implement sensors to calculate the distance in front of the ego vehicle.
- Enhance the MATLAB script for stop/avoidance procedures.
- Create test track for validation of ego vehicle control.
- Adapt object detection for use with the ego vehicle.

2) Background Research

2.1. Current Applications of Autonomy in Road Vehicles

Many systems have already been implemented into vehicles. There are currently 6 levels of automation (Kelechava, 2018), based upon how many features of the Dynamic Driving Task (DDT) the on-board computer is in control of (Wendling, 2017).

- Level 0 – No Automation
No computer aided systems present that interfere with human driver's control of vehicle – ABS.
- Level 1 – Driver Assistance
Assistance to driver from on-board computer in either steer OR acceleration/ deceleration – Lane Assist or Cruise Control. (Snider, 2009)
- Level 2 – Partial Assistance
Multiple systems for driver assistance for BOTH Steering and acceleration/ deceleration – Lane Assist and Adaptive Cruise Control. (TAKAHAMA & AKASAKA, 2017)
- Level 3 – Conditional Automation
Computer controlled assistance systems for ALL aspects of the DDT, however, the human driver is required to respond to intervention requests – Lorry Platooning (European Automotive Manufacturers Association, 2017), Tesla Model S 'Autopilot', Audi A8L (Wasef, 2018), Rail Bus (Hitti, 2017)
- Level 4 – High Driving Automation
Automated control of ALL aspects of the DDT, even if the human driver does not respond to the computers request to intervene – Waymo project, Bosch Automated Valet Parking (AVP).

- Level 5 – Full Driving Automation

Full-time execution of ALL aspects of DDT controlled by the on-board computer under all road conditions and situations – Audi AICon concept (Audi, 2017), Driverless Taxis (Allen, 2019).

Most systems implemented into automated vehicles manufactured today are Level 2, aimed at reducing the likelihood of the ego vehicle accidentally changing lanes due to a distracted or tired driver, as well as controlling the speed of the ego vehicle to improve fuel economy (Kaslikowski, 2019). This usually incorporates the use of a neural network to learn and enhance the performance of the system. (Hadik, et al., n.d.)

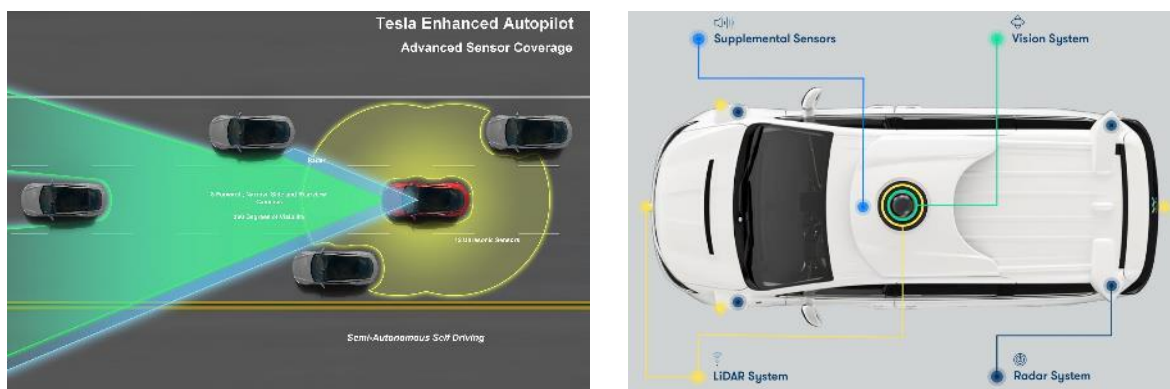


Figure 1 – Sensor Coverage And Implementation; Tesla (Tesla Motors, 2019) (Left) & Waymo (Waymo LLC, 2019) (Right)

There are also fully functional automated driving systems such as; Tesla's Autopilot feature (Tesla Motors, 2019), Bosch's Automated Valet Parking (AVP) (Robert Bosch GmbH, 2019) and Waymo (Waymo LLC, 2019), these take full control of the vehicle in a range of situations including; overtaking, city navigation, pathfinding, summoning to key location and parking space location. Tesla and Waymo vehicles are currently being physically tested for commercial release and production to the public to control all aspects of the DDT. These vehicles use an array of sensors, such as; lidar, radar, cameras (vision) and ultrasonic, to pinpoint both the vehicles position and its surrounding environment to the Central Processing Unit (CPU).

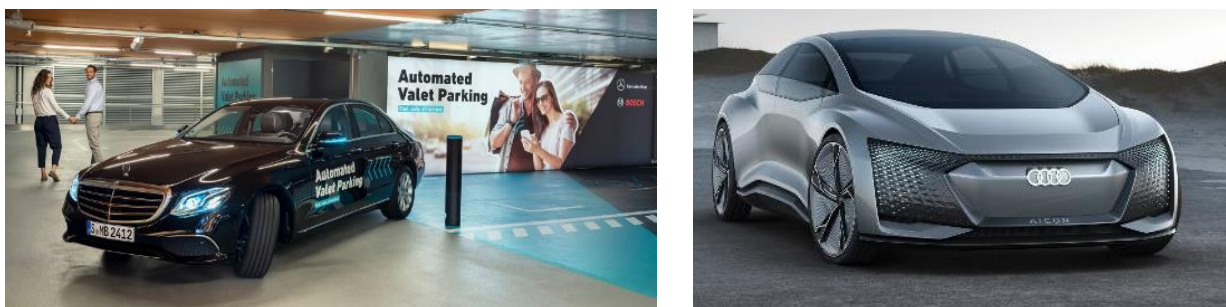


Figure 2 – Bosch Automated Valet Parking (Robert Bosch GmbH, 2019) (Left) & Audi AICon Concept (Audi, 2017) (Right)

Whereas the Bosch collaboration with Daimler to produce the AVP system is already implemented into vehicles and tested within the Mercedes-Benz museum, Stuttgart in 2017 (Transport Xtra, 2019). But as of yet, the infrastructure for this has not been made widely available, since this uses certain technologies that is required from both the vehicle and the parking garage:

“The vehicle must be equipped with a minimum of features that essentially allows it to drive itself: automatic transmission, ESP, electric parking brake and steering assistance, a start/stop function and a communication unit. On the other hand, the parking garage must have an infrastructure installed to be up to the task. Mostly, this infrastructure is comprised of a series of sensors that are used to calculate the travel distances. Also, a communication unit will be needed to ensure the car is heading in the right direction.” (Patrascu, 2019)

Extensive research has been conducted into the vulnerability of interconnected driving automation systems and the potential threat that this network would encounter. They would be connected to a network, allowing the vehicles to talk to each other to *“allow traffic to move safely and seamlessly through cities. They would use less space per vehicle. Traffic flow would be unhindered by traffic lights or other traditional driver signals.”* (Raul A. Barreto, 2019).



Figure 3 – Fiat Chrysler Jeep Cherokee (Lazarra, 2015) (Left), Charlie Miller & Chris Valasek Responsible For The Hacking (Wired, 2015) (Right)

Unfortunately, this would be highly susceptible to hackers and cyber terrorists. Back in 2015, Fiat Chrysler had to recall 1.4 million Jeeps (Lazarra, 2015), after two software engineers created a software that allowed them to remotely access different functions through a Jeep Cherokee's CAN bus control system, driven by a journalist who was part of the experiment. Using an internet connection to the vehicles IP address via 'Uconnect', they were able to gain complete control of the vehicle, altering the air conditioning, windscreen wipers, and more importantly, the throttle, brakes and steering (Wired, 2015).

A lot of time and resources need to be allocated to cybersecurity to ensure that the on-board systems within an autonomous vehicle are secure from remote threats, by means of authentication, confidentiality, alternating IP addresses and encrypted data entry points. (Waldron, 2016)



Figure 4 – Stop Signs That Have Confused An Autonomous Vehicles Neural Network (Highfield, 2017)

It has also been discovered that an automated vehicles' sensors can be easily confused, simply by placing stickers on road signs, such as stop signs, misreading them as speed markers, demonstrated within Figure 4. This could easily lead to the image recognition software unaware of crucial details about the road environment, running through junctions and causing accidents that it's supposed to avoid. (Highfield, 2017)

This point will be discussed further in depth within section 4).

2.2. MATLAB's Automated Driving Toolbox

With the introduction of the 'Automated Driving Toolbox' to MATLAB in 2017 (Mathworks, 2017), Mathworks has introduced immense functionality into an already powerful tool. With the prior ability for object detection using visual sensors, new functions have increased the programs capability to manipulate snapshots taken from a camera, detecting the lanes of the road and plotting lane markers to guide the ego vehicle, along with compiling detections from visual, lidar, radar, ultrasonic and infrared sensors and annotating them onto each snapshot frame for a visual representation of what the ego vehicle is perceiving. (Corless, 2018)

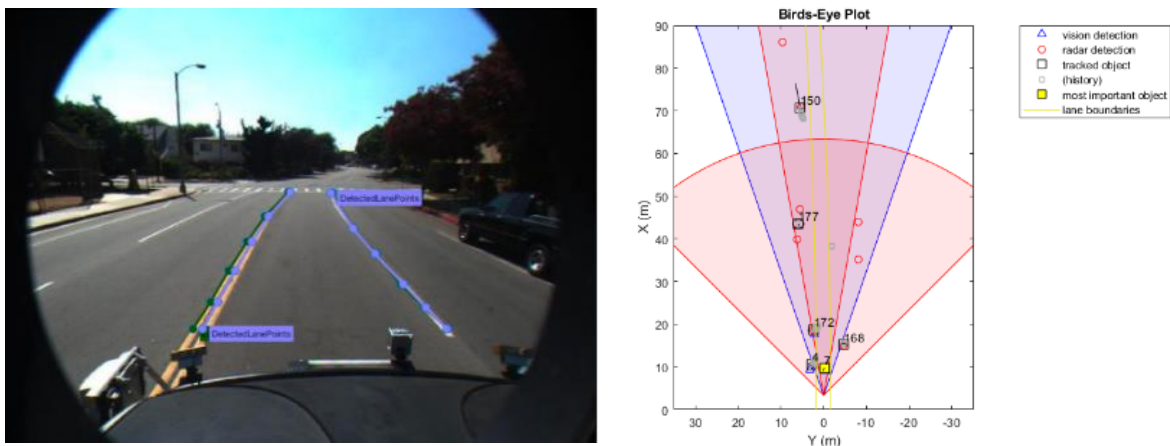


Figure 5 – MATLAB Automated Driving Functionality; Detecting Lane Markers (Left) & Compiling Sensor Detections Data Onto A Bird's Eye Plot (Right) (Mathworks, 2017)

MATLAB and Simulink have been used to develop systems that currently use the Automated Driving Toolbox. This includes a self-driving taxi, developed by Voyage Auto, where longitudinal operations are governed by a Simulink model using a standalone Robotic Operating System (ROS) to perceive and control a form of Adaptive Cruise Control (ACC). The system was created and simulated using Simulink, then the data was analysed in MATLAB to plot graphs of velocity and acceleration to determine if the controller had a reasonable control of the vehicle and if the ride was comfortable for the passengers inside. The vehicle was then tested in the real world and the model was altered to impose limits on acceleration and jerk, once the model had been refined, passengers from a retirement village in Florida became the first people outside the company to use the vehicles to assess its functionality and feeling of safety amongst the passengers. (Mond, 2018)

Companies such as Caterpillar, have employed a machine/deep learning web-based infrastructure using MATLAB to analyse sensor data and query ground truth metadata and implementing the system into a multitude of machinery, such as loaders, bulldozers, excavators and off-highway trucks used for quarrying.

This enables a systems analyst to be off-site and have remote control and access to a fleet of machinery, increasing productivity and reducing staffing costs, by means of a physical human operator not being required to oversee the machinery during its operation. (Mianzo, 2017)



Figure 6 – Applications Of MATLABs Automated Driving Toolbox; Caterpillars Deep-Learning Infrastructure (Mianzo, 2017) (Top) & Scania's Automated Emergency Braking System (Andersson, 2016) (Bottom)

Scania have used MATLAB to create a similar infrastructure as previously discussed, to develop an Advanced Emergency Braking System (AEBS) for their fleet of articulated lorries and other heavy vehicles, “to reduce the risk of rear-end collisions, in 2015 the EU mandated advanced emergency braking systems (AEBS) for all new vehicles.” (Andersson, 2016)

Using the sensor fusion aspect of Simulink, various sensors collaborated to create a virtual picture of the road in front of the test rig, this was then analysed using MATLAB to debug and refine the design using the 'Driving Simulator' application inside of MATLAB, unifying all the sensor data and annotating the video recorded by the onboard camera for an easy visualisation, before testing the system with a controlled road test. (Andersson, 2016)

2.3. Raspberry Pi

MATLAB includes hardware support for numerous amounts of different single board computers, such as Raspberry Pi (Yang, et al., n.d.), Arduino (Tomar & Sengar, 2016), Android OS and the KL25Z (Herniter, 2016). This means that there is a multitude of possible electronic devices that can be designed and created with a connection to MATLAB, wired or wireless, for the purpose of data gathering, sensor detection and even an automated driving system. MATLAB can then make calculations to determine, e.g. how much steering is required, communicate with the appropriate servo motor via the GPIO pins of the controller board, and apply an adequate degree of steer to control the vehicle within set parameters.



Figure 7 – R/C 3D Printed Vehicle Used As A Base Vehicle For This Project (Banana_Science, 2018) (Left) & Raspberry Pi And Arduino Laptop (Electronics For Everyone, 2017) (Right)

For this project a Raspberry Pi will be used in conjunction with MATLAB to control the servo motors of a 3D printed vehicle project sourced on Thingiverse, courtesy of Banana_Science (Banana_Science, 2018). A battery pack will need to be created to supply power to the Pi control board, allowing it freedom to venture without being tethered to a mains power supply. For this, Lithium-ion cells will be connected together in a parallel formation to power the Pi for several hours. One particular project uses this method with three Li-ion cells to create a Raspberry Pi powered laptop (Electronics For Everyone, 2017), this technique will be taken on board to power the Pi, allowing it to manoeuvre around a pre-set course/track and to keep the vehicle between the white lines, as it would in a real-life situation.

Calculations would need to be made to determine how much power is required to power the components and how long the cells would retain power before they would need recharging (Multicopter, 2019). These will be included within the methodology for proof of concept, but both the cells that are used in the battery pack and an online Li-ion battery calculator were researched for validation of the calculations and basic but important information about the battery cells. (Second Life Storage, 2019)

2.4. Potential Sensors

Research was conducted onto the various types of sensors that are used within real world automotive vehicles, that are also compatible with the Raspberry Pi, ensuring that communication with MATLAB was also possible. Several sensors were picked and compared based on key necessities and expectations for that job at hand.

The below sensors are all used to detect motion, distance and map the environment surrounding the ego vehicle.

Infrared – IR sensors emit infrared radiation into the surrounding environment and receives a signal back which can then be manipulated to either be used as obstacle detection or a proximity sensor. They are one of the sensor modules frequently used by electronic hobbyists and vehicle makers alike. With the automobile industry, most new vehicles include close proximity reverse parking sensors, while autonomous vehicles feature more expensive and long-range versions of the same sensors to detect any objects around the vehicle. (Ravi, 2018)

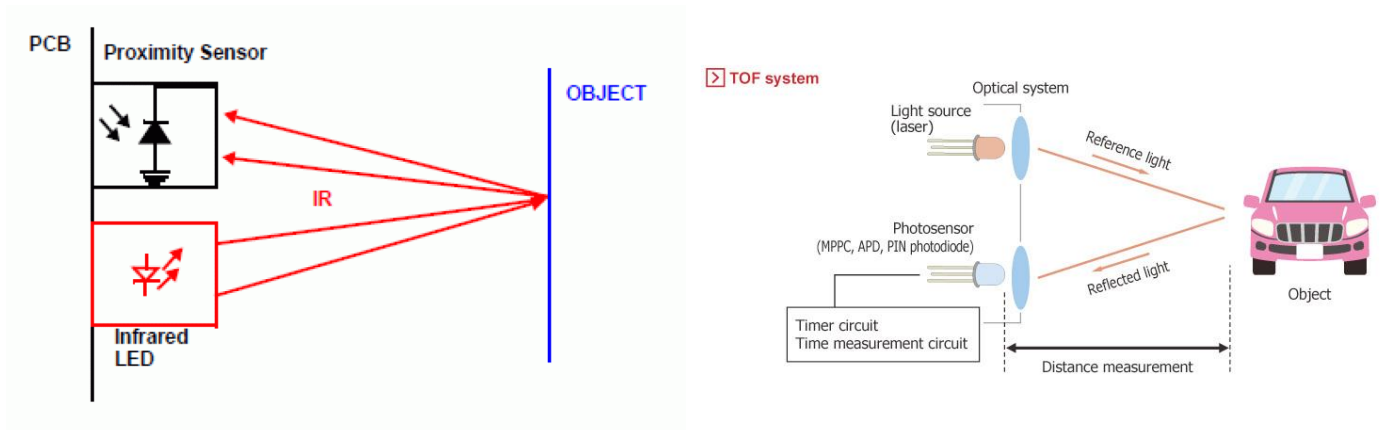


Figure 8 – Operation Of An Infrared Sensor (Burnett, 2017) (Left), & Of A Lidar Sensor (Schweber, 2019) (Right)

Lidar – Light and raDAR sensors fire a laser beam and sweeps the area to map out objects in the environment. “The application of lidar for autonomous mobile devices requires scanning of a wide area both vertically and horizontally to provide a point cloud of distance measurements.” (Merriam, 2016) Most Lidar sensors for autonomous vehicles are large in size and therefore costly, this project is a small-scale version requiring an accurate but inexpensive sensor.

Radar – Radio Detection and Ranging systems create radio waves that travel at the speed of light. They can be directed out and sweep the surrounding environment, reflecting off any object in its path. “The transmission of these signals and the collection of returned energy that bounce off of objects in the path of the radar’s transmission (called returned pulses) is what allows radar to be used to detect objects and range them, meaning establish their position and distance relative to the radar system’s location.” (Thomas Net, 2019)

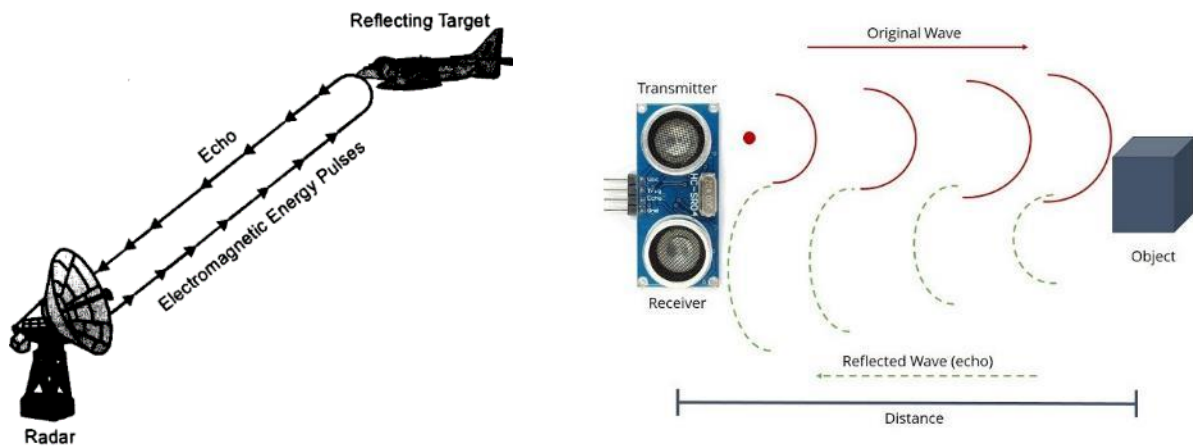


Figure 9 – Operation Of A Radar Sensor (Masinde Muliro University of Science and Technology, 2019) (Left), & Of An Ultrasonic Sensor (Seeed Studio, 2020) (Right)

Ultrasonic – By emitting high-pitched sound waves into the environment, the echo reflected back can be used to calculate the distance to an object by using the speed of sound and the time taken until the echo is received. Ultrasonic sensors can be used to measure the distance to many solid objects and one of the few that can penetrate liquids where they can be used to clean surfaces and submerged objects. “Ultrasonic sensors are also used as level sensors to detect, monitor, and regulate liquid levels in closed containers (such as vats in chemical factories).” (Jost, 2019)

Infrared and Ultrasonic sensors were chosen for their accuracy, speed and cost. All the work undertaken to implement these sensors into the ego vehicle is included within section 8.1.1.3.

3) Methodology

To undertake this project, an ego vehicle fitted with sensors and servo motors would be required for MATLAB to control. The code and functions that operate the vehicle could then be constructed and validated. Appropriate alterations would be made to ensure the best performance and effectiveness of both the script and the ego vehicle, while manoeuvring along scaled down roads with both solid and dashed road lane markers. Construction of both the code and ego vehicle features within the appendix, section 8.1, while the code itself in section 8.5.

Object detection was implemented into the code to classify objects that are within view of the camera (appendix section 8.2), along with Infrared & Ultrasonic sensors to detect the distance of those objects with the intention of stopping the vehicle before a collision is caused. The integration for which is included in the appendix, section 8.1.1.3, while the testing procedure features within section 3.2.5.

3.1. Virtual Code Validation

To validate the effectiveness of the code, it was tested virtually within MATLAB. A video was supplied as part of the Automated Driving course taken in July 2019, this proved instrumental in determining the accuracy of the script and the perceptive functions within. The optimisation of each step of the perceptive process was undertaken while construction was taking place, as seen within Figure 10.

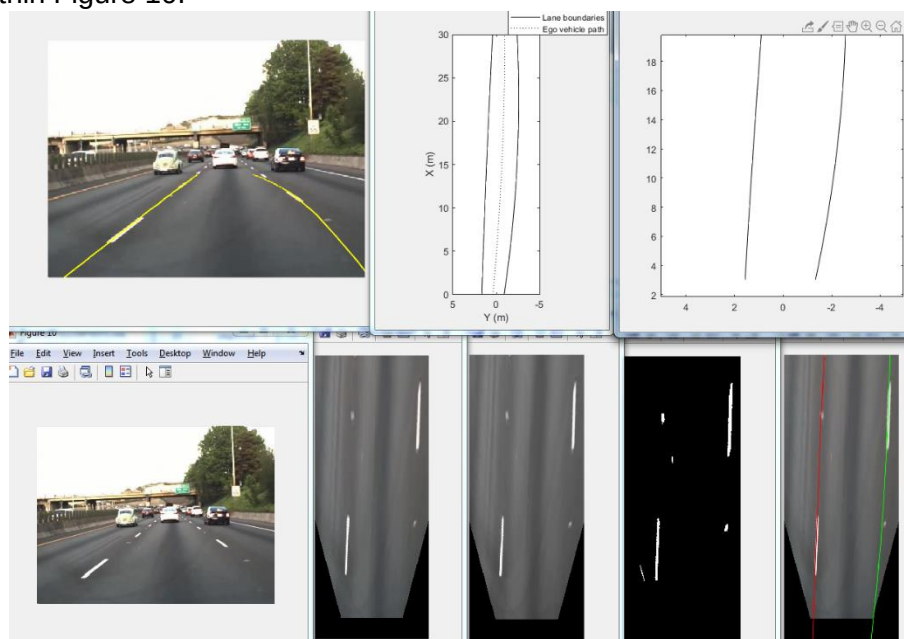


Figure 10 – Screenshot Of Every Function Within A Separate Figure During The Validation Process

Pre-calculated lane boundaries were provided during the Automated Driving course, this raised the expectations of the author before the actual script development took place. This meant that when the author recreated the script for this project, the original accuracy could not be achieved. It was during this phase that the sensitivity factor was discovered and implemented for the '*segmentLaneMarkerRidge*' function and the resultant black and white binary image "BW".

Experiments were undertaken to fine tune the value for sensitivity. This value was set for the entire video because no solution could be found to alternate/automate the changing of this value throughout the video, depending on the situation (shiny road surface causing 'noise', shadows and lane changing). This is displayed within Figure 11.

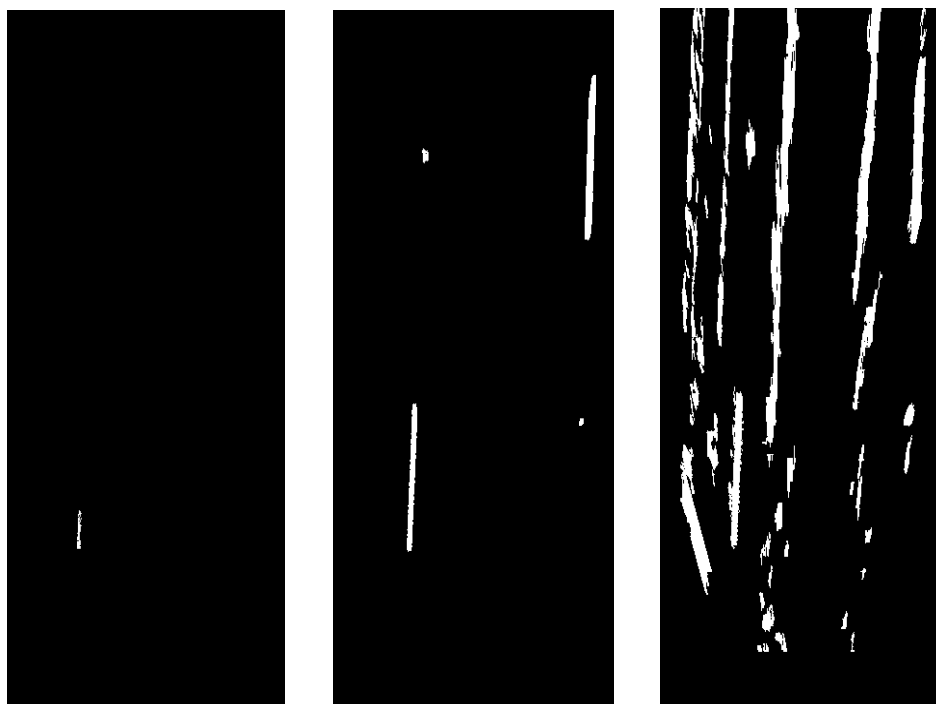


Figure 11 – Varying The Level Of Sensitivity Within The "BW" Image; Too Low (Left), Optimal (Middle) & Too High (Right)

Once the perception was optimised, the steering portion of the script was formulated using this video. The original iteration of these commands took place without the Raspberry Pi connected, focusing purely on the steering implementation.

Construction of the code that controls the vehicles' steering will be discussed within appendix section 8.1.2, but this took several iterations to perform. The basic concept was there, but the ability to link the perception and the vehicle control was missing.

An attempt to craft a link between the two sections by using the 'Stanley Method' (Snider, 2009) and the '*lateralControllerStanley*' function was made. This computes a steering angle that adjusts the vehicles' pose to match a reference pose, while in forward motion (Mathworks, 2019).

The drawback of this function was that the reference pose that determined how much steering angle was required, must be stated/determined before this function became useable, adding another problem that must be solved, increasing the complexity, seen in Figure 12.

```
refPose = [4.8, 6.5, 2];  
% [meters, meters, degrees]  
currPose = [2, 6.5, 0];  
% [meters, meters, degrees]  
currVelocity = 2;  
% meters per second  
steerCmd = lateralControllerStanley(refPose,currPose,currVelocity)
```

Figure 12 – Calculating Steering Angle Command Path Using The Stanley Method

The final solution became evident when reading the MATLAB documentation on the '*computeBoundaryModel*' function (Mathworks, 2019). The authors previous knowledge of this function was limited to what John Moon had taught during the Automated Driving course that had been undertaken.

While struggling with this issue, it was discovered that each lane could be calculated the same way they had been previously within the script to create the variable "*yOut*". This time however, the resultant variables created were column vectors that could be used to calculate the direct centre of the lane, whereas before, the outputs were parabolic structures so they did not provide useable values that could be referenced to, causing confusion to the author on how best to overcome the situation, Figure 107.

The documentation also provided the method to creating the variables "*xWorld*", "*yRight*", "*yLeft*" and "*yCentre*", mentioned in section 8.1.2, where the ability to validate and visually see the ego vehicles path with the bird's eye plot was discovered, Figure 108.

This method proved fruitful. A link to control the steering of the vehicle had been achieved, as seen within Figure 72, where the connection was made through MATLAB to the steering servo. The video was run again with the code controlling the steering and the vehicle reacted successfully, turning the wheels along with the bends in the road, just as the vehicle did within the video. This meant that physical testing could now be undertaken to experiment with the system in a real environment.

3.2. Physical Testing Procedures

This section details how both the MATLAB script and the vehicle performed in the real world. As the limitations of both became apparent, new problems arose, requiring alterations to be made to the code and the vehicle to resolve these.

The vehicle was tested with handmade scaled down representations of a real-life single-lane road, one straight road with solid markers, one corner piece with solid markers (section 3.2.1) and one straight road with dashed markers (section 3.2.2).

Different camera mountings and positions on the vehicle were trailed during this phase, these are detailed within section 3.2.3. A fisheye lens was also tested to determine how this affected the performance of the vehicle during the physical testing process, this experiment will be detailed within section 3.2.4.

The intent was to determine both the durability of the code and its effectiveness to stay between the lines. The modifications that were required to ensure the best performance in a range of scenarios will also be detailed.

3.2.1. Solid Road Lane Markers

The physical testing began by creating two road surfaces, one straight, one curved. The roads were created using sheets of black A3 paper, black insulation tape to join the sheets together, and white insulation tape to represent the road markers. Two A3 sheets of paper joined together lengthways formed a straight road, the curved road however, three sheets of paper were taped together lengthways but angled to one another, as seen within (Figure 14), for a total of 45° .

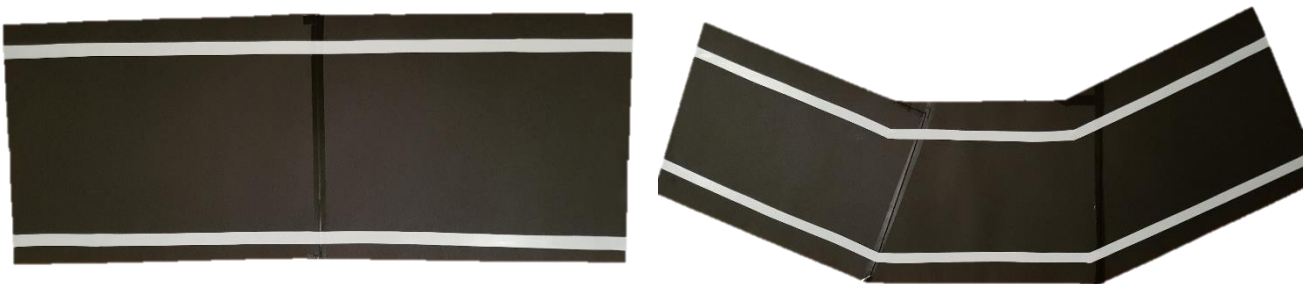


Figure 13 – Physical Representations Of A Straight Road (Left) And Curved Road (Right)

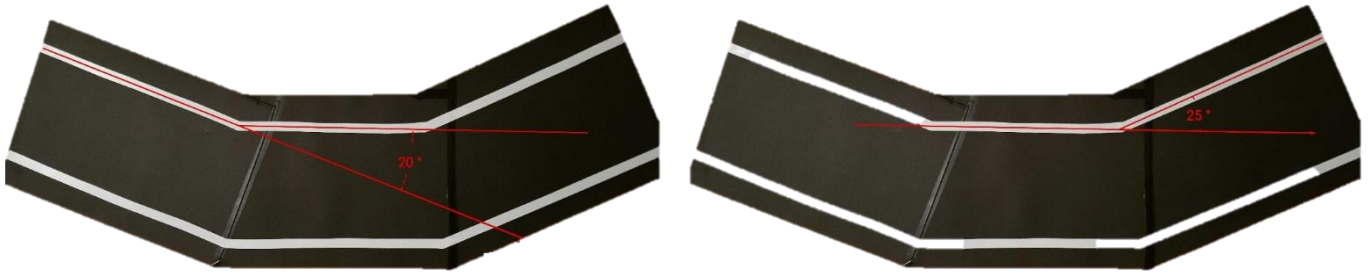


Figure 14 – Measured Angles Of Each Section Of The Curved Road

For this test, the road markers were solid unbroken lines that spanned the length of each road. Each road had two sides, one side with a wide gap between the left and right lane, and the other side has a narrower gap between the lanes, which was just wider than the ego vehicle. (Figure 15)

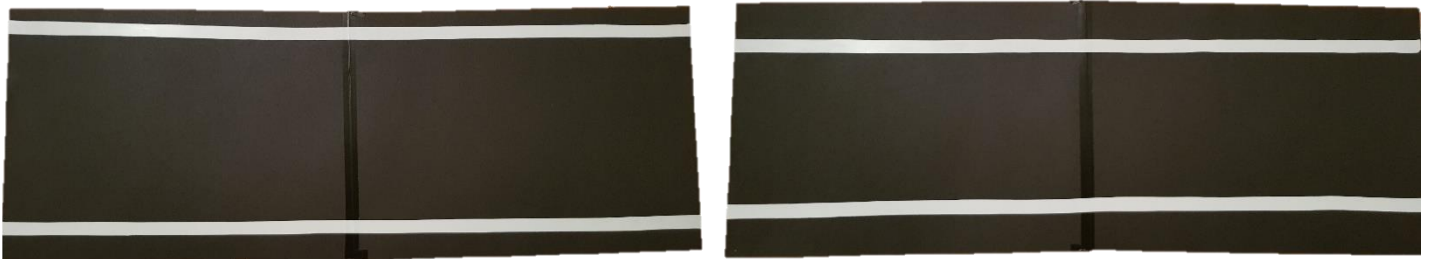


Figure 15 – Wide Lane Road (Left) & Narrow Road (Right)

The initial testing procedure began implementing the same script used within the validation process. As seen below, this did not perform as expected. (Figure 16)



Figure 16 – First Attempt At Physical Testing Using Vertical Camera Mount

To solve the issues that occurred, modifications were made to the code and the functions used within. The *'while'* loop was commented out so that only one frame was processed at a time, each functions output can be displayed using *'imshow'* so each step can be analysed to discover the source of the problem. This experiment took longer than anticipated to perform, it took place both at university and inside the student accommodation the author lived in during term time. Different camera mount heights, angles and positions on the ego vehicle, along with the camera lens' were used during this procedure. They were the biggest changes made and had the most impact upon the accuracy of the system, but are detailed within sections 3.2.3 and 3.2.4.

Each environment provided difficulties, and therefore, required different settings to achieve a working model. For example, the wood effect laminate flooring at the student house created reflections, and hence, white spots that were detected during the computation of the binary image *"BW"*, causing issues right from the beginning.

The purpose of the solid white marker road was to hopefully eliminate the *'noise'* within this image by providing a solid line for the function to process, ensuring the parabolic lanes could *'snap'* to those instead of the unwanted noise. The opposite of this took place, even the slightest amount of disturbance caused the lane detection algorithm to be misled, which in turn, caused the steering of the ego vehicle to encounter incredible amounts of erraticism. To counteract this error, the sensitivity factor for the *'segmentLaneMarkerRidge'* function became incredibly useful. This value was fine-tuned using the same procedure as with the code validation process. (Figure 17)



Figure 17 - Changes To The Sensitivity Factor; Too Low (Left), Optimal (Middle) & Too High (Right)

The Region Of Interest (ROI) was adjusted constantly throughout this and the subsequent experiments to find the best medium. The minimum and maximum X-axis values within the ROI were one of the main contributors to the accuracy and stability of the parabolic lanes. The minimum value was adjusted but nowhere near the extent as the maximum value, this was originally set to zero, the origin of the camera, but was eventually increased.

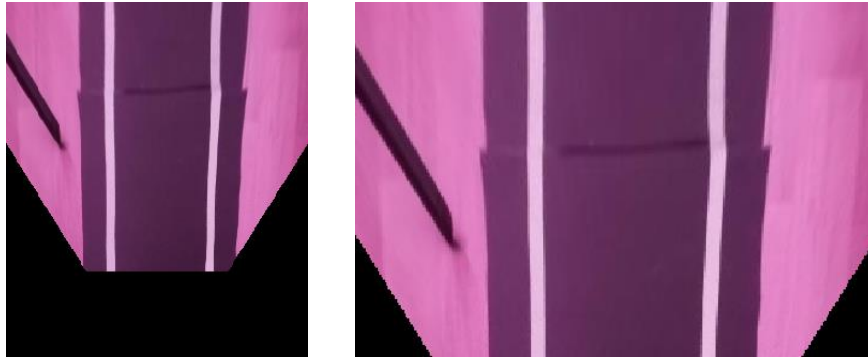


Figure 18 - Effect Of Changing The Minimum ROI Setting On The Bird's Eye View; 0m (Left) & 0.25m (Right)

Shown in Figure 18, setting the minimum X-axis to '0m' produces a large black "negative" area within the bird's eye view, this is the environment that the camera cannot see, and is effectively useless. This was removed by increasing the minimum X-axis value to '0.25m' to focus the functions attention purely on the road within the camera's frame by removing the redundant area. This improved the stability of the algorithm, but not enough to counteract the overall issue of instability.

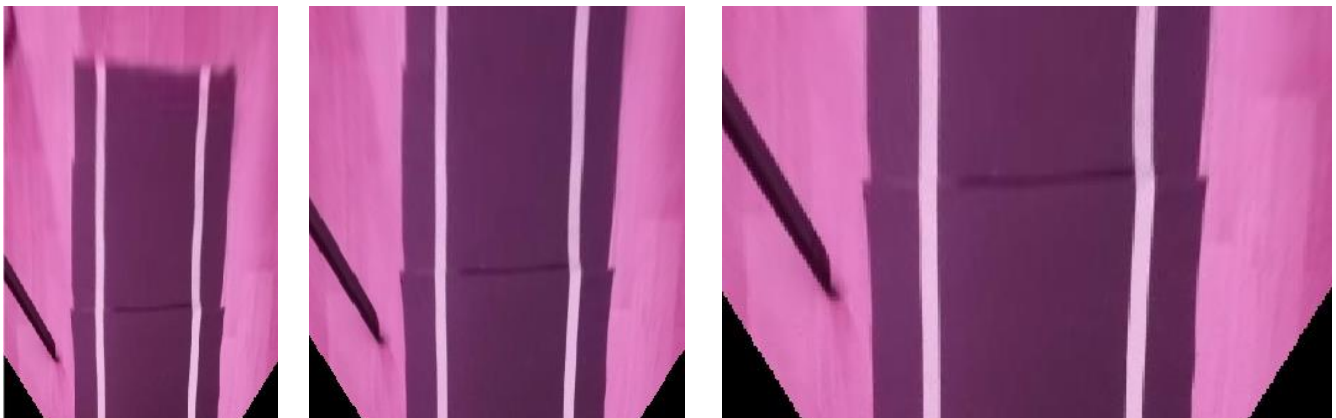


Figure 19 – Effect Of Changing The Maximum ROI Setting On The Bird's Eye View; 1m (Left), 0.8m (Middle) & 0.6m (Right)

The maximum X-axis value directly dictated how far ahead the ego vehicles' camera could see within the bird's eye view imagery (Figure 19). This had a substantial influence on the stability of the lane detection algorithm. As seen within Figure 20 – next page, while the ego vehicle was manoeuvring around the corner with a maximum value of '0.6m', only half as much of the left lane marker can be seen, compared to the right lane marker spanning the entire length of the image.

Because of this, the left lane could not be detected, meaning that both parabolic lanes were designated onto the right physical lane marker, driving the variable “ y_{Centre} ” to be directly over the right lane marker, which in turn, guided the vehicle to follow the lane marker instead of the centre of the lane, As seen within Figure 20 – Top.

When this number was reduced to ‘0.4m’, each lane marker could be seen for the entire length of the image, dramatically improving the stability and accuracy of the algorithm, enabling the vehicle to correctly manoeuvre the corner in the centre of the lane, Figure 20 – Bottom.

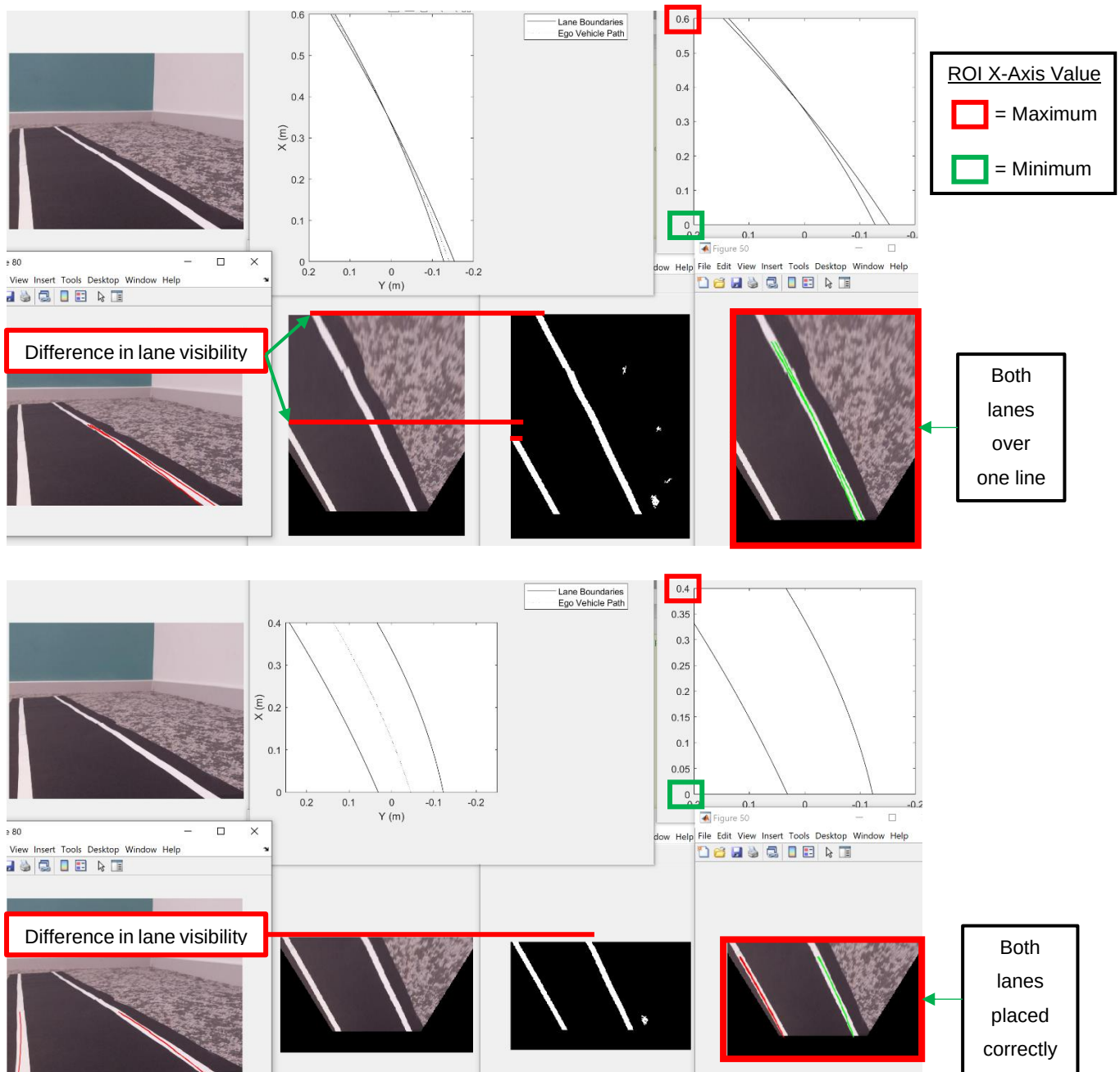


Figure 20 – Effect Of Changing The Maximum ROI Setting When Navigating A Corner; ‘0.6m’ Causes Issues (Top) & ‘0.4m’ Proving More Efficient (Bottom)

Unfortunately, using '0.4' for the maximum ROI X-axis, only became viable when a corner is approached. If the road was straight, it was found that there was not enough visibility of the road markers ahead to create accurate and stable parabolic lane boundaries or enough time to guide the steering of the vehicle without causing understeer.

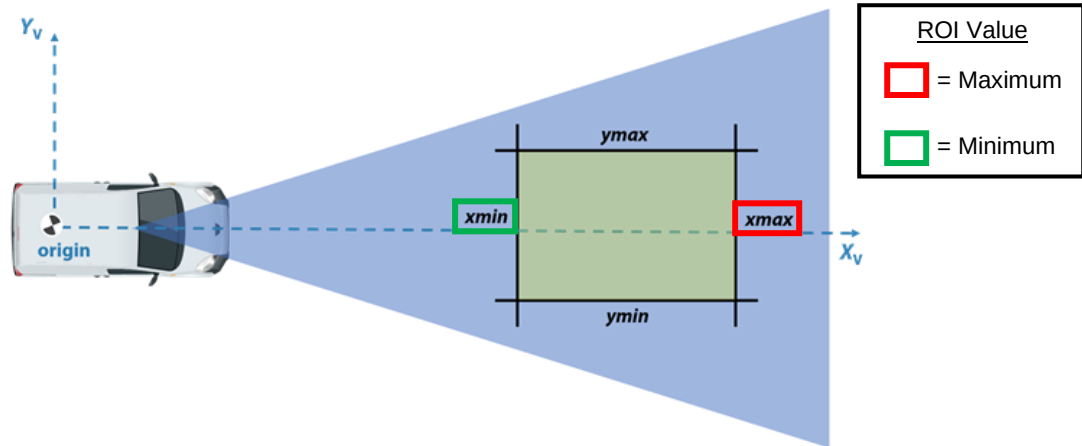


Figure 21 – Region Of Interest For Use With Converting An Image To A Birds' Eye View (Mathworks, 2019)

This was a problem. It required a dynamically changing ROI X-axis, to ensure there was always enough visibility of the road in front to take appropriate action, a demonstration can be seen within Figure 21. After a lot of trial and error, this process was able to become completely automated by the script. It allowed the accuracy of the lane boundaries to stay relatively high, overcoming the consistently changing environment around the ego vehicle, that would otherwise disrupt the flow of the active system.

Three processes were undertaken to overcome this issue.

1. The code would need to determine how to gauge the difference in lengths between the parabolic lane markers to increase/decrease the maximum ROI X-axis value so that the lane markers are roughly equal in length.
2. Reduce the minimum ROI X-axis value, should the maximum X-axis value be decreased less than the original minimum X-axis value, so the road directly in front of the vehicle could always be seen.
3. Set the parabolic lane markers to the same length to prevent any further misdirections.

The first step was to find where the parabolic lane boundary lengths were stored. Once the "findParabolicLaneBoundaries" function had been used to create the variable 'lanes', it generated parameters about the lane markers from what it had perceived within the real-world. Inside these parameters were the maximum lengths of the parabolic lane boundaries in relation to the ego vehicle, named 'XExtent'. This variable contained, amongst others, a number that MATLAB could easily identify and use to manipulate other numbers and variables.

```

if abs(lanes(1,1).XExtent(1,2) - lanes(1,2).XExtent(1,2)) > 0.2
    if (lanes(1,1).XExtent(1,2) > lanes(1,2).XExtent(1,2))
        roi(1,2) = lanes(1,2).XExtent(1,2);
    elseif (lanes(1,1).XExtent(1,2) < lanes(1,2).XExtent(1,2))
        roi(1,2) = lanes(1,1).XExtent(1,2);
    end
    if roi(1,2) < 0.4
        Stop = 1;
        % writePosition(d,90)
    end
elseif lanes(1,1).XExtent(1,2) || lanes(1,2).XExtent(1,2) < 0.6
    roi(1,2) = abs(lanes(1,1).XExtent(1,2) + lanes(1,2).XExtent(1,2))/2;
    if roi(1,2) < 0.55
        roi(1,2) = roi(1,2)*1.05;
        if roi(1,2) >= 0.8
            roi(1,2) = 0.8;
        end
    end
end
end

```

Figure 22 – Code to Increase/Decrease the Maximum ROI X-axis Value

With this value ascertained, the first process could begin. An 'if' statement was created to subtract the absolute value of both parabolic lane marker lengths from one another. If this value was greater than 0.2, then the maximum ROI X-axis value would equal the shortest length. Should the maximum ROI length become less than '0.4m', the ego vehicle would also stop, as there is not enough reaction time for the system to operate efficiently or it may be signifying the end of the road. Figure 22.

The maximum ROI X-axis would then need to increase back up to its original value of '0.6m'. Connected to the previous 'if' statement, an 'elseif' logical statement would be proved true if either parabolic lane marker length was less than '0.6m'. If the previous statement was false, meaning both lane markers are within tolerance or were the same length, then this new statement would come into effect. Figure 23.

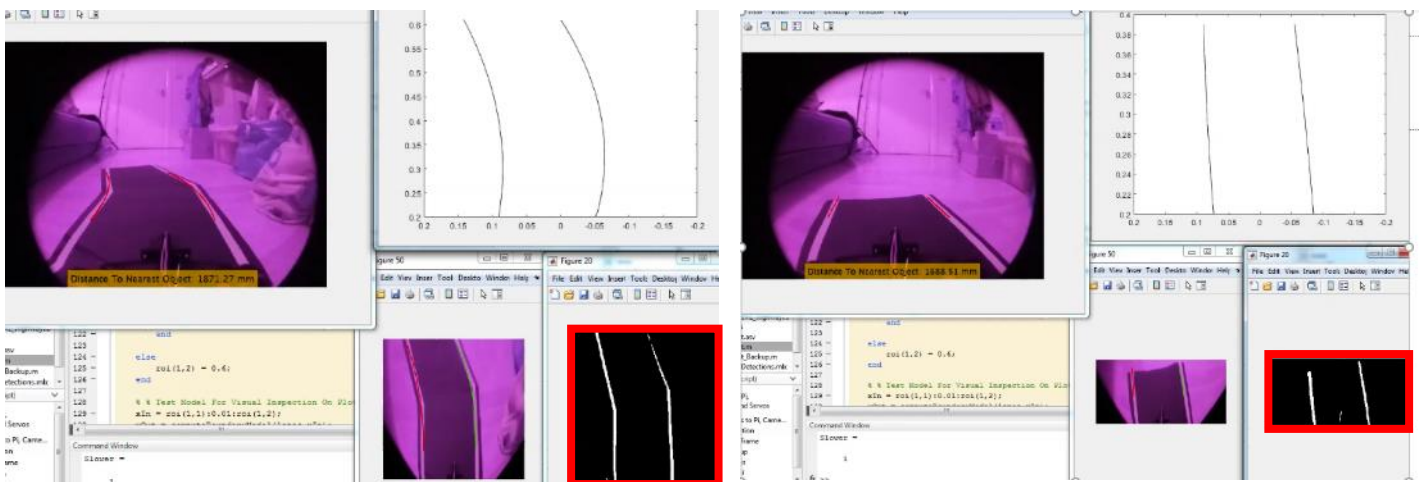


Figure 23 – Effect Of The Automatically Adjusting The Maximum ROI X-axis Distance

This entails overwriting the maximum X-axis value to become the average of both parabolic lane marker lengths. This was then expanded upon by questioning if the maximum X-axis value was less than '0.55m', if this was true the value was multiplied by '1.05' gradually increasing the value up to where it is then tested logically again. If the value became greater than or equal to '0.8m', it was set to '0.8m' which would be best suited to a straight road.

```

if roi(1,2) < 0.5
    if roi(1,1) < 0.25
        roi(1,1) = roi(1,1) * 0.9;
    end
    if roi(1,1) <= 0.2
        Stop = 1
        writePosition(d,90)
    end
    if roi(1,1) <= 0.1
        roi(1,1) = 0.1;
    end
elseif roi(1,2) >= 0.55
    roi(1,1) = roi(1,1) * 1.1;
    if roi(1,1) >= 0.25
        roi(1,1) = 0.25;
    end
end
end

```

Figure 24 – Code to Increase/Decrease the Minimum ROI X-axis Value

The minimum ROI X-axis length is also automated using an 'if' statement, Figure 24. When the maximum X-axis value becomes less than '0.5m', the original minimum X-axis value of '0.25m' is now greater than half of the maximum value. This reduces the view of the road ahead significantly, and if the maximum value equals '0.25m'. To resolve this, when the maximum X-axis value becomes less than '0.5m', the minimum X-axis value would then be multiplied by '0.9', reducing it to provide as greater view as possible.

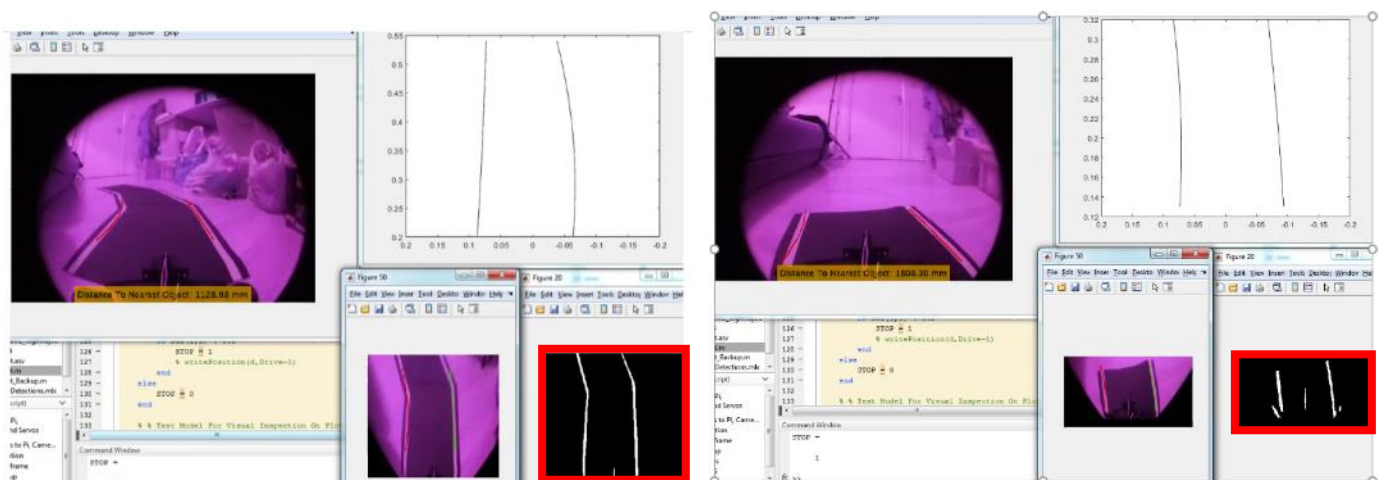


Figure 25 – Effect Of The Automatically Adjusting The Minimum ROI X-axis Distance

But if this minimum value became less than '0.1m' it would be overwritten as '0.1m' to prevent it going negative, causing further errors. If the minimum value went below '0.2m' it could be assumed that either there is an obstacle obscuring the lane markers or the end of the road had been reached. In this case it would be appropriate to tell the vehicle to stop, switching off the electric motor. The minimum ROI length would then need to increase back up, as before. If the maximum ROI X-axis length was greater than or equal to '0.55', the minimum value would be multiplied by '1.1' to gradually increase until the original value of '0.25m' is reached.

```

if (lanes(1,1).XExtent(1,2) > lanes(1,2).XExtent(1,2))
    MaxWorldLength = lanes(1,2).XExtent(1,2);
elseif (lanes(1,1).XExtent(1,2) < lanes(1,2).XExtent(1,2))
    MaxWorldLength = lanes(1,1).XExtent(1,2);
end

```

Figure 26 – Code To Reset The Parabolic Lane Boundary Lengths

The next requirement was to adjust the lengths of the parabolic lane markers to prevent any further disruptions to the code's performance and any errors occurring, Figure 26. Using the simplest 'if' statement, it was possible to assess the largest lane marker length, compared to that of the minimum length. The longest lane marker length would be overwritten with the shortest length. Since throughout the code, the lane marker lengths are set to that of the maximum ROI X-axis length multiplied by '0.9'. This guaranteed that the lengths would decrease and increase in proportion to the ROI.

The final area of refinement came with the approximate width of the lane markers within the 'segmentLaneMarkerRidge' function. The actual width of the insulation tape used to create the longitudinal lane markers was '16mm', but complications arose when this value was used as an input. The sensitivity factor within this function was severely affected by changed the width of the lane markers, seen within (Figure 27), where the sensitivity factor was kept to the same value, but the detected lanes vary, causing instability to the parabolic lane boundaries. The sensitivity was then reduced to compensate for this, but that resulted in the actual lane markers not being detected properly, counteracting any potential improvements. The best values for the lane marker widths came with using '13mm' as an input and a sensitivity of '0.35'.



Figure 27 – Effect Of Changing The Lane Marker Size Within The Code, While Keeping The Same Sensitivity Value Of 0.35; 0.01m (Left), 0.013m (Middle) & 0.016m (Right)

3.2.2. Dashed Road Lane Markers

Once the vehicle was able to navigate the solid road markers, both in a straight line and around the corner. The next procedure was to test the durability with an actual road, where the lane markers are broken up with a specified distance between each line.

To create an accurate representation of a real-life UK road, research was conducted to determine the actual length of both the longitudinal lane markers and the gaps between them. The distance between the left and right lanes on the road were kept the same as the previously used solid lines with two sides, a wide and a narrow distance, to fully recreate the same testing procedure.

The measurements for the road were attained from the UK Department of Transport's Traffic Signs Manual (Department of Transport, 2018), which details the exact specifications about the longitudinal centre lines that appear on the UK's roads. Although on real-life roads only the centre lines are broken, whereas, the edge lines that run parallel to the kerb are generally unbroken, disregarding junctions, bus stops and hard shoulders. Therefore, the physical representation created used broken lines on both sides, as if to replicate lanes on a motorway.

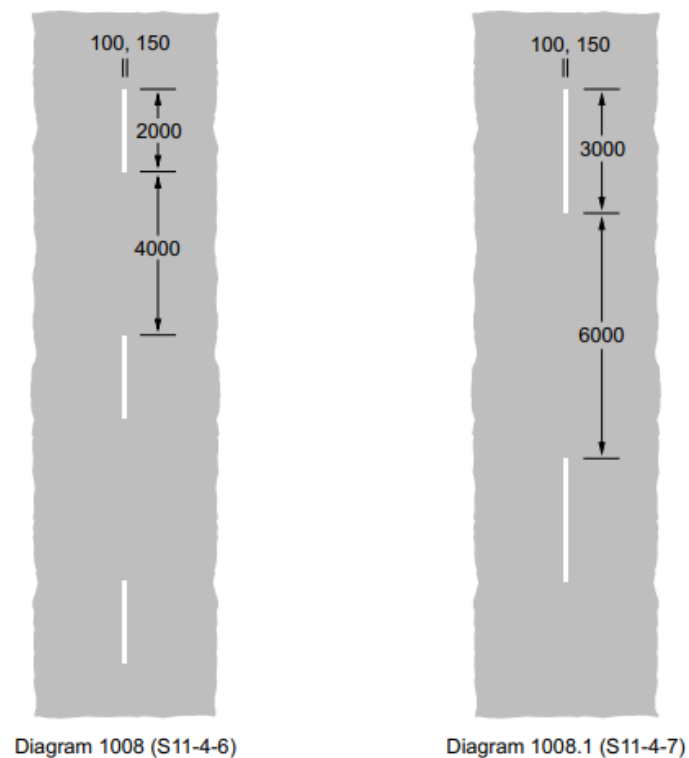


Figure 2-1 Centre lines

Figure 28 – Longitudinal Lane Markers Dimensions As Given By Department Of Transport; 40mph Or Less (Left) & Greater Than 40mph (Right) (Department of Transport, 2018)

There are two generic types of single carriageway road lane markers used within the UK, one where the permanent speed limit is 40mph or less, and the other where the speed limit is greater than 40mph, Shown within Figure 28. The ego vehicle for this project will not exceed more than 5mph, hence, the road markings for roads with vehicles travelling at less than 40mph was selected. This was convenient, because the longitudinal lane markers are closer together, aiding with the computation of the parabolic lane boundaries.

The longitudinal lane markers have a length of 2000mm, and a 4000mm distance between each line on the standard road, a ratio was required to attain a correct scale replica. To do this, the length of both the ego vehicle and the UK average car was required. The ego vehicle is 340mm long, while the UK average car measured at 4500mm (Wilson, 2019). When dividing the greater by the smaller, the ratio can be calculated.

$$\text{Scaling Ratio} = \frac{4500}{340} = 13.24$$

3-1

This shows the ego vehicle is over thirteen times smaller than the average car in the UK. Each longitudinal lane markers length and the distance between them can now be divided respectfully, to result in the appropriately scaled down lengths.

$$\text{Longitudinal Lane Marker Length (m)} = \frac{2000}{13.24} = 151.11\text{mm}$$

3-2

$$\text{Distance Between Lane Markers (m)} = \frac{4000}{13.24} = 302.22\text{mm}$$

3-3

With these dimensions ascertained, the scaled down representation could be fabricated, shown within Figure 29.

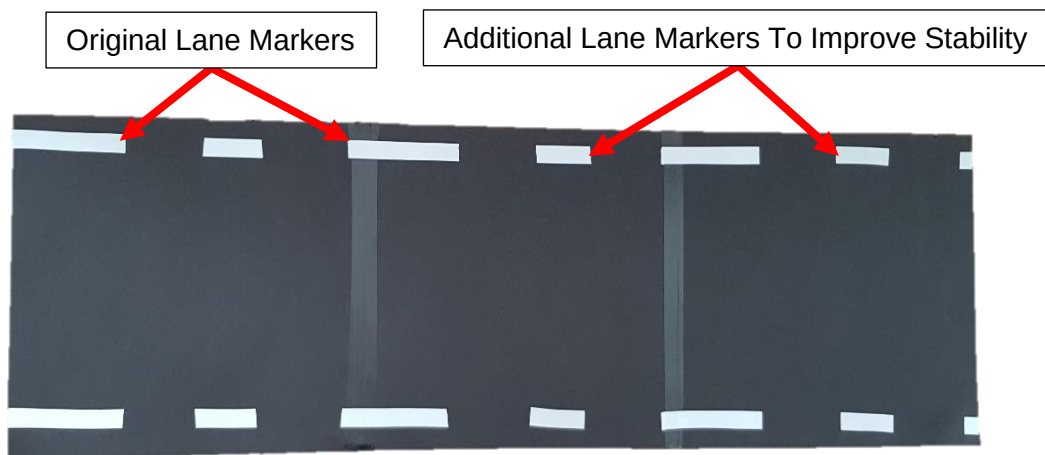


Figure 29 – Scaled Down Representation Of A Dashed Line UK Road

Unfortunately, the parabolic lane boundaries became incredibly unstable during testing due to the significant distance between each line. This made it hard for the function to join the dots which may be caused by the camera being too close to the ground, and therefore, the perspective of the camera is limited. Whereas, autonomous vehicles in the real world have cameras that are mounted at the top of the windscreen for a clear Field Of View (FOV) of the road ahead.

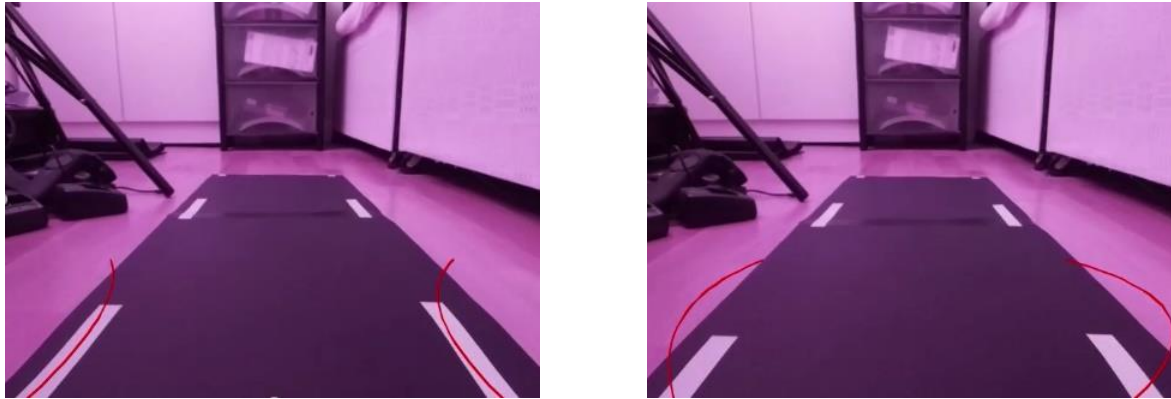


Figure 30 – Inaccuracy Of Parabolic Lane Boundaries Without Smaller Lines

Alterations were made to the code to gain the same level of performance as previously displayed on the solid lane markers, regrettably, this could not be achieved in reality. No matter what changes were made to the code or the cameras mounting position, the parabolic lanes were incredibly erratic, seen within Figure 30.

The only solution was the addition of smaller lines, 70mm long in the gaps between the longitudinal lane markers which significantly improved the performance of the code (Figure 29). Once the parabolic lanes were stable for the majority of the time, refinements were made to the ROI dimensions, the sensitivity factor and the approximate width of the lane markers inside the 'segmentLaneMarkerRidge' function, as before, to further improve the codes durability. This length of road was used throughout the physical testing, with the final iteration shown before progressing within Figure 31.



Figure 31 – Optimised Performance And Stability On The Dashed Lines

3.2.3. Different Mounts And Positions For The Pi Camera

The positioning of the camera mount on the ego vehicle, along with the pitch angle in relation to the road, had a momentous impact upon the accuracy and stability of the system.

The first attempt began by using a camera mount that had a shank of 20mm tall, while the camera was kept at 0° to the vertical plane, this was mounted directly at the front of the vehicle deliberately to give the camera a clear FOV of the road ahead. Unfortunately, there were problems that were encountered, adjusting the height of the camera on the vehicle by mounting it onto the Raspberry Pi Mount, then increasing the pitch relative to the road surface significantly improved the accuracy of the lane detection algorithm.



Figure 32 – Physical Testing Using A Camera Mount With 0° To The Vertical; Static Position (Left) & Raised Rear (Right)

During the first test procedure using the horizontal camera mount, it was immediately evident to the lack of accuracy. Figure 32 – Left, shows the first iteration where the vehicle was in its natural static position, the annotated lane boundaries do not ‘latch’ onto the physical lane markers, but once the rear of the vehicle was elevated by hand, a pitch value was provided onto the entire ego vehicle, enabling more of the road surface to appear within the cameras FOV, Figure 32 – Right.

This gave the author a clear indication that if the camera was pitched towards the roads’ surface, the ‘*findParabolicLaneBoundaries*’ function would work more effectively. This began a new experiment to the adequate amount of pitch required for the best performance within the system.

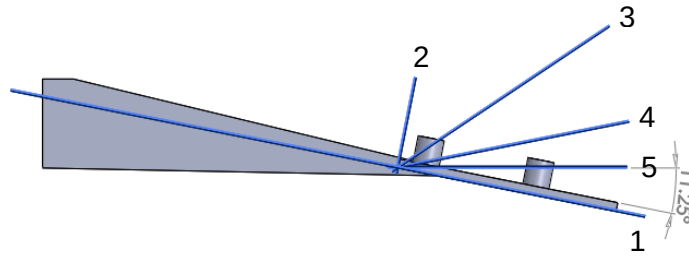


Figure 33 – Displaying The Process Of Creating Multiple Reference Planes To Gain The Desired Angle For The Camera Mount And The Numerical Order In Which They Were Created

New camera mounts were designed within Solidworks and prototypes were 3D printed, but due to the limitations of the Solidworks program creating reference planes, specific angles could not be used (Figure 33).

After a lot of inconvenience, reference planes were created using other reference planes to get the desired angle between them. The aim was to do increasing increments of 10^0 of pitch, unfortunately, the closest values that were able to be achieved were 11.25^0 and 22.5^0 , Figure 34.

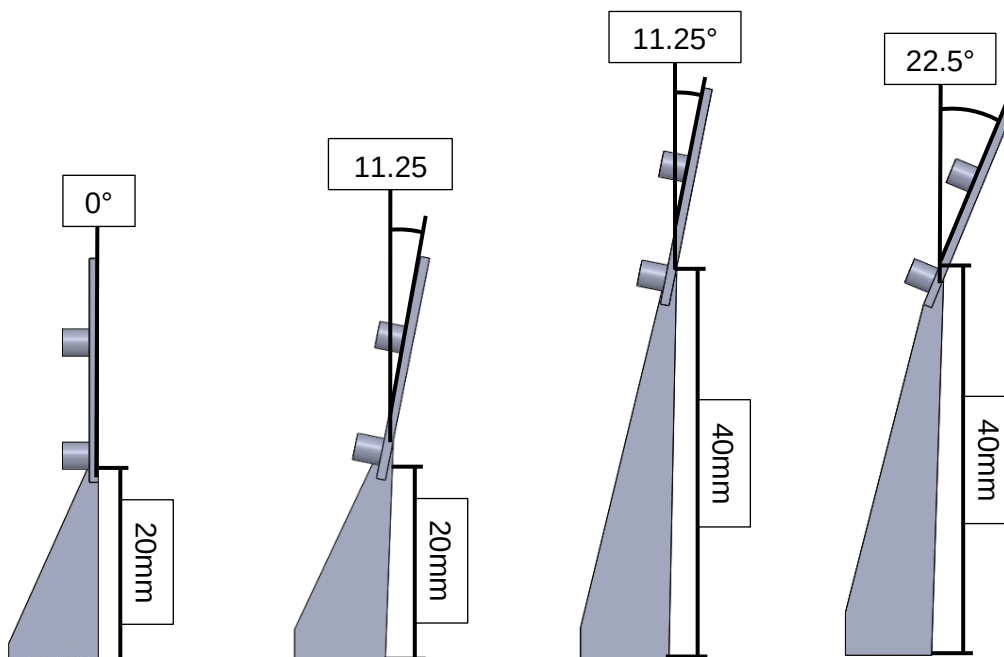


Figure 34 – Various Iterations Of Camera Mount Designs; Initial Vertical Mount 20mm Shank (Left), 11.25° Pitch 20mm Shank (Middle Left), 11.25° Pitch 40mm Shank (Middle Right), 22.5° Pitch 40mm Shank (Right)

The same tests and alterations were conducted on each mounting to achieve the same accuracy that had been witnessed within the code validation process. The full details of these changes are found within the previous section (3.2.1).

Using 11.25° of pitch, raised the horizon above halfway up the image. This allowed the code to have a greater view of the road compared to the original mount, increasing both the accuracy of the parabolic lane boundaries and the performance of the systems 'lane keeping' ability. This modification required barely any alterations to the script, other than designating the pitch value within the '*monoCamera*' intrinsic properties, in order to manipulate the image as expected. (Figure 35 – Left)



Figure 35 – Effect From Changing The Pitch Of The Camera Mounting; 11.25° (Left) & 22.5° (Right)

22.5° of pitch was expected to be the best position, providing a more direct view of the road ahead. Unfortunately, because of the increase in pitch, the road that was directly in front of the vehicle was obscured from the cameras vision, causing the accuracy to drop. Significantly affecting where the code determined the middle of the lane was located, and thus, the angle of steering the vehicle was subjected to. Figure 35 – Right is the best example achieved for a still image, but once the entire 'while' loop was initiated, they flailed much alike a "Wacky Waving Inflatable Tube Guy" (Firebox, 2020).

The parabolic lanes also encountered significant trouble towards the maximum X-axis value of the ROI. During interrogation of the binary "BW" imagery, it was clear that more reflections were witnessed, and noise caused within the image, hence the lane boundaries jumped around even when the ego vehicle was stationary.

To overcome this situation, the code was subjected to more strenuous modifications than exhibited previously when using the 11.25° camera mount.

It was conclusive throughout testing that the 11.25° camera mount had the best compromise between algorithm accuracy and the cameras FOV of the road surface and everything ahead. Henceforth, it was implemented onto the vehicle and used for the ongoing duration of this project.

3.2.4. Varying the Lens Used On The Pi Camera

A separate experiment was undertaken to determine the capability of both the camera and the functions within the code, when a fisheye lens is used. The fisheye lens provides a 180° view in front of the camera, allowing for a greater depth of field, the idea for which, originated from when the ego vehicle was manoeuvring around a corner, where one longitudinal lane marker left the cameras' FOV entirely. Figure 20 – Top.

There was an initial concern before using a fisheye lens, where it may deform the camera imagery to the extent where the code would not be able to detect the physical lane markers, or, if it was able to detect them, they may be deformed, inhibiting the vehicles ability to follow the road correctly. This was found to be false. With minor adjustments made to the sensitivity factor and ROI width to compensate for the extra environment that was now visible, the fisheye lens proved to be an imperative component for the ego vehicle to include.

The final consideration was that the addition of a fisheye lens would now affect the cameras' intrinsic properties. The Focal Length (FL) would be affected by the inclusion of a wider FOV, this was evident upon implementation of the lens when the original snapshot taken from the camera, became slightly blurry. MATLAB requires the FL to be inputted as a two-element vector containing the 'X' & 'Y' dimensions in pixels (Mathworks, 2019)

$$f_x = F * s_x$$

3-4

$$f_y = F * s_y$$

3-5

Where:

F = the FL in millimetres (mm)

s_x & s_y = the number of pixels per mm in the 'X' & 'Y' direction

f_x & f_y = FL in pixels

To equate the FL in pixels, the value for pixel size (mm) was provided within the specifications for the Raspberry Pi Cam, stated as "1.12 μm x 1.12 μm " (μm = micrometre) with a FL of 3.04mm (Raspberry Pi Foundation, 2019). This enabled the calculation of the FL for the standard lens in both 'X' & 'Y' directions.

$$\text{No of Pixels per millimetre} = \frac{1}{0.00112(\text{mm})} = 892.86 \text{ pixels/mm}$$

3-6

$$\text{Focal Length (pixels)} = 3.04 * 892.86 = 2714.28 \text{ pixels}$$

3-7

Since both 'X' & 'Y' direction values are identical, the same number was inputted into the two-element vector. This value was used for the standard camera throughout testing, but once the fisheye lens was applied to the Pi camera, the image was incredibly blurry. The focal length was then adjusted incrementally until the image was as clear as possible, ending with a value of 500 by 400 pixels.

By using a wide-angle lens, the problem that occurred where an entire lane marker vanished from view while navigating a corner was reduced, enabling the ego vehicle to approach with a higher yaw angle than previously possible, before the lane markers disappear from the cameras' FOV. Below is a comparison of both standard and fisheye lens', taken during the same point while undertaking a corner, within Figure 36.

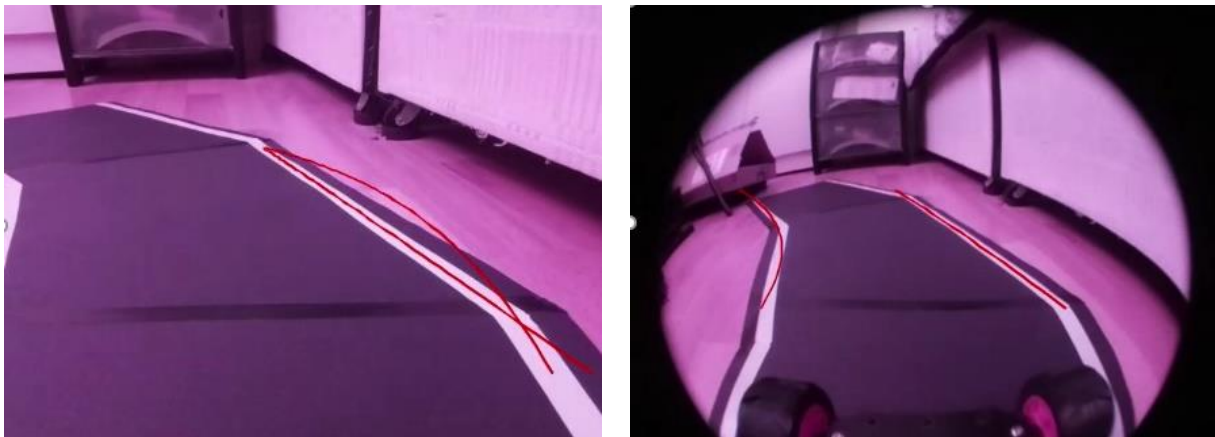


Figure 36 – Comparison Of Standard Lens (Left) Against Fisheye Lens (Right) At The Same Point Along A Corner

The design process for creating the lens mount is featured within appendix section 8.1.1.1.

This, along with the other modifications to both the code and the ego vehicle previously mentioned, ensured that the entire perceptive system had been optimised to its upmost performance. The vehicle followed the longitudinal lane markers (solid and dashed/broken) without hesitation, in both straight lines and around corners. Errors had all but been eliminated from the perceptive portion of the code, other than the occasional blip, showing that the measures taken to prevent instability within the parabolic lane boundaries and steering control had taken a massive effect.

3.2.5. Development Of Stopping Procedures

3.2.5.1. Stopping The Ego Vehicle

After ensuring operation of the infrared sensor and its accuracy assessed (appendix section 8.1.1.3), code could now be written to avoid a collision. The road replicas created for this project were only a single lane, so the best avoidance at this stage would be to bring the ego vehicle to a complete stop. Because of the speed of the electric motor and the weight of the ego vehicle, an accurate distance needed to be ascertained, which once reached, would stop the motor and bring the vehicle to a complete stop, before colliding with an object in the road.

The electric motors' ESC had a voltage cut off of 5V, there was a restriction to the minimum speed achievable before this cut off kicked in. Using a tape measure, an empty box and a stopwatch, an experiment was conducted to find out the time and distance covered while the vehicle is decelerating from the minimum speed, and the time taken to achieve a full stop.

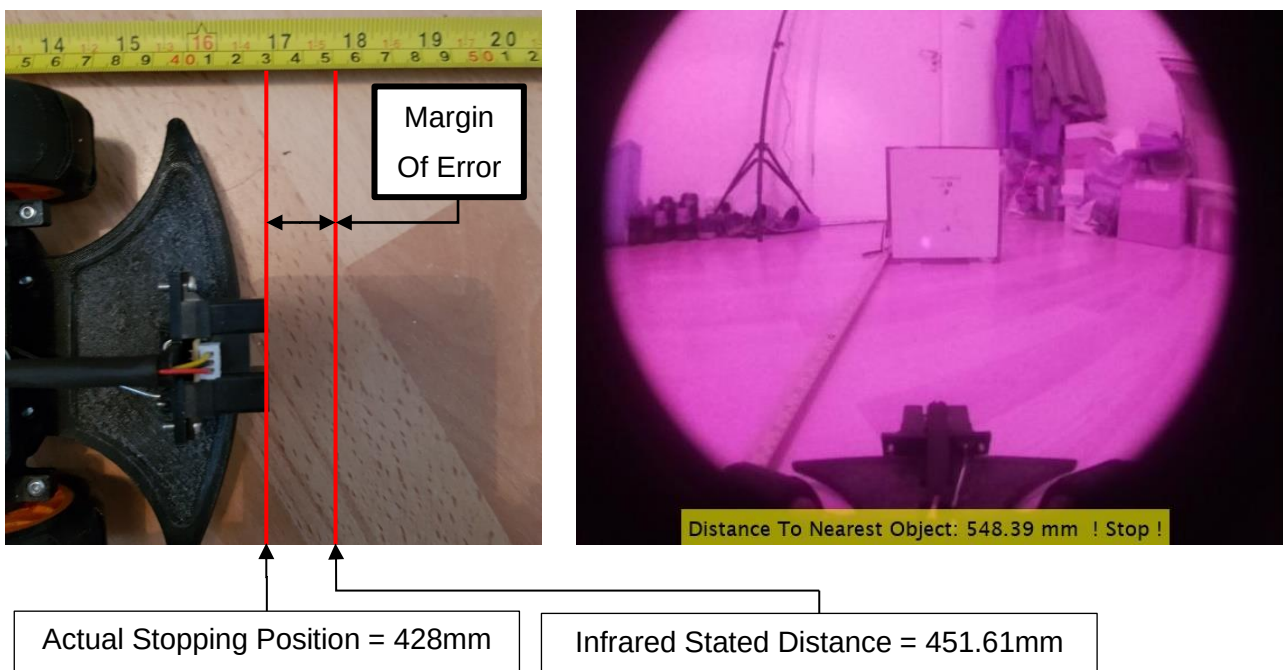


Figure 37 – Vehicle Stopping Test; Measurement Using A Tape Measure (Left), Onboard View Of The Vehicle Annotated With Distance Calculated By The Infrared Sensor (Right)

The distance within the code, telling the vehicle to stop, was set to 1 metre (1000mm), with the empty box aligned with the end of the tape measure. The ego vehicle was set in motion to build up speed and momentum before encountering the box, as it would in the real-world. It then encountered the box and came to a stop in 428mm in 2.75 seconds. Seen in Figure 37.

The 'suvat' equations of motion can be re-arranged to determine the original velocity and the deceleration, displayed within 3-8 and 3-9. It can be assumed that the vehicle was travelling at the minimum velocity allowed before the ESC voltage cut-off and endures a steady linear deceleration due to a low friction wood effect laminate flooring for this test.

$$u \text{ (m / s)} = \frac{2 * s}{t} - v = \frac{2 * 0.428}{2.75} - 0 = 0.3113 \text{ m/s}$$

3-8

$$\alpha \text{ (m/s}^2\text{)} = \frac{v - u}{t} = \frac{0 - 0.3113}{2.75} = - 0.1132 \text{ (m/s}^2\text{)}$$

3-9

Where:

s = Distance travelled while decelerating.

u = Original velocity of the ego vehicle.

v = New velocity of the ego vehicle.

α = Deceleration of the ego vehicle.

t = Time taken to achieve a complete stop.

This experiment was repeated multiple times to gain an average distance covered during the deceleration period. This uncovered a maximum distance of 500mm, which was entered into the code as the distance when the motor would be turned off to stop the vehicle. A test was conducted to guarantee this distance would be effective during real world conditions and repeated 5 times.

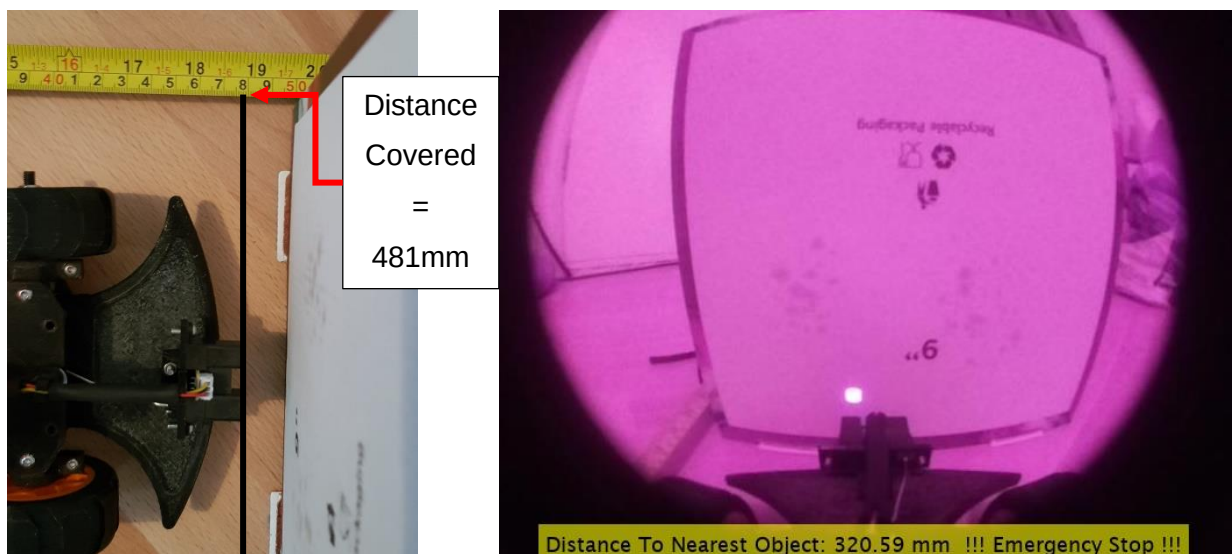


Figure 38 – Closest Approach To Empty Box When Attempting To Stop; Tape Measure (Left), Onboard Image (Right)

This test proved conclusive. The closest the ego vehicle's front bumper came to the empty box without a collision was 19mm (Figure 38), although one time it failed to stop, knocking over the box and exceeded the expected stopping distance by 10mm (Figure 39). This meant that to compensate for any further accidental collisions, an extra 50mm was added into the code, leading to a total distance of 550mm before the ego vehicle is commanded to stop.

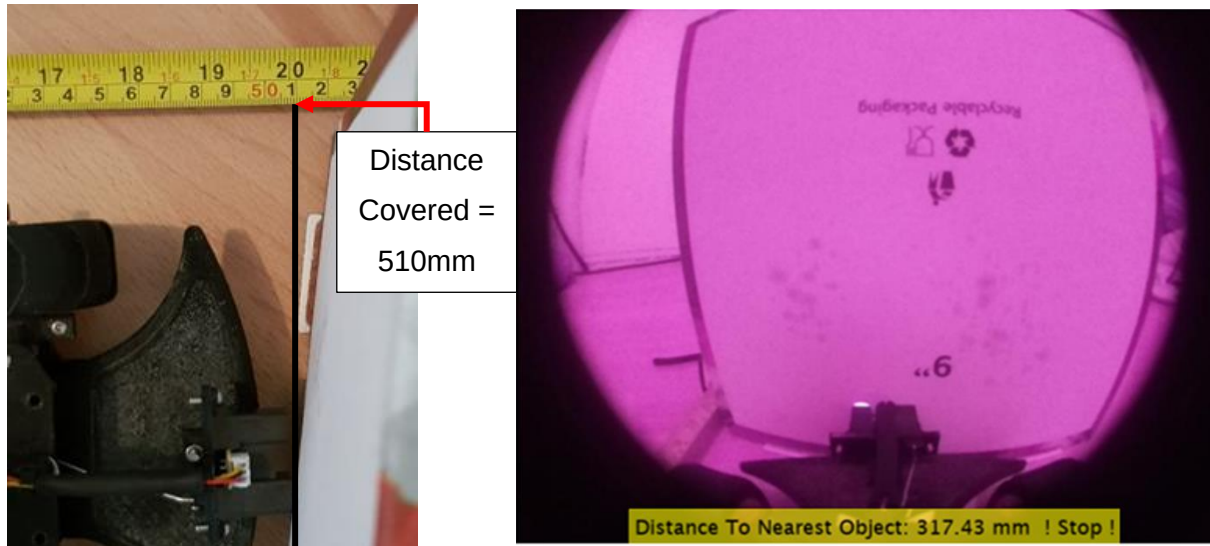


Figure 39 – Collision With Empty Box When Stopping; Tape Measure (Left), Onboard Image (Right)

To prepare for scenarios outside of a controlled stop, such as a pedestrian walking out into the road, or a vehicle in front dabbing their brakes, two more situations were created.

1. If a vehicle was applying their brakes, not to stop, but to gently slow their velocity, the ego vehicle would be required to do the same, keeping a safe distance between them.
2. But if a pedestrian entered the road in front of the ego vehicle, an emergency stop would have to take place to avoid hitting them.

This led to the creation of an 'if' statement in MATLAB, shown within Figure 40, to differentiate between the different required stopping/slowng procedures. Taking the previous 550mm stopping distance into account, if the distance was less than this, the motor would be turned off and the vehicle left to roll to a stop, caused by nothing other than friction.

```
elseif Stop == 1
    if Drive > 90
        Drive = 60;
    end
    writePosition(d,Drive)
    BrakeLog(b,1) = Drive;
    b = b + 1;
    Drive = 90;
    DriveLog(c,1) = Drive;
    writePosition(d,Drive)
    FurtherPotentialDanger = 1;
    Stop = 0;
```

Figure 40 – 'If' Statement To Perform The 'Stop' Procedure

3.2.5.2. Slowing Down The Ego Vehicle

If another vehicle in front of the ego vehicle was slowing down, it would be appropriate for the ego vehicle to do the same. A distance for this to occur would be tested by trial and error, but was first assumed to be 1000mm, this is roughly three times the length of the ego vehicle, which if scaled up to real world vehicle dimensions, is an appropriate distance to keep. This depends on the velocity of the vehicles but can only be judged as safe by a human using the “2 second rule” (Driving Test Tips, 2020).

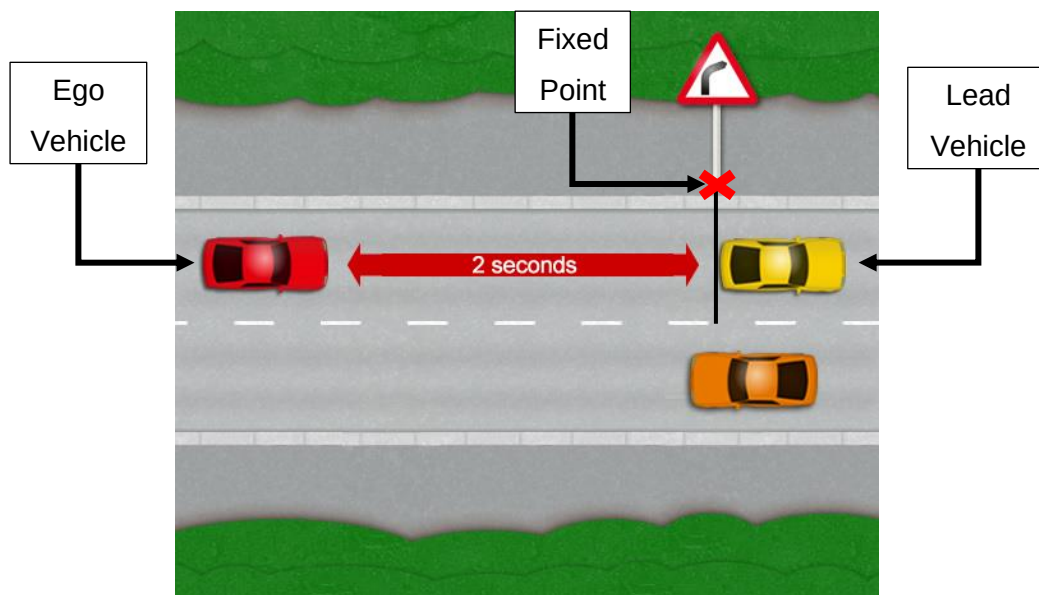


Figure 41 – “Two Second Rule” Demonstration Which Would Be Applied To The Ego Vehicle (Driving Test Tips, 2020)

It has been mentioned before the two environments the author conducted experiments in, this also impacted on the hardware and processing power available at the time. At student accommodation, a desktop computer was able to get the maximum Raspberry Pi camera capability of 30 Frames Per Second (FPS), whereas, at the university campus, a tablet computer only gained an average of 10 FPS. This had a dramatic impact on how quickly the ego vehicle could react, resembling the slower reaction times to that of an older or inebriated driver.

To account for this calamity, the variable named “*Drive*”, which dictated how fast the electric motor span each iteration of the code, would be multiplied/divided by a ‘*factor*’, which would increase/decrease the velocity of the vehicle, at a rate that reflected the computational power and capability of the hardware being used. A separate script was written to test this and provide a value that balanced both a suitable acceleration and deceleration rate, along with the time it would take for each hardware system to process the request and all the frames. For example, the hardware providing 10 FPS would not suit a quick acceleration rate as it may

not be able to react quick enough to any potential situation or danger that may occur during the change in velocity.

The Pi camera was also guilty of 'withholding' frames. The vehicle could be moved from a position after having stopped the code from running, but when the code was then initiated, there would be several frames from the previous position for a few seconds, or an FPS drop would be encountered due to an event which required momentary extensive calculations before then catching up back to live/minimal delay. Without plugging the Pi directly into MATLAB through an ethernet cable and sacrificing the ability to be wireless, this could not be prevented.

Unfortunately, due to the processing rate of MATLAB and the 'withheld' frames from the camera and couriered by Wi-Fi. There was a noticeable lag between when the sensor detected a low distance value, activating the relevant process, to when the camera visibly showed the object when it was detected as a threat, seen in Figure 42.

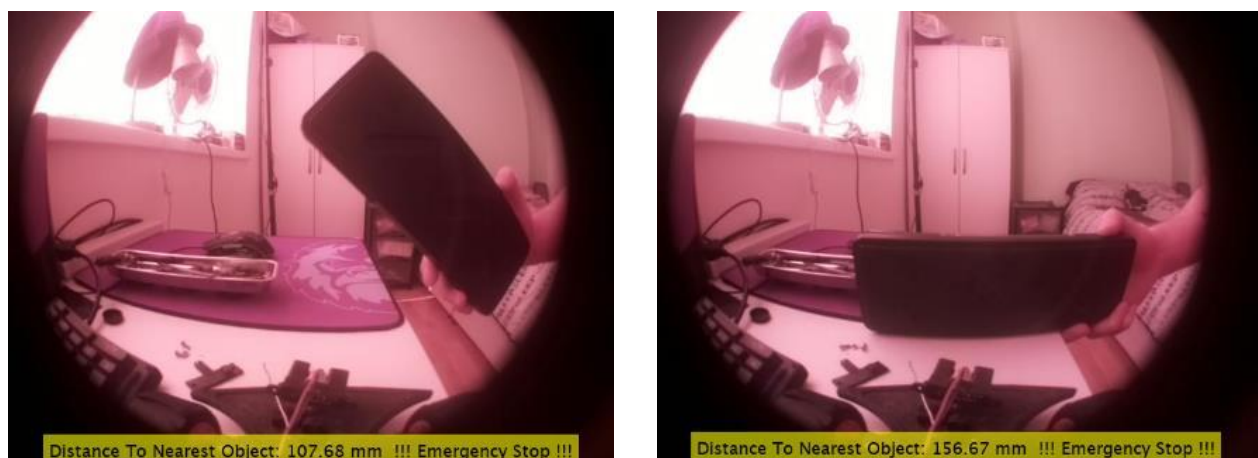


Figure 42 – Lag Experienced While Operating the System; First Frame Sensor Detected Object As A Threat (Left), First Frame Camera 'Caught Up' With The Object (Right)

The same phenomenon occurred when the threat had dissipated, Figure 43.

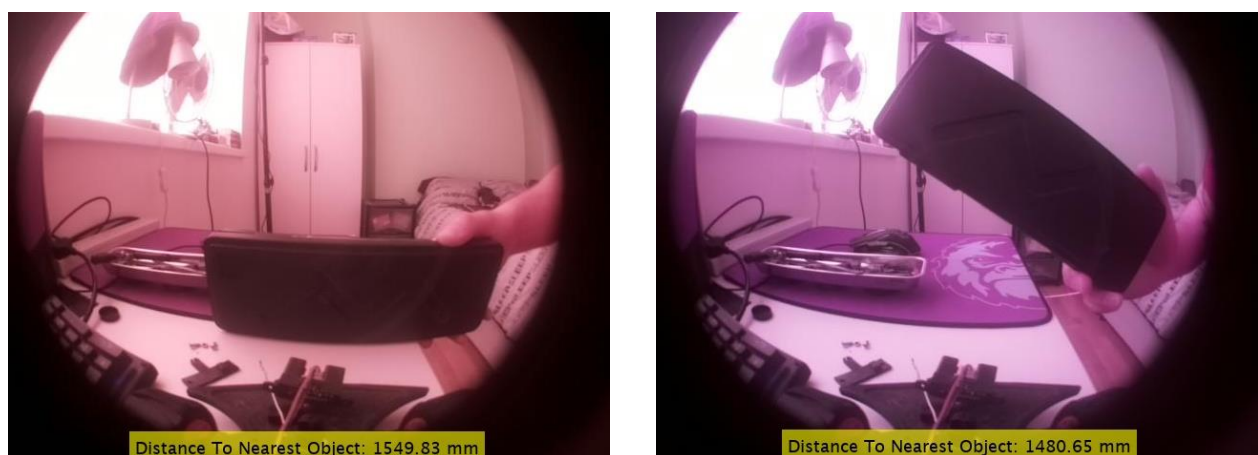


Figure 43 – Lag Experienced While Operating the System; First Frame Sensor Stopped Detecting Object As A Threat (Left), First Frame Camera 'Caught Up' With The Object (Right)

Figure 44 shows the separate code created to determine a balanced rate of change in velocity. The purpose is to calculate how many frames it would take and then estimate the time taken to achieve that velocity at the different framerates to compensate for a slower system.

```

MinSpeed = 108 % Minimum Achievable Torque
MaxSpeed = 130 % Minimum Achievable Torque
% Distinguish Min & Max Speeds For The Motor (Degrees)
UpIterations = 1;
DownIterations = 1;
% Reset Iteration Counters
UpFactor = 1.001
DownFactor = 1.005
% Factor to Increase / Decrease Speed
while MinSpeed < 130
    MinSpeed(UpIterations+1,1) = MinSpeed(UpIterations,1) * UpFactor;
    writePosition(d,Drive)
    UpIterations = UpIterations + 1
end
% Increase Speed Until Max Speed And Apply Command To Motor
while MaxSpeed > 108
    MaxSpeed(DownIterations+1,1) = MaxSpeed(DownIterations,1) / DownFactor
    writePosition(d,Drive)
    DownIterations = DownIterations + 1
end
% Decrease Speed Until Min Speed And Apply Command To Motor
MinSpeed;
MaxSpeed;
% Display Speed Vectors For Each Iteration
UpIterations = UpIterations - 1
DownIterations = DownIterations - 1
% Display Number Of Iterations, Subtracting 1 For The
% Extra One Added During The Final Loop Iteration
TimeUp30FPS = UpIterations/30
TimeDown30FPS = DownIterations/30
% Time Taken To Process Fames At 30 FPS
TimeUp10FPS = UpIterations/10
TimeDown10FPS = DownIterations/10
% Time Taken To Process Fames At 10 FPS

```

Figure 44 – Small Code To Determine An Adequate Acceleration/Deceleration Rate

This led to “Drive” being multiplied by ‘1.001’ to accelerate the vehicle taking 186 iterations to reach the maximum allowable torque (130° in MATLAB) from the minimum achievable (108° in MATLAB) before the voltage cut-off. With a hardware capable of 30 FPS, this would take 6.2 seconds, whereas, at 10FPS this would take 18.6 seconds. This seems viable, if the ego vehicle encountered a reason to slow down or stop 10 seconds after the start of acceleration, it would be more appropriate to not be at the top speed with a lesser capable computational system at the helm.

At the same time, “Drive” would be divided by ‘1.005’ throughout each iteration of the code, to reduce from the maximum speed to the minimum allowable in 38 frames. The rate of deceleration was determined as adequate, taking 1.27 seconds at 30 FPS and 3.8 seconds at 10FPS. This was only to aid in reducing the ego vehicle’s velocity appropriately, not in stopping entirely. When tested, it significantly helped in reducing the momentum of the vehicle, should a collision be detected, making it easier to achieve a complete stop when necessary.

```

if Slow == 1
    if Drive > 108
        Drive = Drive / 1.005;
    end
    if Drive <= 108
        Drive = 108;
    end
    BrakeLog(b,1) = Drive;
    b = b + 1;
    DriveLog(c,1) = Drive;
    writePosition(d,Drive)
    FurtherPotentialDanger = 1;
    Slow = 0;
end

```

Figure 45 – Code To Determine The Value For “Drive” To Operate The Electric Motor

The numbers calculated above for the time taken to process the frames are estimations. Using the same hardware could manifest drastically varied results depending on open background applications, Wi-Fi signal and strength, or the computer just having a bad day. This has been evident throughout the project since the system had been tested in a physical environment.

To show how fast MATLAB was processing the calculations, the current FPS was annotated into the frame to monitor any lag that was experienced. Sadly this wasn’t thought of previously, so it does not appear in all figures. Figure 46 shows the fluctuation in FPS running on the same hardware during operation, but with no change in situation.

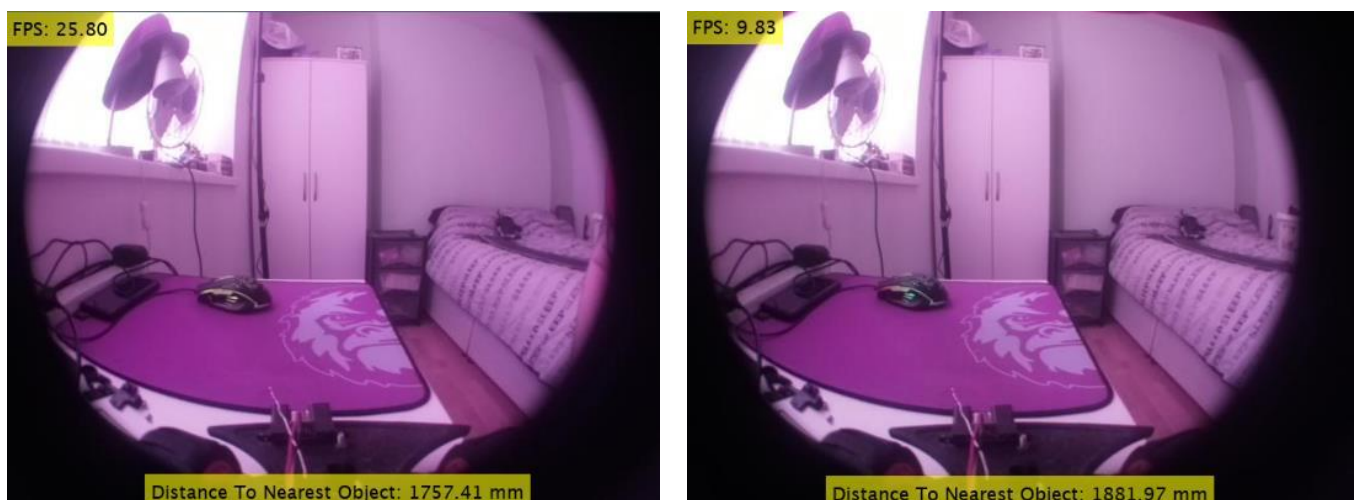


Figure 46 – The Fluctuation Of FPS On The Same Hardware During Operation, High (Left), Low (Right)

3.2.5.3. Performing An Emergency Brake

If an object was less than 400mm away from the vehicle, and the motor was applying forward torque to the vehicle, the motor would then apply a brief but strong amount of reverse torque to aid in the vehicle's deceleration, before turning off the motor entirely. This proved highly efficient at stopping the vehicle to a certain degree, if the object was less than 200mm away before the emergency stop procedure took place, the ego vehicle would carry too much momentum to completely stop before colliding with the object.

This led to more 'if' statements inside the emergency stop procedure, to apply increasing amounts of reverse torque, the closer an object became, then switching the motor off to avoid causing the vehicle to engage in a reverse vector.

This was conceived because it may not always be safe to reverse at that moment in time. If this was a real-world scenario, a third vehicle may have been travelling directly behind, which would also require to undertake an emergency braking procedure as well. Depending entirely on the third driver's reaction times, it could result in the third vehicle being either incredibly close to the ego vehicle's rear bumper or have collided with it.

```
elseif EmergencyStop == 1
    if Distance < 50 && Drive > 90
        Drive = 0;
    elseif Distance < 100 && Drive > 90
        Drive = 10;
    elseif Distance < 150 && Drive > 90
        Drive = 20;
    elseif Distance < 200 && Drive > 90
        Drive = 30;
    elseif Distance < 250 && Drive > 90
        Drive = 40;
    elseif Distance < 300 && Drive > 90
        Drive = 50;
    elseif Distance < 350 && Drive > 90
        Drive = 60;
    elseif Distance < 400 && Drive > 90
        Drive = 70;
    elseif Distance < 400 && Drive == 90
        Drive = 90;
    end
    writePosition(d, Drive)
    BrakeLog(b, 1) = Drive;
    b = b + 1;
    Drive = 90;
    DriveLog(c, 1) = Drive;
    writePosition(d, Drive)
    FurtherPotentialDanger = 1;
    EmergencyStop = 0;
```

Figure 47 – Emergency Stop Procedure With The Code

3.2.5.4. Continuing Along The Path & Acceleration After An Encountered Event

Should there be nothing in the road ahead that would cause an obstruction, or an object which was, but no longer is, a concern, the ego vehicle should be able to continue along its path. In this case, it would be acceptable for the vehicle to accelerate again, using the process described previously by multiplying “Drive” by the factor of ‘1.001’, while being between the minimum and maximum allowable velocities.

If this logical statement was untrue, then “Drive” would be set to the appropriate value. E.g. if “Drive” was less than the minimum value with no obstacle dictating otherwise, it would be set to the minimum value, initiating forward momentum. The next iteration would prove the initial logical statement as true, continuing the vehicle’s acceleration of multiplying “Drive” by ‘1.001’.

```
elseif Continue == 1
    if FurtherPotentialDanger == 1 && OldDrive == 90
        Drive = 90;
    elseif FurtherPotentialDanger == 0 && Drive >= 108 && Drive < 130
        Drive = Drive * 1.001;
    elseif FurtherPotentialDanger == 0 && Drive > 130
        Drive = 130;
    elseif FurtherPotentialDanger == 0 && Drive < 108
        Drive = 108;
    end
    writePosition(d, Drive)
    DriveLog(c, 1) = Drive;
    Continue = 0;
end
```

Figure 48 – Code To Accelerate The Ego Vehicle Should The System Deem It Safe To Proceed

A small addition was created should it now be safe to proceed, but not straight away. Due to the narrow beam of the infrared sensor, it was noticed that if a tennis ball was rolled in front of the ego vehicle during operation, the stopping procedure would only be undertaken during the short time the beam was interrupted. Unfortunately, the amount of time that the tennis ball is deemed a threat by the sensor is significantly less than the time it’s visible by the camera. Figure 49.

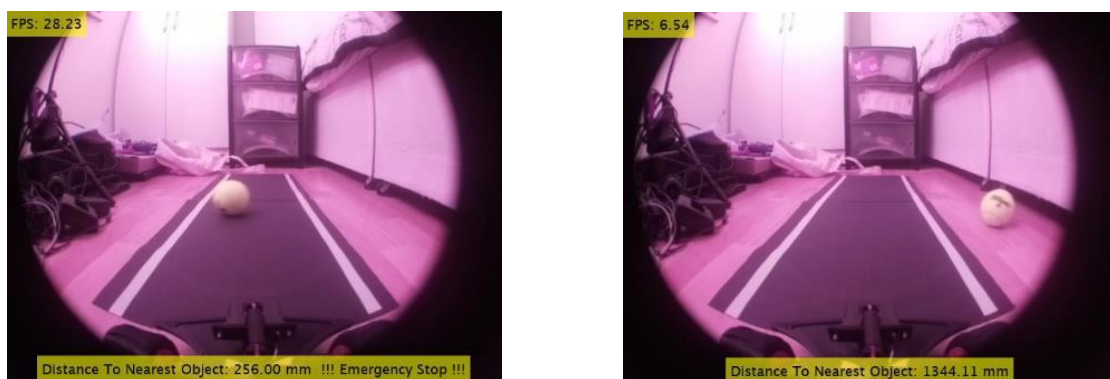


Figure 49 – Tennis Ball Rolling In Front Of The Ego Vehicle, Detected By Sensor And Determined As A Danger (Left), Still In Shot Of Camera And Potential Danger (Right)

This would be an appropriate situation for the ego vehicle to react accordingly. “*FurtherPotentialDanger*” is a variable that is utilised to create a waiting period to a set length of frames, relative to the capability of the hardware running MATLAB. E.g. the vehicle would wait stationary for up to 30 frames (either one second at 30FPS or three seconds at 10FPS) before proceeding to accelerate again. The code within Figure 50 does just that.

```

if FurtherPotentialDanger == 0
    Drive = Drive;
elseif DriveLog(c,1) < 90
    Drive = 90;
    FurtherPotentialDanger = 1;
elseif Distance > 550
    if length(DistanceLog) > 30
        Detections = DistanceLog(c-30:c,1);
    elseif length(DistanceLog) > 20
        Detections = DistanceLog(c-20:c,1) < 550;
    elseif length(DistanceLog) > 10
        Detections = DistanceLog(c-10:c,1) < 550;
    else
        Detections = DistanceLog(c,1) < 550;
    end
    StillDetections = Detections <= 550;
    if any(StillDetections == 1)
        Drive = 90;
        FurtherPotentialDanger = 1;
    elseif all(StillDetections == 0)
        Drive = Drive;
        FurtherPotentialDanger = 0;
    end
end
DriveLog(c,1) = Drive;
writePosition(d,Drive)

```

Figure 50 – Code To Determine Whether There Is Potential Of Further Danger

If “*FurtherPotentialDanger*” equals 0, it can be perceived that there has been no recent obstruction, and the path is clear, so the previously defined value for “*Drive*” is carried through. But should an object be detected, the motor would be set to the ‘off’ position, ensuring the vehicle would not move.

“*DriveLog*” is a variable which records all the forward torque values applied to the electric motor and if the current value is less than ‘90’, braking is being applied. The “*Drive*” variable would be set to ‘90’ to stop the electric motor, or ensures it remains stationary, along with acknowledging the potential for further danger.

If the distance at any point is less than ‘550mm’, the length of the “*DistanceLog*” is then questioned. During initiation of the script, there would not be enough previous frames to judge an adequate waiting period until accelerating again.

When attempts were made to gain the value of the previous frame in the relevant “Log” file, if the counter is ‘1’, then MATLAB would encounter an error as it could not access row ‘0’, stopping the code. This led to the creation of ‘if’ statements based on the length of the “DistanceLog” to decide how far back the assessed value would be selected from, in order to achieve a safe waiting period varying upon system performance.

“Detections” contains all values within “DistanceLog” to a length defined above, that is less than ‘550mm’. “StillDetections” ascertains the logical condition of whether any of the values within “Detections” are true, if so, “Drive” is set to ‘90’ and “FurtherPotentialDanger” is set to ‘1’ to begin or continue the waiting period.

3.2.5.5. Annotated Text Onto Frame To Display Distance & Procedure

Since the Raspberry Pi camera was being fed live and displayed through MATLAB it was customary to supply information consisting of the distance measured by the infrared sensor and the appropriate protocol the ego vehicle was undertaking (slow/stop). 'Continue' was deliberately left out from this to bring more attention to the fact that the ego vehicle has detected, then reacted with the appropriate process to a potential collision. This shows the system has acknowledged that the situation is taking place in the environment and brings the user's eyes directly to them when they appear.

The 'if' statements created previously that depend on the distance to an object in front, for a protocol to be undertaken, dictated the appropriate message to display on the screen. The distance to an object in front is displayed regardless, but the protocol being chosen during that iteration of the script would be displayed alongside it as part of a text string.



Figure 51 – Distances Annotated Onto The Pi Camera Live Feed Before Setting Specific Values For Each Protocol

This was annotated onto the vehicle's camera imagery using the "insertText" function, updating each iteration of the script. The text was situated at the bottom of the frame as it proved to be more eye-catching, but still less intrusive and easier to read as opposed to being at the top. (Figure 51)

Overall, physical testing had proved that the sensor was an effective, efficient and accurate method of controlling the powertrain of the ego vehicle. It had enabled the ability to detect an object, calculate the distance to it, applying appropriate control to the vehicle that best suited the scenario, regardless of how far out of sync the camera was.

4) Discussion

Throughout this project, problems have been encountered and actions taken to prevent or overcome a loss of control of the ego vehicle. These problems are also evident within real-life situations that autonomous vehicles currently being developed for use on public roads experience.

Both the road and the floor surface that the physical road is placed upon within the cameras peripheral vision, influenced the performance of the code. This resulted in a required change to the sensitivity factor within the `'segmentMarkerRidge'` function to compensate for the algorithm misplacement of parabolic lane boundaries. The effects of this has already been discussed, but the problem is not limited purely to this project. Automated Driving Assistance Systems (ADAS) that are currently being developed around the world are encountering the same issue, especially during adverse weather conditions such as rain and snow.

“Rain can reduce the range and accuracy of laser-based Lidar sensors, obscure the vision of on-board cameras and create confusing reflections and glare.” (Solon, 2016)



Figure 52 – 3D Virtual Simulation To Assess The Performance Of An Automated System (Lozé, 2019)

ADAS can be simulated within a virtual environment and scripted scenario and test its performance, seen in Figure 52. “The first tests of such vehicles were performed on physical test tracks, but it soon became clear that it was much more efficient—and much safer—to perform such tests on a virtual car first.” (Lozé, 2019)

A virtual car can be simulated to respond exactly like its real-world counterpart. It can then be fitted with the same sensors, with visual data being fed to them. It allows engineers to try various sensor placements and weather conditions to virtually assess for any adverse flaws.

The quality of the road markings and surface can significantly impede the performance of an automated vehicles' guidance system (Matowicki, et al., 2016). There are many roads within the UK alone that have lane markers that have worn away and become faded over time, these can be hard for a human driver to see, let alone a vehicle fitted with ADAS. This project has been utilised onto a scale model within a controlled environment, but this issue is far more dangerous in a real-world scenario, where the ego vehicle is required to operate within a dynamically changing environment. This could mean damage to public or private property and human life which could easily occur, should a flaw like this not be prevented from taking place.

The placement and orientation of the camera on the ego vehicle contributed to the effectiveness of the lane detection algorithm. When the camera was positioned too low to the ground, the code struggled to gain an effective placement of the parabolic lane boundaries. Even when the camera was raised up 70mm from its original position, the boundaries still proved problematic, therefore, it was also discovered that the cameras' pitch angle became an important consideration. Although the 11.25° pitch mount that was used within the final iteration of this project proved to be the most effective position, an experiment could be undertaken to uncover the best angle of pitch to use for the visual perception. Most autonomous systems currently in production use a static pitch camera, where the camera is in a fixed orientation during operation unless otherwise repositioned.

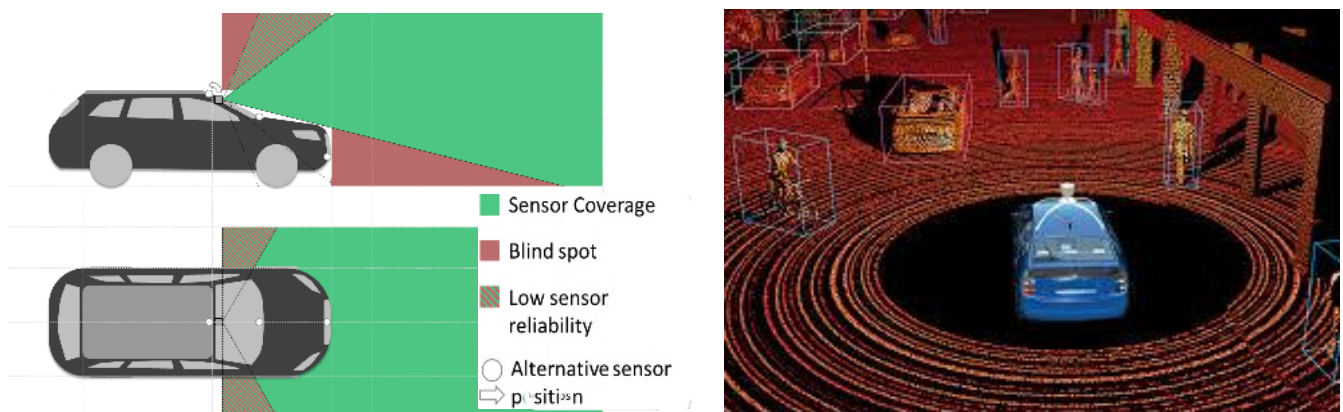


Figure 53 – Sensor Coverage On A Ego Vehicle; Windscreen Mounted Camera Blind Spots (Felix, 2019) (Left) & Shadows Can Be Seen Within A LiDar Sensor Sweep (Scott Drew Pendleton, 2017) (Right)

This yields issues during the Dynamic Driving Task (DDT), where the vehicle is expected to navigate along a path through roads that are not consistent, e.g. varying gradient or winding roads. When this happens the camera is inhibited from achieving its full potential. A patent by Gergely Debreczeni shows the development of a variable pitch camera to maximise image clarity (Debreczeni, 2017). Should this system be implemented in all current and future ADAS vehicles, the performance of the visual perception system could increase by a momentous margin.

This project has highlighted the importance of the cameras' pitch during the period of physical testing. The objective is to provide the visual perceptive system with the clearest FOV possible to ascertain the highest level of accuracy with both lane and object detection to avoid any repercussions.



Figure 54 – Visual Representation Of Tesla Vehicles Sensor Range (Tesla Motors, 2019)

Tesla utilise numerous cameras within its autonomous vehicles, for the front facing cameras they have both standard and wide-angle lens' for various tasks, demonstrated within Figure 54. The wide-angle lens is for close environmental analysis, whereas, the standard lens is used, depending on the focal length, for long and mid-range detections. This displays the fact that no singular lens can be beneficial throughout the DDT, as demonstrated throughout this project. The experiment validated this point, where although the standard lens provided higher quality imagery, it did not display enough of the environment to suitably control the vehicle, as the fisheye lens did.

A significant safety and security risk would be encountered if the autonomous system was hacked by a cyber terrorist group. As discussed previously, it has been easily proved that the system can be hacked through a remote access point within the vehicle for its 'wireless' features that connect to the drivers' phone.

Sensors have been implemented within road vehicles long before the introduction of automation, measuring; air flow, velocity and orientation. Currently, physical access is required to extract this data, but with the levels of connectivity ever increasing, this could soon

change, where the vehicle manufacturer or breakdown/recovery service provider (AA, Greenflag or RAC) can remotely access your vehicle to ascertain the problem at hand.

Unfortunately, should this method be accessible to the wrong person, the danger could be considerable, the attacker could easily change its trajectory and lock the throttle fully open to strike a building or a crowd of pedestrians, with catastrophic consequences.

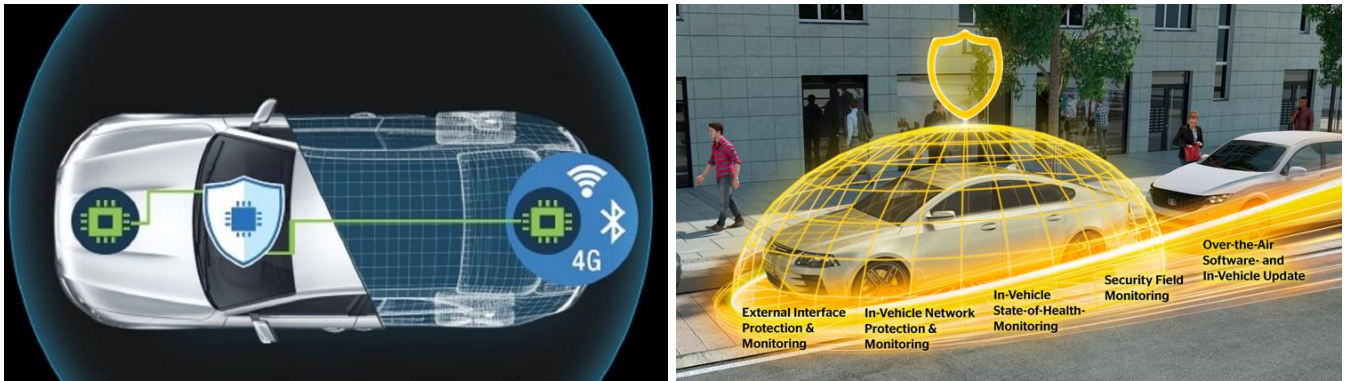


Figure 55 – Cyber security On Automated Vehicles; Vulnerable Systems (Bedigian, 2018) (Left), Threat Security (Continental, 2020) (Right)

“Sensor readings will be validated by a control unit to ensure that it is within tolerance and as expected. However, providing the attacker knows the range of this tolerance, they can ensure that the behaviour of the vehicle is adjusted without causing the Engine Control Unit (ECU) to enter a safe mode.” (Parkinson, et al., 2017)

For the system to be secured from outside intrusion which could put both the lives of the vehicles' occupants and those within the surrounding environment at risk, the wireless infrastructure would need to have a floating or alternating IP address, ensuring that the vehicles IP address changes each times it's switched on, this would make a remote connection harder to obtain. This could also be improved where if any connection to the vehicle was attempted, the driver/occupant of the vehicle would have to confirm they know the device and therefore approve that the connection is valid and legitimate.

The significance played by the object detection functions within the industry, can have momentous consequences if the neural network that classifies objects, cannot accurately determine what an object in its path is.

This have been proven by one of Ubers' self-driving fleet, a Volvo XC90 in Tempe, Arizona, March 18, 2018 (BBC News, 2019). The vehicle which was travelling along an empty road at night when Elaine Herzberg, stepped out to cross the road with her bicycle, both the vehicle and the safety driver failed to react and apply the brakes, which resulted in the first pedestrian to be killed by a self-driving vehicle.

Although the safety driver was watching television on her phone during the time leading up to the incident, the vehicle also failed to categorise Ms Herzberg when it first detected her, 6 seconds before the crash. During this time, the perceptive system became confused, switching her classification between an unknown object, a vehicle and eventually a bicycle. The worse part came when investigators learned that 1.3 seconds before the collision, the accident avoidance system that is standard on the Volvo XC90, realised that emergency braking was required, however, Uber had disabled this system to prevent conflict with the self-driving system they had installed.



Figure 56 – Fatal Crash In Arizona; Dashcam Last Moment Before Impact (Levin, 2018) (Left), News Coverage (Gartenberg, 2018) (Right)

Although police had deemed this incident entirely avoidable, prosecutors had determined that Uber were not criminally liable for Ms Herzberg's death, this has sparked a massive debate as to who exactly is to blame. Had Uber not disabled the Volvo emergency braking system that detected the pedestrian, the fatal collision may have been avoided. The safety driver, who is required to pay attention to the road at all times, did not fulfil this requirement and was estimated to have taken her eyes off the road more than 160 times to watch her phone, during her route. Some blame Ms Herzberg herself for not checking for oncoming vehicles when crossing the road. When watching the dashcam footage of the incident, she is only visible within the last two seconds before the crash, and during that time she did not look directly at the vehicle. (BBC Click, 2019)

This calls into question the standard practice of courtroom justice for road traffic collisions. Who should be penalised if something went wrong? Would the company as a whole be blamed, or an individual programmer who may have accidentally typed a full stop instead of a comma? This would require special laws to be written to protect the innocent from being prosecuted, but still satisfy the public's requirement for justice to be served and prevent a similar situation becoming common place.

5) Conclusion

Overall, this project has managed to achieve every aim that was set out at the start and along the way, has encountered the same limitations of self-driving systems within real-life applications.

It has been made clear that no single physical or virtual setup can endure the harsh reality of the Dynamic Driving Task (DDT). For this project, only one camera was used for the visual perception, whereas in reality, multiple cameras are used to counteract blind spots around the vehicle. Should the 'variable orientation camera' be implemented into every autonomous system, the problems encountered throughout real-world roads, where varying vehicle pitch and yaw angles could inhibit the system from accurate lane control and object detection to avoid potential collisions with the surrounding environment.

There are numerous companies attempting to be the first in achieving level 5 autonomous systems that can understand and interpret the road ahead, so as to never require the driver to intervene under any circumstance. The amount of time developing these systems is extraordinary, but it is required to log several hundred thousand hours of driving in a multitude of environments, for the system to be able learn and act accordingly.

The importance of correctly classifying objects, road signs and traffic lights has been instrumental within the industry, otherwise a similar incident, such as that seen in Arizona, could be repeated. This would reverse the industries goal of improving road safety by removing the human from behind the wheel of a vehicle.

6) Further Work

A method of automating the sensitivity of the black and white binary image that detects the lane markers depending on the environment or lighting would be undertaken. There is potential to achieve this by separating the random blips and noise that appear from the rest of the image. One process was attempted to accomplish this but proved fruitless. The resultant image either had no difference from the original, or it eliminated everything and provided a completely black image.

The same could be attempted to resolve the situation that would occur, should the ego vehicle approach a yellow box junction. These should not be entered unless the road ahead is clear to prevent obstructing any other approaching vehicles. If the current ego vehicle approached these road markings, it would be confused by number of lines that were detected by the perceptive system. This links into the point made previously which could remove the excess lines, keeping the vehicle in the appropriate lane and pointing the correct direction, but also to acknowledge this junction letting the system take the correct action to keep the occupants and surrounding drivers safe.

Another camera and alternative sensors could be implemented into the vehicle, enabling the ability to safely reverse. This opens the extra possibility of reversing away from a dangerous situation, such as a vehicle in front being on fire. It could also allow the vehicle to reverse a set distance, then navigate around an obstruction in the road, should it be safe to do so.

Along the same notion, the front facing camera could utilise a servo motor that turns in relation to the front wheels. This would ensure that the road ahead is constantly within the cameras Field Of View (FOV) guaranteeing the best performance that the coding and vehicle are capable of.

During this project, the ego vehicle was only tested on a single lane road. This means that in its current state, the ego vehicle may approach a two-lane road, either two-way or one-way, accidentally mistaking the outside road markers for the individual lane markers. This could lead to an accident, due to the ego vehicle positioning itself in the centre of the road, rather than the centre of the lane. This would need to be extensively tested to prevent such a disaster.

Further potential to enable this system to run without an external computational source would be an incredible leap in gaining the necessary requirements to create a completely fool-proof system that could tackle all aspects of the Dynamic Driving Task (DDT).

7) References

AGTx, 2019. *Driving an ESC/Brushless-Motor Using Raspberry Pi*. [Online]
Available at: <https://www.instructables.com/id/Driving-an-ESCBushless-Motor-Using-Raspberry-Pi/>
[Accessed 05 2019].

Allen, J., 2019. *AUTONOMOUS TAXIS HAVE MADE THEIR DRIVERLESS DEBUT IN LONDON*. [Online]
Available at: <https://www.driving.co.uk/news/autonomous-taxis-made-driverless-debut-london/>
[Accessed 05 11 2019].

Anderson, M., 2020. *Hardware And Software Parts List Used For This Project*, Swansea: UWTSD.

Anderson, M., 2020. *Table Concerning Connections Of MCP3007 ADC To Raspberry Pi*, Swansea: UWTSD.

Andersson, J., 2016. *Developing Advanced Emergency Braking Systems at Scania*, s.l.: Scania and Mathworks.

Attia, R., Orjuela, R. & Basset, M., 2014. *Combined longitudinal and lateral control for automated vehicle guidance*, Brunstatt: Modélisation Intelligence Processus Systèmes (MIPS).

Audi, 2017. *Audi AI:CON*. [Online]
Available at: <https://www.audi.com/en/experience-audi/models-and-technology/concept-cars/audi-aicon.html>
[Accessed 04 11 2019].

Bakker, B. d., 2019. *How to use a SHARP GP2Y0A710K0F IR Distance Sensor with Arduino*. [Online]
Available at: <https://www.makerguides.com/sharp-gp2y0a710k0f-ir-distance-sensor-arduino-tutorial/>
[Accessed 10 2019].

Banana_Science, 2018. *DIY RC Street Racing Car: One Week Classroom Project*. [Online]
Available at: <https://www.thingiverse.com/thing:2653508>
[Accessed 2019].

BBC Bitesize, 2019. *Sound waves*. [Online]

Available at: <https://www.bbc.co.uk/bitesize/guides/z8d2mp3/revision/3>

[Accessed 11 2019].

BBC Click, 2019. *The Self-Driving Car Capital Of The World*. [Online]

Available at: <https://youtu.be/QC2yOrXfkUs>

[Accessed 29 01 2020].

BBC News, 2019. *Uber in fatal crash had safety flaws say US investigators*. [Online]

Available at: <https://www.bbc.co.uk/news/business-50312340>

[Accessed 29 01 2020].

Bedigian, L., 2018. *Carmakers Must Stack Cyber-Security in Their Favor, Says Karamba*. [Online]

Available at: <https://www.tu-auto.com/carmakers-must-stack-cyber-security-in-their-favor-says-karamba/>

[Accessed 02 2020].

Boroun, A., 2015. *Lane Detection and Tracking Using a Linear Parabolic Model*,

Gazimağusa: Eastern Mediterranean University.

Brownlee, J., 2019. *A Gentle Introduction to Object Recognition With Deep Learning*.

[Online]

Available at: <https://machinelearningmastery.com/object-recognition-with-deep-learning/>

[Accessed 01 2020 2020].

Bruls, T., 2018. *The Right (Angled) Perspective: Improving the Understanding of Road Scenes Using Boosted Inverse Perspective Mapping*. [Online]

Available at: <https://ori.ox.ac.uk/the-right-angled-perspective-improving-the-understanding-of-road-scenes-using-boosted-inverse-perspective-mapping/>

[Accessed 09 2019].

Burnett, R., 2017. *Ultrasonic vs Infrared (IR) Sensors – Which is better?*. [Online]

Available at: <https://www.maxbotix.com/articles/ultrasonic-or-infrared-sensors.htm>

[Accessed 11 2019].

Burnett, R., 2020. *Understanding How Ultrasonic Sensors Work*. [Online]

Available at: <https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm>

[Accessed 03 2020].

CB Insights, 2019. *40+ Corporations Working On Autonomous Vehicles*. [Online]
Available at: <https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/>

[Accessed 04 11 2019].

Continental, 2020. *Cyber Security*. [Online]

Available at: <https://www.continental-automotive.com/en-gl/Cyber-Security>

[Accessed 02 2020].

Corless, M., 2018. *What's new for MATLAB and Simulink for Automated Driving*. s.l., Mathworks.

CRRC, 2017. *First railless train unveiled in CRRC Zhuzhou*. [Online]

Available at: <https://www.crrcgc.cc/en/g7389/s13996/t286142.aspx>

[Accessed 24 11 2019].

Davide Quaglia, R. M., 2016. *Communication-Aware Bandwidth-Optimized Predictive Control of Motor Drives in Electric Vehicles*, s.l.: Institute of Electrical and Electronics Engineers.

Debreczeni, G., 2017. *Autonomous Vehicle With Improved Visual Detection Ability*. United States, Patent No. US 9,535,423 B1.

Department of Transport, 2018. Chapter 5 - Road Markings. In: *Traffic Signs Manual*. s.l.:Crown, pp. 11-13.

Derpanis, K. G., 2010. *Overview of the RANSAC Algorithm*, Toronto: York University.

Driving Test Tips, 2020. *2 Second Rule Explained*. [Online]

Available at: <https://www.drivingtesttips.biz/2-second-rule.html>

[Accessed 01 2020].

Electronics For Everyone, 2017. *Raspberry Pi and Arduino Laptop*. [Online]

Available at: <https://www.instructables.com/id/Raspberry-Pi-and-Arduino-Laptop/>

[Accessed 13 03 2019].

European Automotive Manufacturers Association, 2017. *What Is Truck Platooning*, London: European Automotive Manufacturers Association.

Felix, 2019. *Sensor Set Design Patterns for Autonomous Vehicles*. [Online]

Available at: <https://autonomous-driving.org/2019/01/25/positioning-sensors-for->

autonomous-vehicles/

[Accessed 02 2020].

Firebox, 2020. *Wacky Waving Inflatable Tube Guy*. [Online]

Available at: <https://www.firebox.com/Wacky-Waving-Inflatable-Tube-Guy/p8924>

[Accessed 04 2020].

Fleet Industry News, 2018. *Autonomous cars could make driving less safe, says*

International Transport Forum. [Online]

Available at: <https://www.fleetnews.co.uk/news/fleet-industry-news/2018/05/29/autonomous-cars-could-make-driving-less-safe-says-international-transport-forum>

[Accessed 2019].

Gartenberg, C., 2018. *Safety driver of fatal self-driving Uber crash was reportedly watching Hulu at time of accident*. [Online]

Available at: <https://www.theverge.com/2018/6/22/17492320/safety-driver-self-driving-uber-crash-hulu-police-report>

[Accessed 02 2020].

Guardian, T., 2017. *Semi-automated truck convoys get green light for UK trials*. [Online]

Available at: <https://www.theguardian.com/politics/2017/aug/25/semi-automated-truck-convoy-trials-get-uk-go-ahead-platooning>

[Accessed 04 11 2019].

Hadik, A., Jung, S. K. & Lester, E., n.d. *Enhancing the Performance of the Neural-Network for the Self-Driving Car*, Providence: Brown University.

Herniter, M. E., 2016. *Autonomous Vehicle Control, Modelling, and Design*, s.l.: Rose-Hulman Institute of Technology.

Highfield, V., 2017. *It's worryingly easy to confuse a driverless car with stickers on road signs*. [Online]

Available at: <https://www.alphr.com/cars/1006543/its-worryingly-easy-to-confuse-a-driverless-car-with-stickers-on-road-signs>

[Accessed 04 11 2019].

Hitti, N., 2017. *Trackless, driverless "rail bus" takes to the roads in China*. [Online]

Available at: <https://www.dezeen.com/2017/11/06/worlds-first-driverless-trackless-train-launches-china-zhuzhou-transport-design/>

[Accessed 04 11 2019].

Hui, J., 2018. *Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)*. [Online]

Available at: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[Accessed 03 03 2020].

Jacopo Guanettia, Y. K. F. B., 2018. *Control of Connected and Automated Vehicles: State of the Art and Future Challenges*, Berkeley: Department of Mechanical Engineering, University of California.

Javiya, S. N., 2016. *Design of Model Predictive Control for Position Tracking of DC Servo*, Rajkot: Atmiya Institute of Technology and Science.

Jie Ji, A. K. ., W. W. M. ., Y. H., 2016. *Path Planning and Tracking for Vehicle Collision Avoidance Based on Model Predictive Control With Multiconstraints*, Waterloo: Institute of Electrical and Electronics Engineers.

Jost, D., 2019. *What is an Ultrasonic Sensor?*. [Online]

Available at: <https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor>

[Accessed 11 2019].

Kable, G., 2018. *Audi Aicon concept to enter limited production under pilot project*. [Online]

Available at: <https://www.autocar.co.uk/car-news/motor-shows-frankfurt-motor-show/audi-aicon-concept-enter-limited-production-under-pilot>

[Accessed 04 11 2019].

Kaslikowski, A., 2019. *Everything you need to know about autonomous vehicles*. [Online]

Available at: <https://www.digitaltrends.com/cars/the-current-state-of-autonomous-vehicles/>

[Accessed 04 11 2019].

Kelechava, B. -. A. N. S. I., 2018. *SAE Levels of Driving Automation*. [Online]

Available at: <https://blog.ansi.org/2018/09/sae-levels-driving-automation-j-3016-2018/>

[Accessed 04 11 2019].

Koifman, V., 2017. *Waymo Self-Driving Car Relies on 5 LiDARs and 1 Surround-View Camera*. [Online]

Available at: <http://image-sensors-world.blogspot.com/2017/10/waymo-self-driving-car-relies-on-5.html>

[Accessed 24 11 2019].

Kurtz, N. & Jayaraman, A., 2018. *Automated LiDAR Point-Cloud Annotation for Sensor Verification*, s.l.: Autoliv and Mathworks.

L&T Technology Services, 2016. *Autonomous Drive*. s.l., s.n.

Langridge, M., 2019. *Self-driving cars: Autonomous driving levels explained*. [Online]
Available at: <https://www.pocket-lint.com/cars/news/143955-sae-autonomous-driving-levels-explained>

[Accessed 04 11 2019].

Lazarra, P., 2015. *Fiat Chrysler recalls 1.4m Jeeps affected by hack attack*. [Online]
Available at: <https://www.alphr.com/cars/1001240/fiat-chrysler-recalls-14m-jeeps-affected-by-hack-attack>

[Accessed 04 11 2019].

Levin, S., 2018. *Video released of Uber self-driving crash that killed woman in Arizona*.

[Online]

Available at: <https://www.theguardian.com/technology/2018/mar/22/video-released-of-uber-self-driving-crash-that-killed-woman-in-arizona>

[Accessed 02 2020].

Lexico, 2020. *ground-truth*. [Online]

Available at: <https://www.lexico.com/definition/ground-truth>

[Accessed 03 03 2020].

Lozé, S., 2019. *Making autonomous vehicles safer before they hit the road*. [Online]

Available at: <https://www.unrealengine.com/en-US/blog/making-autonomous-vehicles-safer-before-they-hit-the-road>

[Accessed 02 2020].

Masinde Muliro University of Science and Technology, 2019. *INTRODUCTION TO RADAR SYSTEMS*. [Online]

Available at:

[http://mmust.elimu.net/BSC\(ELEC COMM\)/Year 4/ECE%20451%20L Radar Eng and Facsimile/Introduction to Radar/Introduction to Radar.htm](http://mmust.elimu.net/BSC(ELEC COMM)/Year 4/ECE%20451%20L Radar Eng and Facsimile/Introduction to Radar/Introduction to Radar.htm)

[Accessed 11 2019].

Mathworks, 2017. *Automated Driving Toolbox*. [Online]

Available at: <https://uk.mathworks.com/products/automated-driving.html>

[Accessed 12 03 2019].

Mathworks, 2019. *Automated Driving with MATLAB*. [Online]
Available at: <https://www.mathworks.com/training-schedule/automated-driving-with-matlab.html>

[Accessed 07 2019].

Mathworks, 2019. *birdsEyeView*. [Online]
Available at: <https://www.mathworks.com/help/driving/ref/birdseyeview.html>

[Accessed 07 2019].

Mathworks, 2019. *cameraIntrinsics*. [Online]
Available at: <https://www.mathworks.com/help/vision/ref/cameraintrinsics.html>

[Accessed 06 2019].

Mathworks, 2019. *computeBoundaryModel*. [Online]
Available at:
<https://www.mathworks.com/help/driving/ref/paraboliclaneboundary.computeboundarymodel.html>

[Accessed 11 2019].

Mathworks, 2019. *Deep Learning with MATLAB*. [Online]
Available at: <https://uk.mathworks.com/training-schedule/deep-learning-with-matlab>

[Accessed 05 2019].

Mathworks, 2019. *findParabolicLaneBoundaries*. [Online]
Available at: <https://uk.mathworks.com/help/driving/ref/findparaboliclaneboundaries.html>

[Accessed 07 2019].

Mathworks, 2019. *lateralControllerStanley*. [Online]
Available at: <https://www.mathworks.com/help/driving/ref/lateralcontrollerstanleyfunction.html>

[Accessed 11 2019].

Mathworks, 2019. *rgb2gray*. [Online]
Available at: <https://www.mathworks.com/help/matlab/ref/rgb2gray.html>

[Accessed 07 2019].

Mathworks, 2020. *Analog Input Using SPI*. [Online]
Available at: <https://uk.mathworks.com/help/supportpkg/raspberrypiio/ref/analog-input-using-spi.html>

[Accessed 2 2020].

Mathworks, 2020. *monoCamera*. [Online]

Available at: <https://www.mathworks.com/help/driving/ref/monocamera.html>

[Accessed 07 2019].

Mathworks, 2020. *trainingOptions*. [Online]

Available at: <https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html>

[Accessed 10 02 2020].

Matowicki, M., Pribyl, O. & Pribyl, P., 2016. *Analysis of possibility to utilize road marking for the needs of autonomous vehicles*, Prague: Czech Technical University.

Merriam, R., 2016. *HOW TO USE LIDAR WITH THE RASPBERRY PI*. [Online]

Available at: <https://hackaday.com/2016/01/22/how-to-use-lidar-with-the-raspberry-pi/>

[Accessed 11 2019].

Mianzo, L., 2017. *Big Data, Data Analytics, and Machine/Deep Learning Infrastructure at Caterpillar*. s.l., Caterpillar.

Microchip, 2008. *2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface*. [Online]

Available at: <https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf>

[Accessed 11 2019].

Mond, A., 2018. *Developing Longitudinal Controls for a Self-Driving Taxi*, s.l.: Voyage Auto and Mathworks.

Multicopter, 2019. *lipo battery calculator*. [Online]

Available at: http://multicopter.forestblue.nl/lipo_need_calculator.html

[Accessed 20 04 2019].

Nikon, 2020. *DSLR Camera Basics*. [Online]

Available at: <https://imaging.nikon.com/lineup/dslr/basics/19/01.htm>

[Accessed 07 2019].

Niu, M., 2018. *Object Detection and Tracking for Autonomous Driving by MATLAB toolbox*, Columbus: The Ohio State University.

Odne Stokke Burheim, 2018. Chapter 7 - Secondary Batteries. In: *Engineering Energy Storage*. Trondheim: Academic Press, pp. 111-145.

Parkinson, S., Ward, P., Wilson, K. & Miller, J., 2017. *Cyber Threats Facing Autonomous and Connected Vehicles: Future Challenges*, s.l.: IEEE Transactions on Intelligent Transportation Systems.

Patrascu, D., 2019. *How the Mercedes Automated Valet Parking Works*. [Online]
Available at: <https://www.autoevolution.com/news/how-the-mercedes-automated-valet-parking-works-128807.html>
[Accessed 04 11 2019].

PCI Geomatics, 2019. *Defining the principal point*. [Online]
Available at: https://www.pcigeomatics.com/geomatica-help/concepts/orthoengine_c/Chapter_45.html
[Accessed 07 2019].

Pinout!, 2019. *The comprehensive GPIO Pinout guide for the Raspberry Pi.* [Online]
Available at: <https://pinout.xyz/>
[Accessed 04 2019].

Raspberry Pi Foundation, 2019. *Camera Module*. [Online]
Available at: <https://www.raspberrypi.org/documentation/hardware/camera/>
[Accessed 06 2019].

Raspberry Pi Tutorials, 2018. *Infrared Distance Measurement with the Raspberry Pi*. [Online]
Available at: <https://tutorials-raspberrypi.com/infrared-distance-measurement-with-the-raspberry-pi-sharp-gp2y0a02yk0f/>
[Accessed 10 2019].

Raspberry Pi tutorials, 2019. *Using a Raspberry Pi Ultrasonic distance sensor*. [Online]
Available at: <https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/>
[Accessed 11 2019].

Raul A. Barreto, 2019. *How we feel about our cars means the road to a driverless future may not be smooth*, Adelaide: The Conversation.

Ravi, 2018. *Interfacing IR Sensor with Raspberry Pi*. [Online]
Available at: <https://www.electronicshub.org/interfacing-ir-sensor-with-raspberry-pi/>
[Accessed 11 2019].

Robert Bosch GmbH, 2019. *Automated Valet Parking*. [Online]
Available at: <https://www.bosch.com/stories/automated-valet-parking/>
[Accessed 04 11 2019].

Samsung SDI Co., Ltd. - Energy Business Division, 2014. *Introduction of INR18650-30Q*, s.l.: Samsung SDI Co., Ltd..

Samsung SDI Co., Ltd. - Energy Business Division, 2015. *Lithium-ion rechargeable cell for power tools*, s.l.: Samsung SDI Co., Ltd..

Sasena, M. & Abhinav, A. S., 2018. *Building Real-Time Driver-in-the-Loop Simulators*, s.l.: Mathworks.

Schenk, P., 2015. *Real-Time Control Prototyping of Driver Assistance and Autonomous Driving Technologies at Mobileye*, s.l.: Speedgoat GmbH and Mathworks.

Schweber, B., 2019. *LIDAR and Time of Flight, Part 2: Operation*. [Online]
Available at: <https://www.microcontrollertips.com/lidar-and-time-of-flight-part-2-operation/>
[Accessed 2 2020].

Scott Drew Pendleton, H. A. X. D. X. S. M. M. Y. H. E. D. R. M. H. A. J., 2017. *Perception, Planning, Control, and Coordination for Autonomous Vehicles*, Singapore: Department of Mechanical Engineering, National University of Singapore.

Scott, B., 2016. *The Impact of Connectivity/Autonomous Driving on Electric Vehicles*. s.l., IHS Markit.

Second Life Storage, 2019. *Cell Database Information*. [Online]
Available at: <https://secondlifestorage.com/celldatabase.php>
[Accessed 12 07 2019].

Second Life Storage, 2019. *Samsung INR18650-30Q Cell Specifications*. [Online]
Available at: <https://secondlifestorage.com/showthread.php?tid=1854>
[Accessed 12 07 2019].

Seed Studio, 2020. *Types of Distance Sensor and how to select one?*. [Online]
Available at: <https://www.seedstudio.com/blog/2019/12/23/distance-sensors-types-and-selection-guide/>
[Accessed 2 2020].

Sharp, 2006. *GP2Y0A02YK0F*, s.l.: SHARP Corporation.

Snider, J. M., 2009. *Automatic Steering Methods for Autonomous Automobile Path Tracking*, Pittsburgh: Carnegie Mellon University.

Solon, O., 2016. *Why self-driving cars aren't safe yet: rain, roadworks and other obstacles*. [Online]

Available at: <https://www.theguardian.com/technology/2016/jul/05/tesla-crash-self-driving-car-software-flaws>

[Accessed 27 02 2020].

Tabora, V., 2018. *Tesla Enhanced Autopilot Overview — L2 Self Driving HW2*. [Online]

Available at: <https://medium.com/self-driving-cars/tesla-enhanced-autopilot-overview-l2-self-driving-hw2-54f09fed11f1>

[Accessed 24 11 2019].

TAKAHAMA, T. & AKASAKA, D., 2017. *Model Predictive Control Approach to Design Practical Adaptive Cruise Control for Traffic Jam*, Kanagawa, Tokyo: SAE Japan.

Tesla Motors, 2019. *Autopilot*. [Online]

Available at: https://www.tesla.com/en_GB/autopilot

[Accessed 09 10 2019].

Thomas Net, 2019. *How Radar Detectors Work*. [Online]

Available at: <https://www.thomasnet.com/articles/instruments-controls/how-radar-detectors-work/>

[Accessed 11 2019].

Thompson, A., 2017. *China Built a Self-Driving... Something*. [Online]

Available at: <https://www.popularmechanics.com/technology/infrastructure/a26782/china-built-a-self-driving-something/>

[Accessed 24 11 2019].

Tomar, V. & Sengar, S., 2016. *Arduino-MATLAB based Semi-Autonomous Car*, Meerut: Meerut Institute of Engineering & Technology.

Transport Xtra, 2019. *Daimler and Bosch's driverless parking to operate without human supervision*. [Online]

Available at: <https://www.transportxtra.com/publications/parking-review/news/61778/daimler-and-bosch-s-driverless-parking-to-operate-without-human-supervision/>

[Accessed 04 11 2019].

Vikrant Tomar, S. S., 2016. *Arduino-MATLAB based Semi-Autonomous*, Meerut:

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering.

Vm, S., 2019. *3D-Object Detection for Autonomous Vehicles*. [Online]
Available at: <https://towardsdatascience.com/3d-object-detection-for-autonomous-vehicles-b5f480e40856>

[Accessed 01 2020].

Waldron, S., 2016. *Cybersecurity Considerations in Today's Connected and Autonomous Vehicles*. s.l., Vector GB Limited.

Walker, J., 2019. *The Self-Driving Car Timeline - Predictions From The Top 11 Global Automakers*. [Online]

Available at: <https://emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/>

[Accessed 04 11 2019].

Wasef, B., 2018. *2019 Audi A8 L Review | Brilliant engineering in an unassuming wrapper*.

[Online]

Available at: https://www.autoblog.com/2018/10/16/2019-audi-a8-l-review-first-drive/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlMmNvbS8&guce_referrer_sig=AQAAANp5ZMM43O-7OuBRsZ_vHv1s6jDfvjcYsLh4XKfWq4LQweYV7TQHuDqr_XQ1A9H4jc8zlaABQ4to_3nXSWMIXV7b14kUbUNHcnfT3es

[Accessed 04 11 2019].

Waymo LLC, 2019. *Technology*. [Online]

Available at: <https://waymo.com/tech/>

[Accessed 03 11 2019].

Wendling, B., 2017. *Taxonomy And Definitions For Terms Related To Driving Automation Systems For On-Road Motor Vehicles*. s.l., SAE.

Wilson, D., 2019. *Average Car Length – List of Car Lengths in Details*. [Online]

Available at: <https://anewwayforward.org/average-car-length/>

[Accessed 01 2020].

Wired, 2015. *Hackers Remotely Kill a Jeep on a Highway - Youtube*. [Online]

Available at: <https://www.youtube.com/watch?v=MK0SrxBC1xs>

[Accessed 04 11 2019].

Woollaston, V., 2018. *Driverless cars of the future: How far away are we from autonomous cars?*, s.l.: Alphr.

World Health Organization, 2018. *Road traffic injuries*. [Online]
Available at: <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>
[Accessed 24 11 2019].

www.linuxcircle.com, 2017. *Problem with ESC, brushless motor and Raspberry Pi*. [Online]
Available at: <https://raspberrypi.stackexchange.com/questions/40636/problem-with-esc-brushless-motor-and-raspberry-pi>
[Accessed 06 2019].

Yakowicz, W., 2017. *The World's First Autonomous Valet Parking Garage is Here*. [Online]
Available at: <https://www.inc.com/will-yakowicz/daimler-bosch-mercedes-autonomous-valet-parking.html>
[Accessed 24 11 2019].

Yang, Z. et al., n.d. *Self-driving car: a neural network practice project using MATLAB*.
[Online]
Available at: <https://www.congying-zhang.space/#!/new-page-4>
[Accessed 10 10 2019].

8) Appendix

8.1. Appendix A – Code & Vehicle Construction

This section details the extra work undertaken for this project but unfortunately did not bring the desired effect to the project. The intent of these aspects was to detect an object in front of the ego vehicle and to judge the distance between them, if the object came too close to the vehicle, the code would stop the driving motor bringing the vehicle to a standstill before a collision was caused.

8.1.1. Ego Vehicle Construction

8.1.1.1. Vehicle Structure

A base model for the vehicle was sourced from Thingiverse, courtesy of Banana Sciences' DIY RC Street Racing Car: One Week Classroom Project. (Banana_Science, 2018) This included '.stl' files for most of the assembly that could be 3D printed, with tutorials on creating your own version of selected parts, these include; the side panels, bumpers, wheels, tyres, top and bottom panels. A parts list featuring all the electrical components and fasteners that would be required for its' construction, featured within section – 8.3.

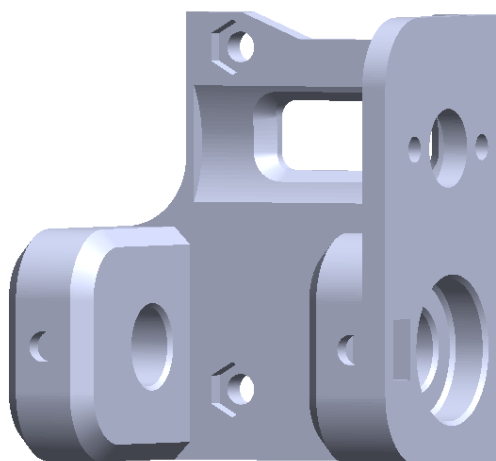


Figure 57 – Imported '.stl' File As A Graphics Object

Since the '.stl' files provided featured no dimensions, they were imported into SolidWorks as a graphics object (Figure 57) then recreated by replicating the shapes and dimensions using the graphics object as a reference point to gain the correct dimensions. (Figure 58) Unfortunately, this process had one drawback, the parts were scaled up to metres when they should be in millimetres, e.g. if a part had a dimension of 26mm it actually came out at 26m. This meant that the function 'Scale' was required, with this, a uniform scale of '0.001' ensured that every part was the correct size.

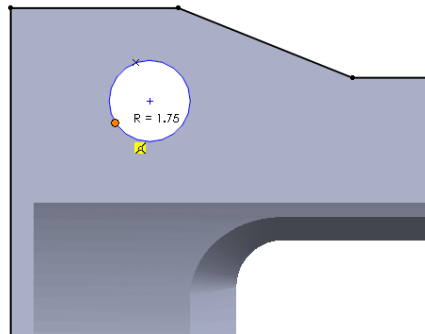


Figure 58 – Using The Graphics Object As A Reference Point To Recreate The Model

Once all the components were recreated, the tutorials were followed to create the remaining parts for the vehicle with the authors custom designs used for the wheels and tyres. The basic vehicle assembly had now been completed and available space within the assembly could be analysed to determine how it could be utilised for use within this project.

With the modified CAD of the ego vehicle, the parts could now be 3D printed using the authors' Creality Ender 3 printer, and then assembled. Polylactic Acid (PLA) filament was used for the rigid components, while Thermoplastic Polyurethane (TPU) filament created the fisheye lens mount to allow for tight, yet malleable grip, as well as the tyres, enabling flexibility and allowing them to react much alike real tyres, deforming the contact patch to increase the grip applied to the ground during operation. (Figure 59)

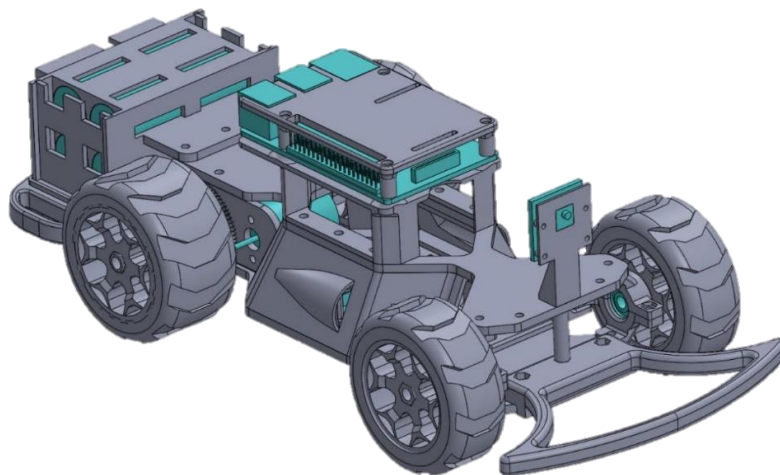


Figure 59 – Completed CAD Model Of The Ego Vehicle (Left),

By using a Vernier gauge to measure the Raspberry Pi board, the Pi Cam Noir and the 18650 lithium-ion cells, the dimensions required to create mountings for each of the components were ascertained. (Figure 60) The mounts were designed in SolidWorks using the basic vehicle assembly as a reference point for determining where these components would be situated on the vehicle, with minimal modifications made to the assembly to simplify the process.



Figure 60 – Ascertaining Measurements For The Raspberry Pi Board

The camera mount was positioned at the front of the vehicle for an optimal field of view (FOV), this was adjusted at a later date, discussed further on within this report. (Figure 61)



Figure 61 – Displaying The Camera Mount At The Front Of The Vehicle

The mount for the Raspberry Pi was placed in the centre of the vehicle due to the width of the board, so as not to come in contact with or inhibit the operation of the wheels, it was raised up and shaped accordingly to encourage air flow to both the Pi board and the components inside the vehicle for a cooling effect and to allow wires to be fed to the General Purpose In/Out (GPIO) pins that send the corresponding signals to the steering servo and driving motor.

The housing for the 18650 lithium-ion cells took several iterations to gain the correct operation that the author required, enabling the creation of rechargeable battery pack to power the Raspberry Pi and implement the ability for the vehicle to be mobile and not be tethered to a plug socket with easy access to the cells for removal/replacement should the need arise.

This housing encompasses a mount for the power bank board to attach to the rear support that bolts to the top plate of the vehicle. Slots cut into the housing to expose the batteries to the ambient air to ensure that the cells had adequate cooling. As well as side panels that slot in and out of the housing which allows connection between the cells and the leaf springs, which were soldered together and wired up to the power bank board, completing the circuit and creating the battery pack. (Figure 62)

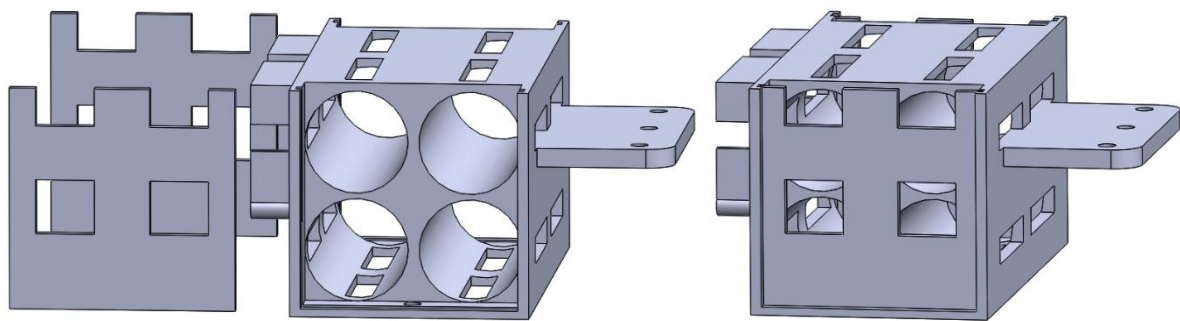


Figure 62 – Battery Case Assembly; Sides Separate (Left) & Sides Inserted (Right)

Because of the weight of the battery pack, if the vehicle encountered a bump, the battery pack would bounce up and down. To resolve this issue, the rear bumper was modified so that the battery pack sat flat on top, with a surrounding lip to discourage horizontal displacement and holes that enabled the bumper and the battery pack to be bolted together to stop any unwanted movement, (Figure 63).



Figure 63 – Different Iterations Of The Rear Bumper; Original (Left) & Modified (Right)

The fisheye lens came provided with a small rubber holder that reaches over to support the lens from the other side (Figure 64 – Top Left). This proved adequate for static testing, but once the vehicle was in motion using the electric motor, the camera lens shaking causing it to drop down and reduce visibility. (Figure 64 – Bottom)



Figure 64 - Fisheye Lens; Attached (Top Left), Motor's R/C Battery Pack (Top Right), Camera Lens Vibration (Bottom)

Two solutions were found. Cable ties were used to hold the battery, firm enough to stop the vibration, but loose enough to allow removal, shown in Figure 65. The battery wire was also cable tied back to avoid contact with the rear wheel.

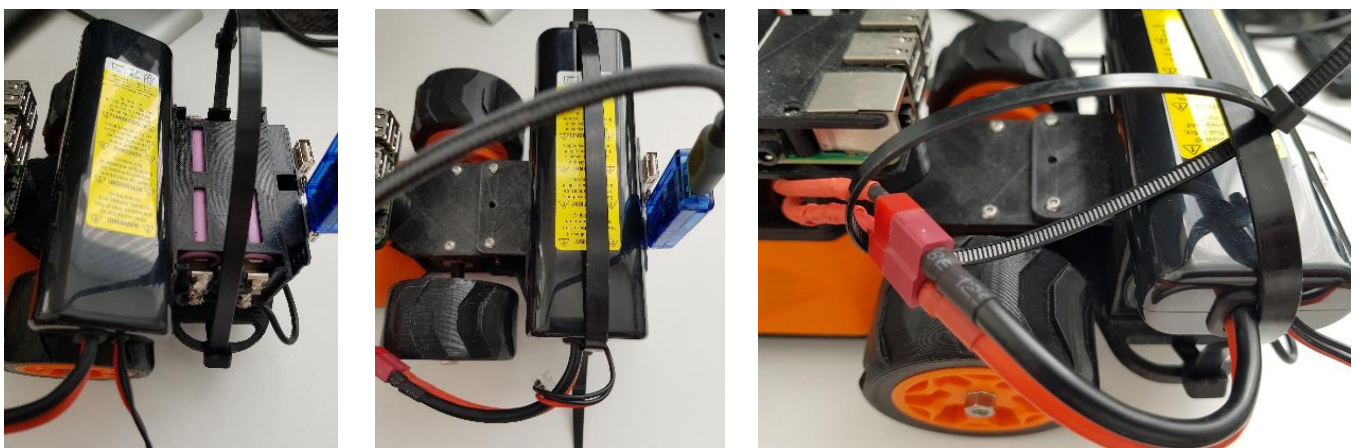


Figure 65 – Cable Ties Use To Secure The Battery In Place, Attached (Left), Removed (Middle), Battery Wire (Right)

While a new mount was designed for the lens, shown in Figure 66 – Left. This would utilise the screws that already hold the Pi camera in place on the pitched mount, and once printed using TPU, would allow flexibility to stretch over the ridges utilised by the original mount but be strong enough to hold it still. Figure 66 – Right shows the fisheye lens being held securely in place. The Pi camera view, once the mount had been placed into position, can be seen in Figure 67 – Left.

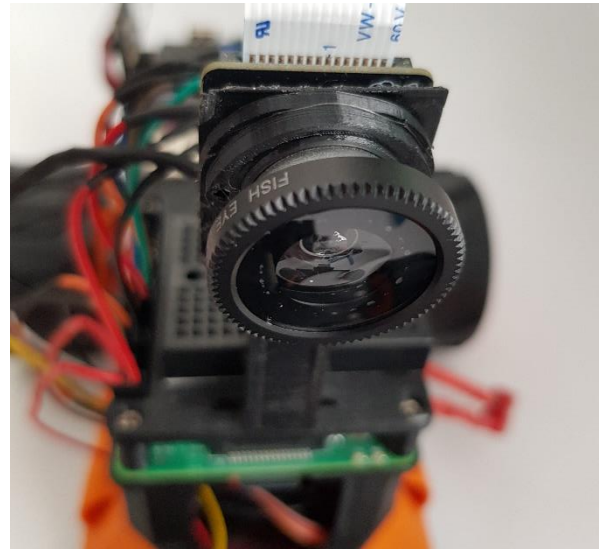
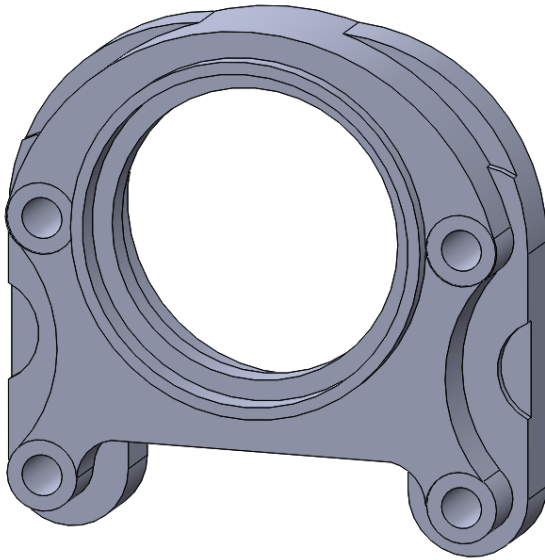


Figure 66 – Mount Created To Securely Hold The Fisheye Lens, CAD model (Left), Assembled (Right)

These new modifications could now be 3D printed, if not already, and assembled onto the ego vehicle to ensure compatibility with existing components and no restrictions to operations that had not been seen otherwise within the Solidworks assembly. The final version of the ego vehicle CAD model is shown below in Figure 67 – Right, where the *grey* components have all been 3D printed and those in *turquoise* have been acquired for this project.

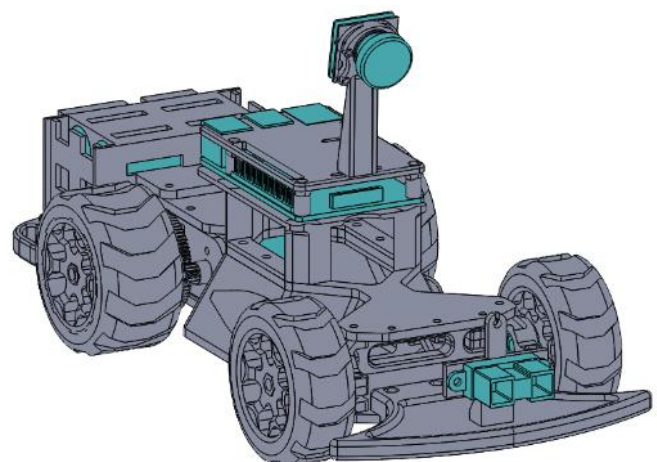


Figure 67 – View From The Fisheye Lens With New TPU Mount (Left), Updated CAD Model Of The Ego Vehicle After All Modifications (Right)

During physical testing, it became apparent that the two battery packs on the rear of the vehicle, began lifting the front of the ground, causing no/understeer. This was a problem. Small weights were sourced and eight '50g' weights were provided courtesy of Mark Fall and Paul Davies. These were set into a position over the front wheels to provide enough downforce to enable the vehicle to steer, without sacrificing handling.

The vehicle was set on the hard flooring to visualise the problem at hand. The front wheels were turned from one extreme to the other, while pushing the vehicle forwards. With no extra weight, the vehicle understeered and at points only one of the front wheels had contact with the ground, creating no directional steering.

When four weights (200g) were added, two over each wheel, bringing the total vehicle weight up from 1.08kg to 1.28kg. The vehicle turned when necessary to the required amount without much extra strain on the motor. When all eight (400g) were placed over the wheels, it provided the same effect as with '200g', but once the electric motor had been connected, there was too much weight, causing the motor to stall unless a gentle push by hand persuaded it to move forwards.



Figure 68 – Weights Used To Apply Downforce To The front Wheels Of The Ego Vehicle, In place (Left), Attached Using Insulation Tape (Right)

Only four weights were applied to the vehicle for the time being, where the remaining could be added at a later date, should they be required. They were positioned as best to apply the added weight to the problem area and assessed to whether they obstructed the view of the camera or blocked the front wheels from turning, Figure 68. They did not. The black insulation tape used to create the physical roads, secured the weights onto the car to prevent movement.

This all but eliminated the understeer issue caused by the longitudinal front-to-rear weight split.

8.1.1.2. Electrical Components

With the vehicle assembly completed, construction of the electrical components could now be undertaken, this would comprise of the connection of the Pi camera, the steering servo motor and the 2430 brushless motors' Electronic Speed Controller (ESC) to the Raspberry Pi, and the creation of the battery pack.



Figure 69 – Connections To Servo Motors; Steering Servo (Left) & Driving Motor (Right)

The easiest tasks were undertaken first. The Pi camera features a ribbon cable that plugs directly into the Raspberry Pi board. The steering servo motor has three output wires (Figure 69 – Left); a red Voltage Common Collector (VCC) the power input for the device, an orange Pulse Width Modulation (PWM) which interprets signals sent to the servo, and a brown ground (GND) wire which completes the circuit. These wires were plugged into the corresponding GPIO pin on the Raspberry Pi (Figure 70 – Left). The red wire was connected to a 5V pin, the brown wire attached to a ground pin, while the orange wire was linked to one of the pins capable of sending PWM signals, in this case, physical pin 12. (Pinout!, 2019)

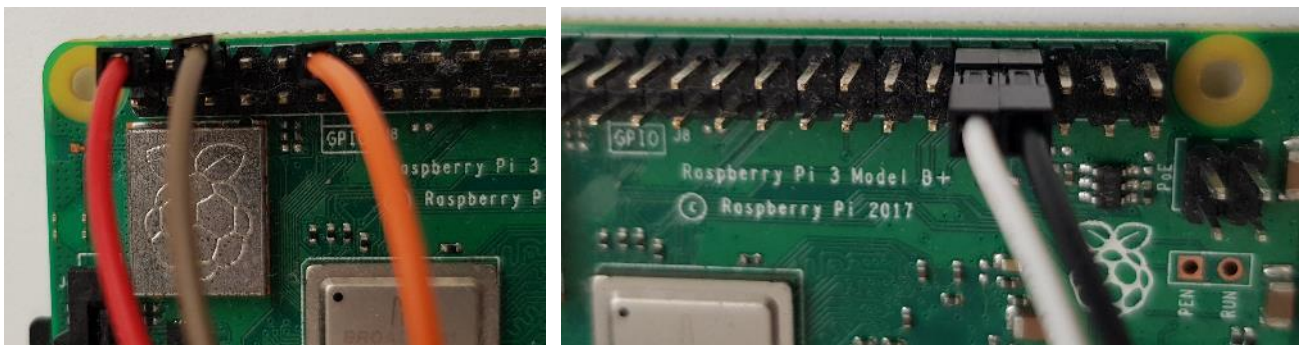


Figure 70 – Servo Motors Connected To The Raspberry Pi's GPIO Pins; Steering Motor (Left) & Driving Motor (Right)

The 2430 brushless motor also has 3 output wires; a white PWM wire, a black ground wire, and a red Battery Eliminator Circuit (BEC) wire (Figure 69 – Right). The white wire was plugged into another PWM capable GPIO pin (physical pin 32), the black wire was attached to a ground pin (Figure 70 – Right). The red wire was not connected at all as it can burn out the Raspberry Pi's processor if connected (AGTx, 2019). The brushless motor requires more than 5V to operate, if 5V or less is supplied, then the motors' ESC low voltage cut-off initiates, this means that an external battery will be required to power the 2430 motor to drive the ego vehicle.

“Your motor battery is already powering the ESC's BEC, so you should never connect the ESC's red wire to anything, because double-powering will cause voltage confusion (ampere overload)” (www.linuxcircle.com, 2017)



Figure 71 – Both Motors Connected To The Raspberry Pi GPIO Pins

With the two motors connected (Figure 71), MATLAB was now able to communicate and apply the required control to the three devices, provided that the connection between MATLAB and the Raspberry Pi had been established, as seen in Figure 72.

```
rpi = raspi('192.168.0.21','pi','raspberrypi');
% Connection To Raspberry Pi Board
picam = cameraboard(rpi,'Resolution','640x480','FrameRate',30);
% Connection To Pi Camera
picam.Rotation = 180;
% Rotate Camera As It's Mounted Upside Down
d = servo(rpi,12);
% Connection To Driving Motor
modeD = configurePin(rpi,12,'PWM');
% Configure GPIO Pin Mode To PWM
s = servo(rpi,18);
% Connection To Steering Motor
modeS = configurePin(rpi,18,'PWM');
% Configure GPIO Pin Mode To PWM
enableSPI(rpi)
% Enable SPI On Raspberry Pi
IRSensor = spidev(rpi,'CE0');
% Create Connection To IR Sensor
```

Figure 72 – MATLAB Code To Establish Connection To The Raspberry Pi Board, Camera, Sensor & Motors

The creation of the battery pack came next. Four Samsung INR18650-30Q 3000mAh lithium-ion cells were purchased for this project, which when wired in parallel, providing a total of 12000mAh. The specifications for these batteries can be found in section 8.4. The equations below are provided by Multicopters' LiPo calculator (Multicopter, 2019).

$$\text{Total Capacity (mAh)} = 1 \text{ Cell Capacity (mAh)} * \text{No. of Cells} = 3000 * 4 = 12000 \text{ mAh}$$

8-1

$$\text{Maximum Discharge (mAh)} = \text{Total Capacity (mAh)} * \text{Discharge Rate (C)} = 12000 * 0.2 = 2400 \text{ mAh}$$

8-2

$$\text{Constant Discharge Rate (A)} = \frac{\text{Max Discharge (mAh)}}{1000} = \frac{2400}{1000} = 2.4 \text{ A}$$

8-3

$$\text{Runtime (Hours)} = \frac{\frac{\text{Total Capacity (mAh)}}{1000}}{\text{Constant Discharge Rate (A)}} = \frac{\frac{12000}{1000}}{2.4} = 5 \text{ Hours}$$

8-4

When all the electrical components connected to the Raspberry Pi, there is a maximum current draw of 1.8 amps (A), where one cell supplies 0.6A into the system, so to successfully have enough leeway on the current draw so as not to put excess strain onto the cells, four will be used in parallel to provide a constant discharge rate of 2.4A. As seen in equation 8-4, this should enable the Raspberry Pi to be powered for a minimum of 5 hours of constant use, should the maximum current be drawn.

With the calculations done and the results satisfactory, constructions for the battery pack could now take place. The leaf springs were attached onto the sliding panels for the battery housing, to gain the correct placements. Strips of wire were cut to the correct lengths, each leaf spring needed to be connected together, and then joined to the power bank board to complete the circuit, as seen within Figure 73.

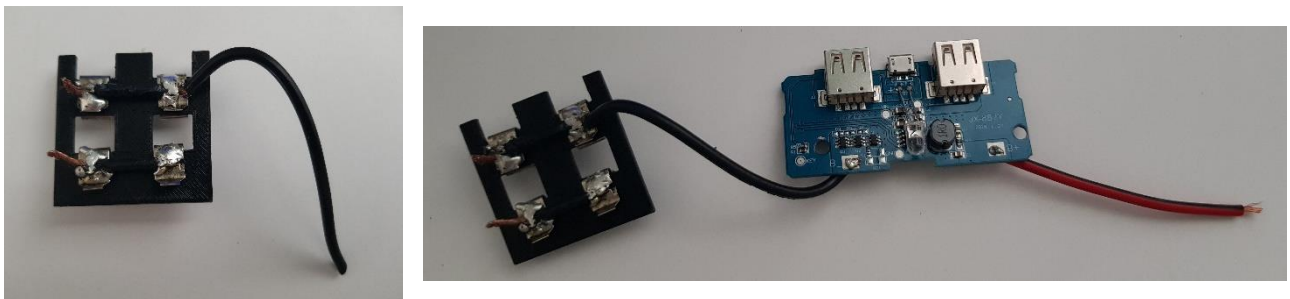


Figure 73 – Soldering Of The Leaf Springs (Left) & Soldering To The Power Bank Board (Right)

With the lengths of wire ready, they were soldered onto the leaf springs with the continuity checked upon each joining, ensuring there will be no problems further down the line. To guarantee that each battery will be used and there is no short paths for the current to take, the circuit was wired to take 'cell 1' for the positive side (Figure 74 – Left), and 'cell 4' for the negative side, (Figure 74 – Right).

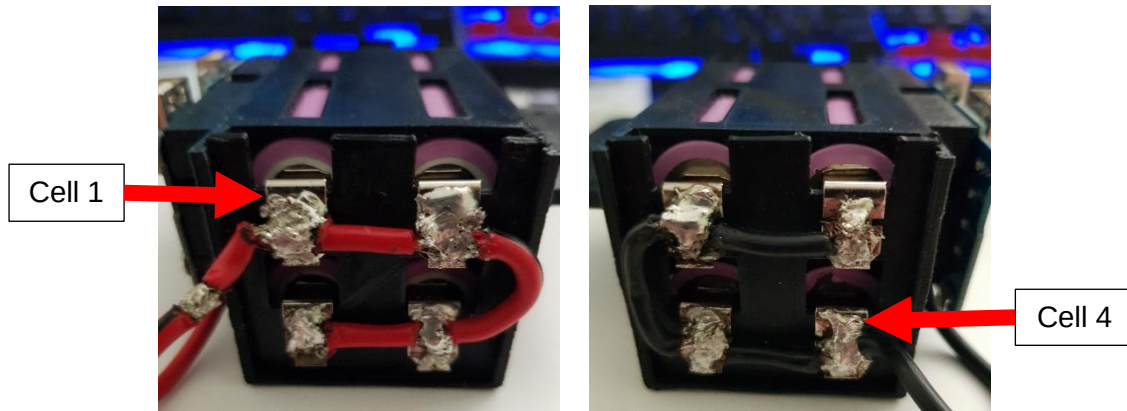


Figure 74 – Soldering Of The Battery Pack; Positive (Left) & Negative (Right)

The power band board implements a MC34063 monolithic control circuit which uses a PWM controller to switch between emitting and collecting current to the on-board capacitors. When the capacitors are charged in parallel and then discharged in series, they can double to voltage that has been inputted into the circuit, but with the use of resistors, this voltage can be regulated into a desired output. Which for this project, is 5V.

Once the battery pack had been created, a digital voltage tester was placed into a USB slot of the power bank, in addition to a USB variable resistance loader to apply either a 1A or 2A load onto the system.

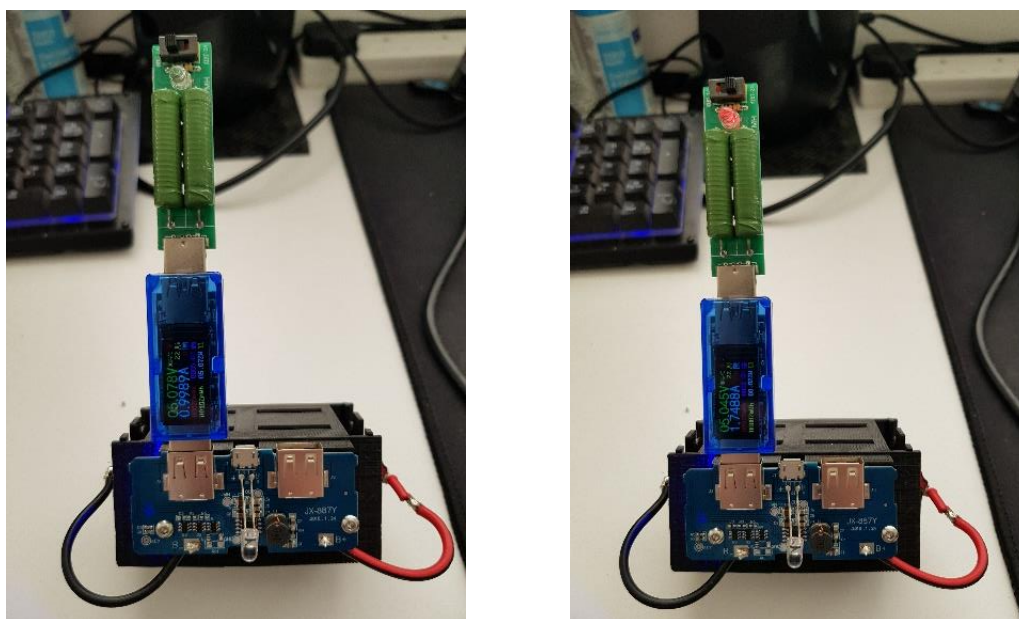


Figure 75 – Battery Pack Connected To A Variable Load; 1A (Left) & 2A (Right)

The battery pack was switched on and monitored to ensure that no cell was getting hot and that the correct amount of current was drawn from the system. Figure 75 show the both the 1A (Left) and 2A (Right) loads onto the battery pack, after this test, the cells were removed from the housing to determine if any cell had warmed up, signifying thermal runaway.

“A thermal runaway can be defined as a temperature-triggered exothermic process that produces heat faster than the battery can cool, thus leading to a temperature increase, which in turn sets yet another such reaction, eventually leading to an aggressive fire.” (Odne Stokke Burheim, 2018)

The final test was to use a USB to Micro USB cable, in addition to the digital voltage tester to monitor the current draw and determine if the battery pack could maintain a stable current draw while the Raspberry Pi was in operation. (Figure 76)

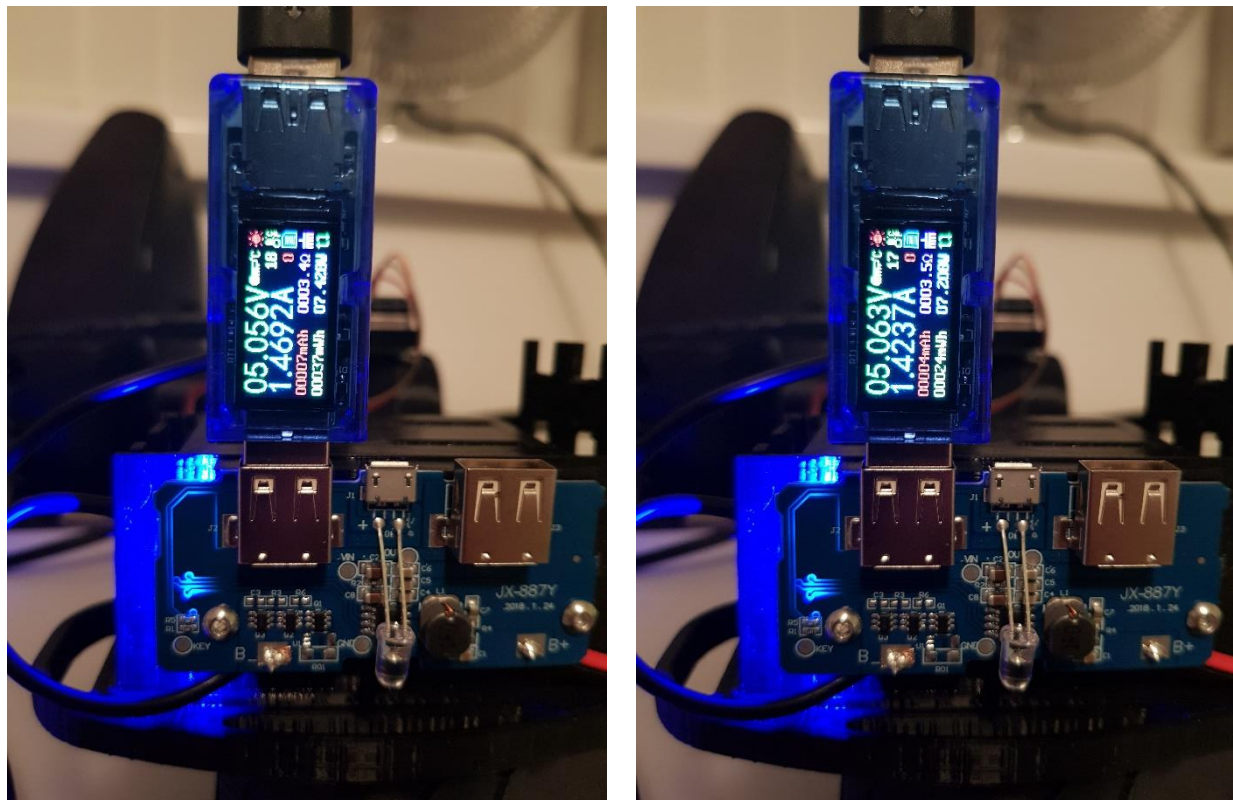


Figure 76 –The Battery Pack Connected To The Raspberry Pi Displaying The Current Draw; 1 Minute After Connection (Left) & 5 Minutes After Connection (Right)

8.1.1.3. Sensor Integration

To accurately gauge the distance to an obstacle in front of the ego vehicle, sensors would be required send out their respective wavelength signals and echoes. They would then communicate this response to the Raspberry Pi, and then MATLAB, so that appropriate control can be applied to the ego vehicle, should a collision be detected.

To idealise the circuit, both sensors were connected to the Raspberry Pi through a breadboard, guaranteeing the components and wiring work as intended before securing the connections permanently with solder.

8.1.1.3.1. Infrared Sensor

The Sharp GP2Y0A02YK0F infrared sensor operates by sending out a projection of infrared light directly forwards; an echo is then sent back from any object in its path sending an analogue signal and voltage through the 'data' wire to be interpreted by the Raspberry Pi, and the distance calculated using MATLAB.

"IR distance sensors output an analog (sic) signal, which changes depending on the distance between the sensor and an object." (Bakker, 2019)

Unfortunately, since the Raspberry Pi does not contain any built-in analogue functionality, it requires a MCP3008 analogue to digital convertor (ADC) (Microchip, 2008) as it is unable to read the sensor. The MCP3008 contains 16 pins but only nine of them will be used for this project, Figure 77.

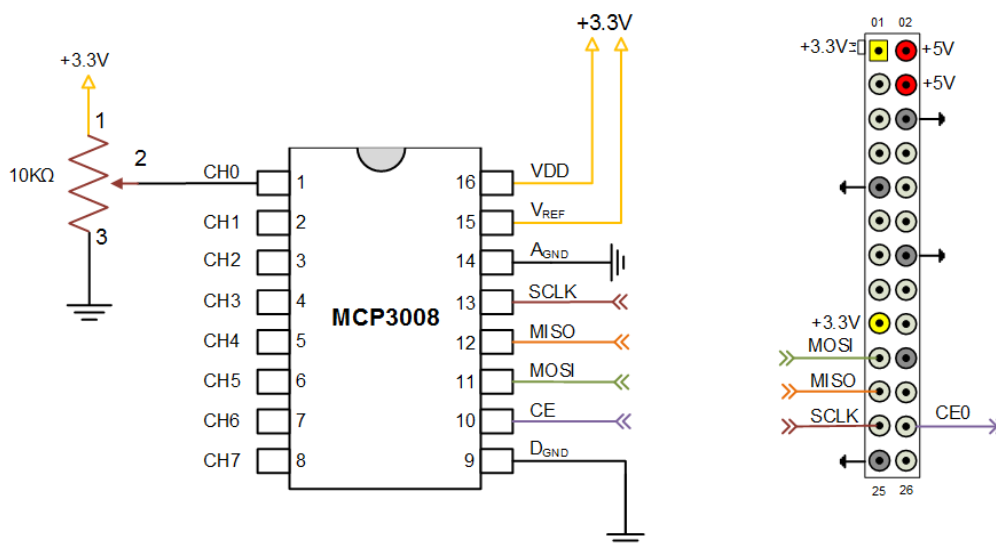


Figure 77 – Wiring Layout For Connecting The MCP3008 ADC To the Raspberry Pi (Mathworks, 2020)

The sensor itself consists of three wires; a red positive wire, a black negative wire and a yellow ‘data’ wire. The positive and negative wires were connected directly to the respective Raspberry Pi’s GPIO 5V (GPIO physical pin 2) and ground pins, however, the ‘data’ wire was connected to the channel zero (‘CH0’) pin of the MCP3008 ADC, leaving eight connections to make from the output side of the MCP3008 to the Pi’s GPIO pins, all seen within Table 1.

Table 1 – Wiring Connections For The MCP3008 & The Raspberry Pi (Anderson, 2020)

MCP3008 Pin	Pin Name	Pin Description	GPIO Physical Pin
16	VDD	Digital Voltage (3.3V)	1
15	VREF	Reference Voltage (3.3 V)	17
14	AGND	Analogue Ground	9
13	SCLK	Serial Clock	23
12	MOSI	Master Output Slave Input	19
11	MISO	Master Input Slave Output	21
10	CE	Chip Enable	24
9	DGND	Digital Ground	25

Once the wiring had been completed, alterations to the software needed to take place to enable the Raspberry Pi’s Serial Peripheral Interface (SPI). This can be done with the Pi connected directly to a display device/monitor by means of a HDMI cable, by remote interface to the Terminal (equivalent to Window’s Command Prompt) through a Secure Shell (SSH) through MATLAB, or remote access to the Pi’s full desktop environment using Virtual Network Computing (VNC) allowing access from devices such as; a computer or a mobile phone.

Because the device would need to be restarted throughout this process, it is simpler to connect a display device directly through the HDMI port on the Pi, along with a keyboard and mouse. It was then possible to alter the Raspberry Pi’s configuration options to enable the SPI interface with the following steps. (Raspberry Pi Tutorials, 2018)

1. From the Pi’s Terminal type “*sudo raspi-config*”, select “*Interfacing Options*”, then “*SPI*”, then “*Enable*”, a prompt to reboot the system *may* appear, it is highly suggested to proceed with this reboot.
2. After the reboot, again using Terminal, type “*sudo nano /boot/config.txt*”, this will open the configuration file for the Pi’s operating system (Rasbian) start up (boot) procedure. Add the following line to the bottom of this file, “*dfparam=spi=on*”. This will ensure that the SPI interface is enabled every time the Raspberry Pi starts. Reboot the Pi once this has been completed.

3. It is then advisable to ensure that the SPI interface has been enabled. From Terminal, type “lsmod”, a list of modules enabled on the device is then displayed. If “spi_bcm2708” or “spi_bcm2835” is shown, then the SPI interface has been successfully activated, if not, repeat steps 1-3.

Once this process has been completed, connect the Raspberry Pi to MATLAB, then enable then SPI interface through MATLAB and create a connection to the infrared sensor (Figure 72). This completes the systematic process of physical connections and software alterations required for the sensor to communicate with both the Raspberry Pi and MATLAB. The sensor can now be operated and by using a MATLAB code, read the voltage returned from the sensor.

Figure 78 shows the process of sending a signal from the sensor and converting the echo into a readable voltage value using the MCP3008 to convert the signal. This is then multiplied by a polynomial equation supplied by Raspberry Pi Tutorials (Raspberry Pi Tutorials, 2018) to calculate the distance to a high degree of accuracy up to 2 metres away, depending on the reflectivity of an object's surface.

```
Data = uint16(writeRead(IRSensor,[1,bin2dec('10000000'),0]));
% Create A 16-bit Unsigned Integer Array From Bits Sent & Retrieved From Sensor
Highbits = bitand(Data(2),bin2dec('11'));
% Logical Binary Operation Of Multiplying "Data" And Binary '3' For Comparison
Voltage = double(bitor(bitshift(Highbits,8),Data(3)));
% Transform Either Highbits Or Data Into A Single Value
Voltage = (3.3/1024) * Voltage
% Multiply Value By (3.3 Volts Divided By 1024 Analogue Bits) For Voltage

Distance = (16.2537 * Voltage^4 - 129.893 * Voltage^3 + 382.268 * Voltage^2 - 512.611
* Voltage + 301.439) * 10;
% Fourth Order Polynomial Equation Converting The Voltage Into Linear Distance
```

Figure 78 – Read Data From Infrared Sensor and Convert That Value Into A Voltage, Then Into A Distance

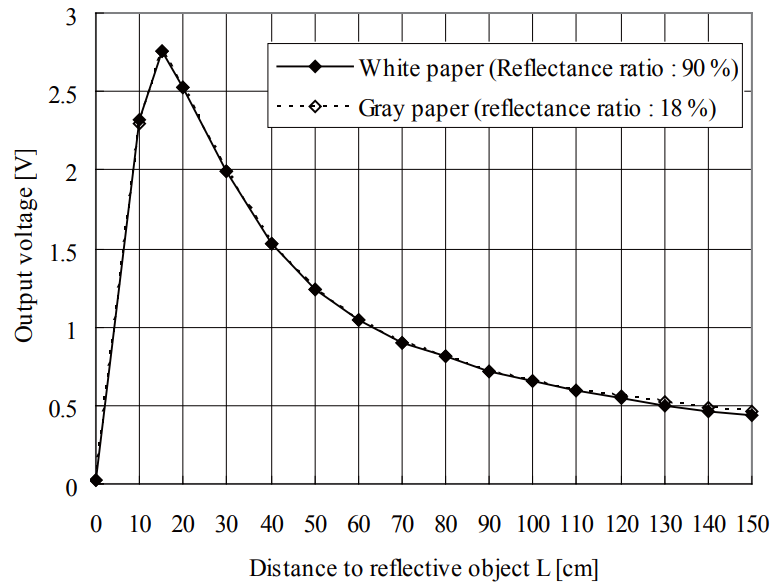


Figure 79 – Graph Depicting The Voltage Output Based On The Relevant Object Distance (Sharp, 2006)

As seen in Figure 79 after 15cm the voltage returned drops exponentially, this means that a polynomial equation can be used to calculate the distance from the voltage returned from the sensor, as seen in Figure 78. This equation returns values in centimetres; hence it is multiplied by 10 to convert into millimetres.

An attempt at recreating this polynomial was undertaken by curve fitting the graph (Figure 80), unfortunately, this proved to be a fruitless effort. Since the sensors recommended accurate distance is 20-150cm, the graph provided in the datasheet only ranges from 0-150cm, the peak at 15cm is hard to recreate with a polynomial with high degree of accuracy without excessive curvature, leading to inaccuracies throughout.

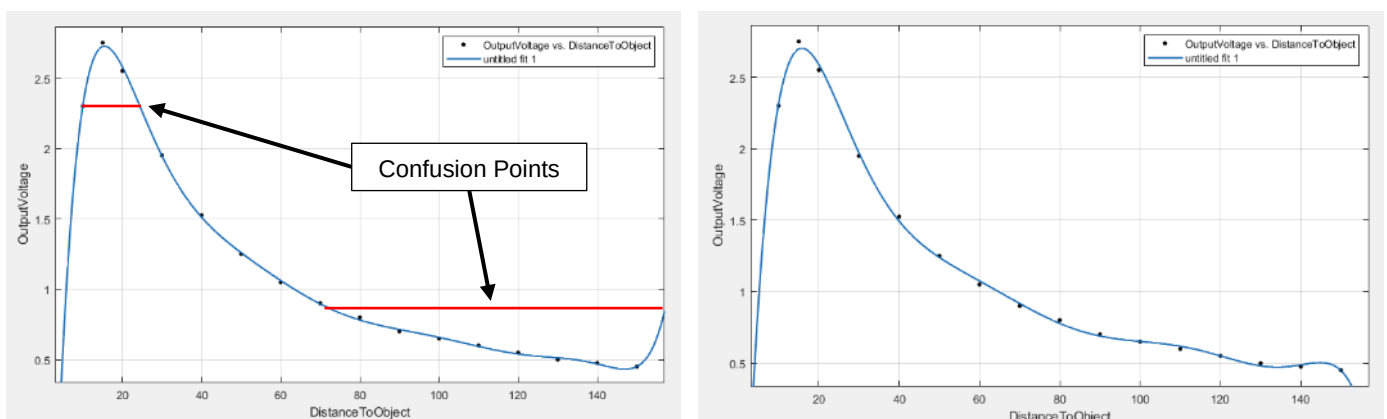


Figure 80 – Attempts At Gaining A Polynomial Equation To Calculate The Distance Using The Infrared Sensor; 9th Degree Polynomial (Left), 8th Degree Polynomial (Right)

This same peak provides inaccurate values below 15cm, this is because if 2.8 volts is returned, the graph indicates this can be either 10cm or 25cm. This was noticed during testing, when an object was closer than 15cm to the sensor, the distance value would increase from 15cm, to 20cm up to 30cm before an object was physically touching the sensor, blocking all signals. The accuracy of the sensor was measured using rulers seen in Figure 81, this concluded that the sensor was accurate, $\pm 10\text{mm}$, up to 110mm away.



Figure 81 – Testing The Accuracy Of The Infrared Sensor; Object At 400mm Away (Left), Closest Object Could Get To The Sensor (110mm) Before Distance Increased Again (Right)

8.1.1.3.2. Ultrasonic Sensor

The HC-SR04 ultrasonic sensor works using the same principle as the infrared sensor, but instead of emitting light it emits sound waves. The time taken between the ultrasonic waves being sent out and then returned through the echo can be used to calculate the distance to an object. Given the known speed of sound at sea level as 343 m/s at 20°C. (BBC Bitesize, 2019)

“Ultrasonic sensors, like many others, use a single transducer to send a pulse and to receive the echo. The sensor determines the distance to a target by measuring time lapses between the sending and receiving of the ultrasonic pulse.” (Burnett, 2020)

The sensor has four pins that need to be connected to the Raspberry Pi; VCC which connects to one of the 5V GPIO pins, GND connects to one of the many ground pins on the Pi, TRIG, which sends out the initial wave, plugs into a PWM capable pin (such as physical pin 33). ECHO, which receives the echo of the sound wave, requires more intricate circuitry, Figure 82. The sensor requires 5V to operate, but if the GPIO pins on the Pi receive more than 3.3V, the pin, and potentially the whole Raspberry Pi, will be fried.

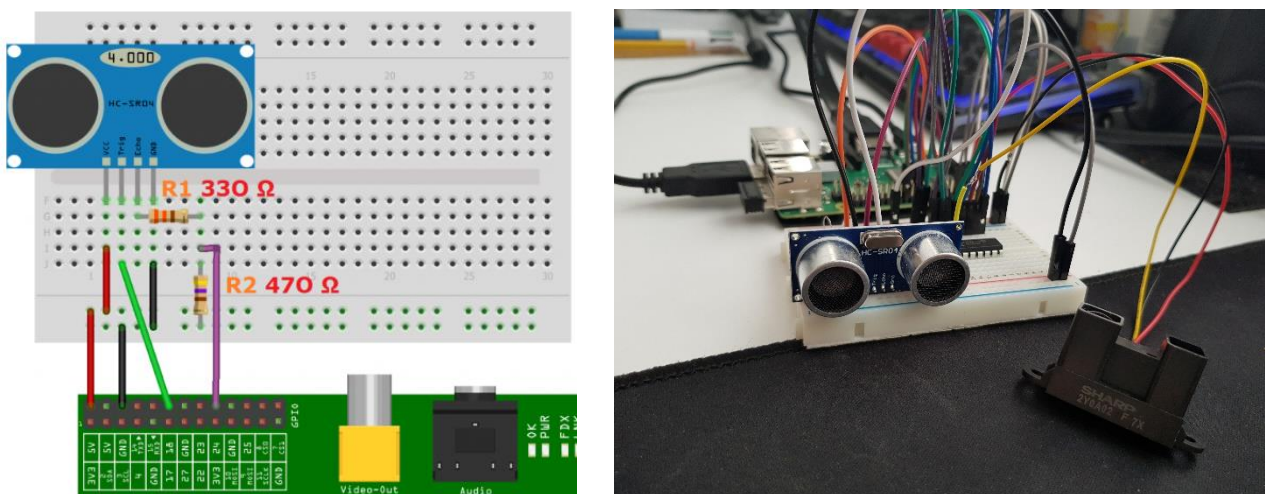


Figure 82 – Wiring The Ultrasonic Sensor Into The Raspberry Pi (Raspberry Pi tutorials, 2019) (Left), Both Ultrasonic And Infrared Sensor Connected To The Pi Using A Breadboard (Right)

To counteract this, two resistors will be used to bring the voltage down below 3.3V. the first resistor is a 330Ω (Ohm) that connects directly to the sensors' ECHO pin, the end of this resistor is connected in two places, the first to 'CE0' physical pin 24 on the Pi, while the second uses another 470Ω resistor to connect the remaining current to the ground pin used for GND.

“We do this because the GPIO pins only tolerate maximal 3.3V. The connection to GND is to have a (sic) obvious signal on GPIO24. If no pulse is sent, the signal is 0 (through the connection with GND), else it is 1. If there would be no connection to GND, the input would be undefined if no signal is sent (randomly 0 or 1), so ambiguous.” (Raspberry Pi tutorials, 2019)

With the physical connections made, the sensor should now be operational. Once connecting to the Pi with either a HDMI cable, a Secure Shell (SSH, or using Virtual Network Computing (VNC), Terminal can be used to create a script within python that operates the sensor can calculates the distance to an object in front with the command “*sudo nano ultrasonic_distance.py*”.

The following script (Figure 83, on the next page) has been provided by Raspberry Pi Tutorials to operate the sensor. This allowed the use of the command “*sudo python ultrasonic_distance.py*” within Terminal to send out an ultrasonic wave, wait for the echo to return, calculate the time taken, multiply it by the speed of sound, and then half it to determine the distance to the object.

The value is halved, because the time taken for the wave to travel to the object and then back to the sensor is encompassing double the distance.

The script worked as intended, unfortunately, there were two downsides. It constantly required the command to run the code to be inputted to calculate the distance, there was no way to automate this, and the operation could not be performed through MATLAB, only on the Pi, meaning it could not be implemented into the current script to control the ego vehicle.

Several attempts were made to overcome these problems but to no avail. At this point the infrared sensor was operational, so to get the vehicle up and running, the ultrasonic sensor was benched until a working method of integration could be found.

```

Python
1 #Libraries
2 import RPi.GPIO as GPIO
3 import time
4
5 #GPIO Mode (BOARD / BCM)
6 GPIO.setmode(GPIO.BCM)
7
8 #set GPIO Pins
9 GPIO_TRIGGER = 18
10 GPIO_ECHO = 24
11
12 #set GPIO direction (IN / OUT)
13 GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
14 GPIO.setup(GPIO_ECHO, GPIO.IN)
15
16 def distance():
17     # set Trigger to HIGH
18     GPIO.output(GPIO_TRIGGER, True)
19
20     # set Trigger after 0.01ms to LOW
21     time.sleep(0.00001)
22     GPIO.output(GPIO_TRIGGER, False)
23
24     StartTime = time.time()
25     StopTime = time.time()
26
27     # save StartTime
28     while GPIO.input(GPIO_ECHO) == 0:
29         StartTime = time.time()
30
31     # save time of arrival
32     while GPIO.input(GPIO_ECHO) == 1:
33         StopTime = time.time()
34
35     # time difference between start and arrival
36     TimeElapsed = StopTime - StartTime
37     # multiply with the sonic speed (34300 cm/s)
38     # and divide by 2, because there and back
39     distance = (TimeElapsed * 34300) / 2
40
41     return distance
42
43 if __name__ == '__main__':
44     try:
45         while True:
46             dist = distance()
47             print ("Measured Distance = %.1f cm" % dist)
48             time.sleep(1)
49
50         # Reset by pressing CTRL + C
51     except KeyboardInterrupt:
52         print("Measurement stopped by User")
53         GPIO.cleanup()

```

Figure 83 – Python Script To Operate The Ultrasonic Sensor Through The Raspberry Pi (Raspberry Pi tutorials, 2019)

8.1.1.3.3. Vehicle Integration

With the infrared sensor operational, it was required to be mounted onto the vehicle, the best position was at the direct front of the vehicle. The front bumper had already been designed but not 3D printed, this meant that it could easily be adapted to attach the sensor, but because of the mounting holes on the sensor, it needed its own mount to be designed to hold it in the correct orientation, on the front bumper.

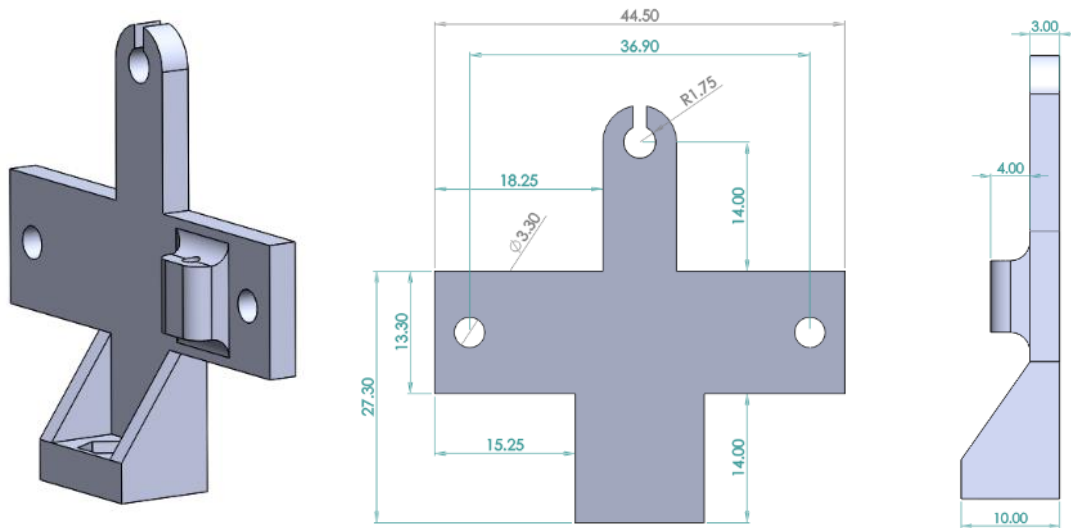


Figure 84 – Infrared Sensor Mount; CAD Model (Left), Front View (Middle), Side View (Right)

The sensor was measured for its height, width and the distance between the mounting holes. A basic rectangular shape was created in Solidworks that the sensor would bolt onto, a shank was created to poise the sensors' horizontal centre line above the top plate of the ego vehicle. This meant that the sensor would not detect small bumps in the road, and only objects that would pose a threat/danger to the ego vehicle.

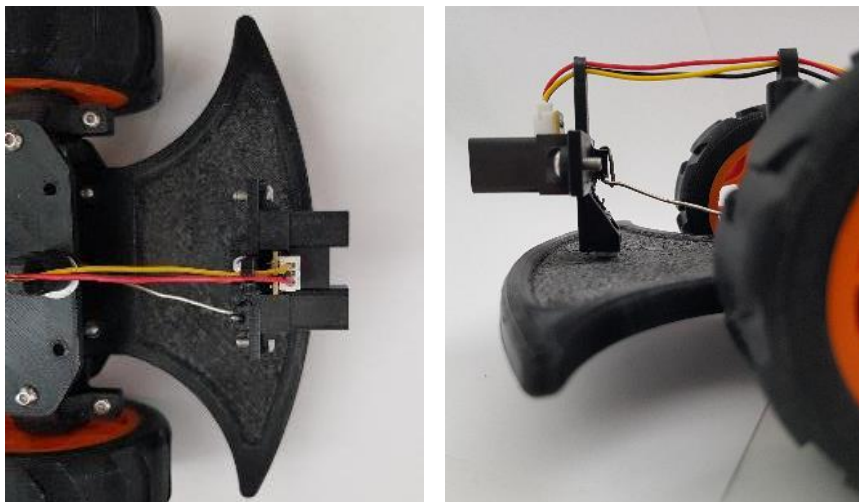


Figure 85 – Top View Once Assembled (Middle) Side View (Right)

To stop the wires from the sensor obstructing the camera, a small wire tidy was created to keep the wires out of the way of the camera. A strip of heat shrink covered the wires because the perceptive portion of the code, mistook them as a lane marker during testing.

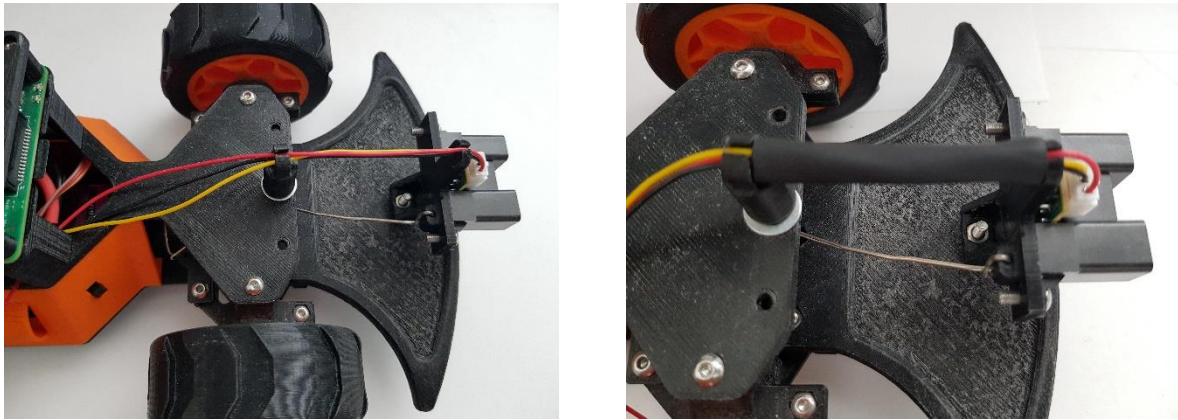


Figure 86 –Tidy For The Sensor Wires (Left), Heat Shrink To Prevent Perceptive System Being Misled (Right)

The original half-size breadboard used to connect and test the wiring for the sensor was too big to fit anywhere on the vehicle. It was situated at the rear of the vehicle just in front of the battery pack, but during testing, some of the pins in the wires bent, meaning this was not a viable location. A quarter-size breadboard was sourced to solve this issue, this meant that with the sticky-back on the reverse side of the breadboard, it could be situated onto the top cover for the Raspberry Pi. This enabled the wires within the circuit to be shortened for a cleaner aesthetic feel.

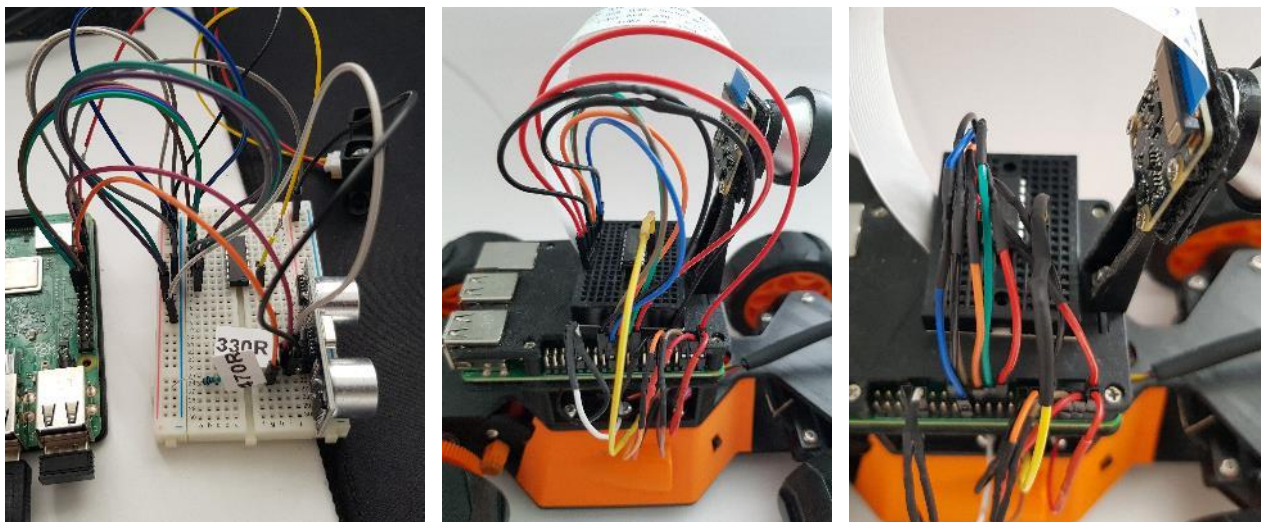


Figure 87 – Shortening All Of The Wires For A More Aesthetically Pleasing Feel; Original Setup (Left), New Placement (Middle), Wires Shortened (Right)

Due to an overlook with the measurements taken, the position of the hole to bolt the infrared sensor through caused the sensor to hang over the bumper, shown in Figure 88 – Left.

The system was turned on one morning to test a new addition to the code. Everything was connected but only a basic script ran to check operation of the driving motor was still functional, so the distance measurements and stopping procedure were not activated. This led to a collision with a fire door.

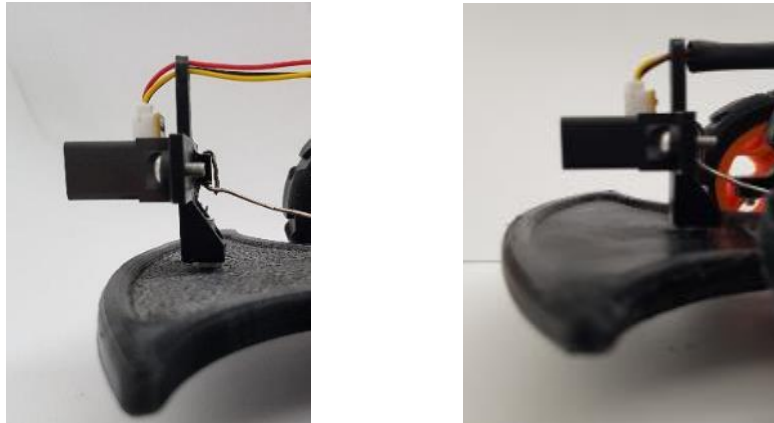


Figure 88 – Infrared Sensor Mount; Overhanging The Bumper (Left), New Design (Right)

The annoyingly avoidable event snapped the sensor mount and broke the hole in the bumper, but thankfully the sensor remained unscathed. Figure 89.



Figure 89 – Aftermath Of Devastating Autonomous Vehicle Collision With Fire Door That Appeared Out Of Nowhere

A new bumper was designed with the positioning hole for the sensor being back moved 30mm. This brought the sensor back under the protection of the bumper, in case of a collision. Figure 88.



Figure 90 – New Bumper vs. Original (Left), New Bumper Design With Infrared sensor Mount Repositioned

The final aspect that was considered for the sensor's integration was the potential for the sensor to turn in relation to the front wheel. This was to aid the sensors ability to correctly gauge the distance in front, while the vehicle was engaging a corner. The idea was the connect the sensor mount to the steering rack, similar to the steering and the motor, by means of a thick paper clip. As the steering rack moves to the right, turning the wheels left, the edge of the sensor mount would be pushed to also face left, and vice versa to face the right.

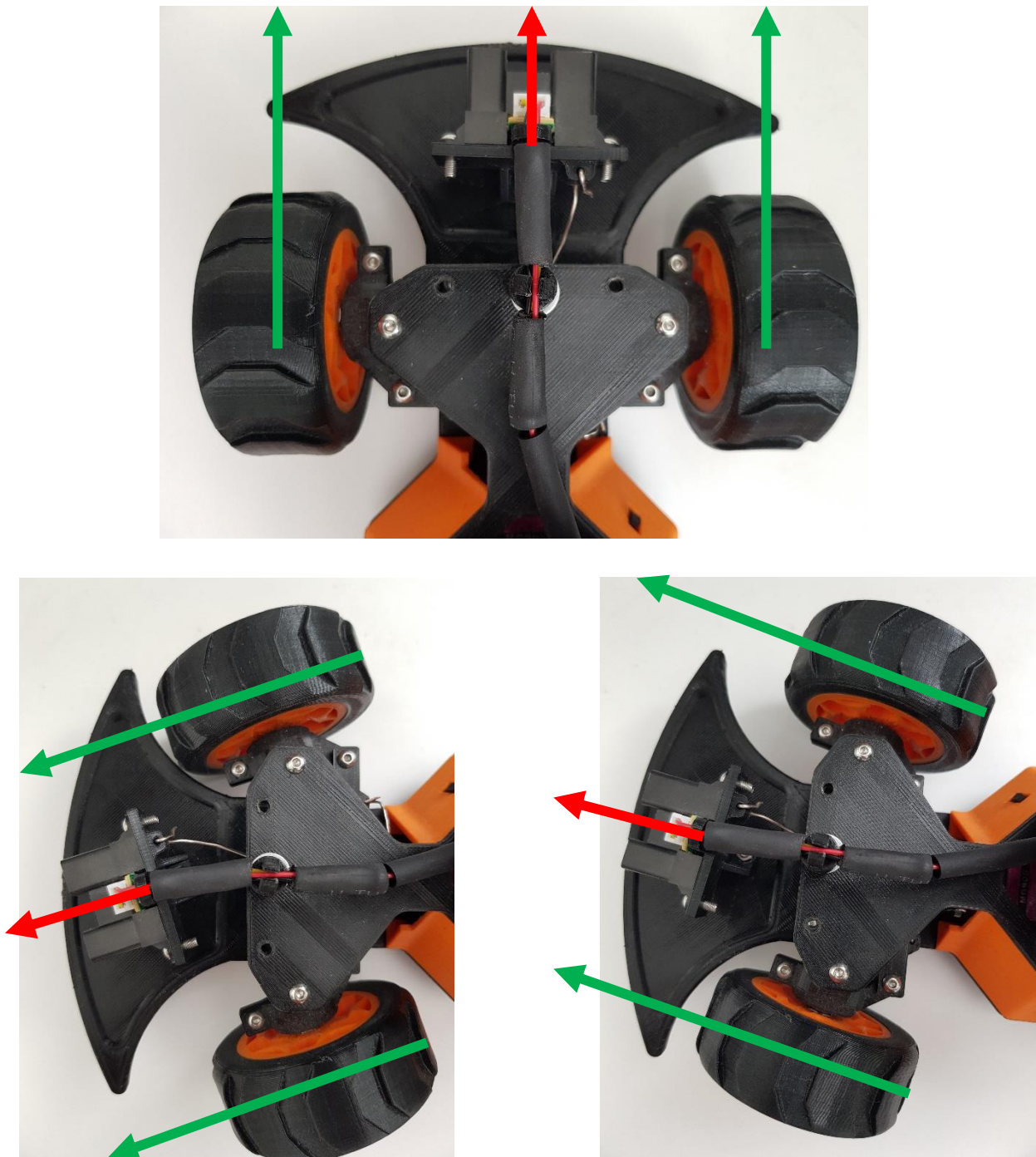


Figure 91 – Effect Of Connecting The Infrared Sensor Mount To The Steering Rack; Static (Top), Steering Left (Bottom Left), Steering Right (Bottom Right)

8.1.2. Code Construction

To understand the capability of MATLAB's Automated Driving Toolbox (Mathworks, 2017), an Automated Driving course was undertaken by the author in July 2019, hosted by Mathworks employee John Moon. (Mathworks, 2019) This course gave an understanding on the functions within the toolbox and how they should be used to ascertain the best performance of the perceptive systems within an automated vehicle. Within this section, each function will be broken down in the sequence they are used within the code and how they work.

With the connection to the Raspberry Pi board already established (Figure 72), the first action taken was to create a '*monoCamera*' object, a virtual camera sensor that contains configuration information to aid with transforming point locations between image coordinates and vehicle coordinates (Mathworks, 2020). This required key intrinsic and extrinsic parameters about that physical camera. The intrinsic information relates to the Focal Length (FL) distance between the lens and the image sensor when the subject is in focus (Nikon, 2020) the calculation for which is shown within section 0, the Principal Point (PP) on the image plane onto which the perspective centre is projected (PCI Geomatics, 2019), designated as the direct centre of the image, and the image size in pixels. The extrinsic parameters include; the height of the camera to the road surface in metres, the pitch angle of the camera along the X-axis, the yaw angle along the vehicles' Z-axis and the roll angle along the X-axis, all in degrees. With all these pieces of information, the '*monoCamera*' object can be successfully created.

```
% Camera Intrinsics & Sensor Setup

FL = [2714.28 2714.28];
% Define Focal Length
PP = [320 240];
% Define Principle Point
IS = [480 640];
% Define Image Size

intrinsics = cameraIntrinsics(FL,PP,IS);
% Combine Intrinsics

height = 0.1415;
% Define Camera Height From Ground
pitch = 11.25;
% Define Camera Pitch
roll = 0;
% Define Camera Roll

sensor = monoCamera(intrinsics,height,'pitch',pitch,'roll',roll);
% Create monoCamera Sensor
```

Figure 92 – MATLAB Code To Create '*monoCamera*' Object

To display the final frame after computation of the lane boundaries as a video rather than images with a constantly updating ‘figure’ window, the “*player*” function can be introduced.

```
player = vision.DeployableVideoPlayer;  
% Setup Player For Loop
```

Figure 93 – MATLAB Code To Initiate Deployable Video Player

A ‘while’ loop was now initiated, within this would be all the functions that control and manipulate a snapshot taken with the Pi camera to control the vehicle, this loop would be required to run indefinitely, hence it’s criteria would be a logic statement that could never be false.

```
while 1 > 0  
% Define 'while' Loop Criteria  
  
frame = snapshot(picam);  
% Take A Snapshot From The Pi Camera
```

Figure 94 – MATLAB Code To Begin The ‘while’ Loop And Take A Snapshot From The Pi Camera

The first operation of the ‘while’ loop would take a snapshot from the Pi camera and save it to a variable called “*frame*”. This image can be transformed into a bird’s eye view in two parts. Firstly, the function ‘*birdsEyeView*’ create a view of a 2D scene using Inverse Perspective Mapping (IPM).

“IPM takes the frontal view as input, applies a homography transformation, and produces a top-down view of the scene by mapping the pixels to a different 2D-coordinate frame” (Bruls, 2018).

```
% Birds Eye Parameters  
roi = [0.25 0.6 -0.25 0.25];  
% Region Of Interest [BottomY, TopY, LeftX, RightX]  
imgSize = [NaN 300];  
% Define BE Size [NoDefinedX, 300PixelsY]
```

Figure 95 – MATLAB Code To Define ROI & Image Size For Bird's Eye View

To achieve this bird’s eye view, the function requires three inputs; the previously created ‘*monoCamera*’ object, the Region Of Interest (ROI), and the size of the output image. The ROI is a 4-element vector, with the minimum and maximum of both the ‘X’ & ‘Y’ axis in metres. The output image size is in the form of a 2-element vector, determining the height (X-axis) and width (X-axis) of the bird’s eye view image, in pixels. For this application, the image was defined as 300 pixels wide, with the image height set to ‘NaN’ so that the function maintains the same pixel-to-world unit ratio once the image has been transformed, meaning the image

will not be distorted or elongated after the implementation of the second function (Mathworks, 2019). This is saved to a variable called “view”.

```
view = birdsEyeView(sensor,roi,imgSize);  
% Create Birds Eye View  
IBE = transformImage(view,frame);  
% Apply Birds Eye View
```

Figure 96 – MATLAB Code To Create A Bird's Eye View And Transformed Image

The second function called ‘*transformImage*’ uses the bird’s eye view previously created and transforms the input image “*frame*” into a bird’s eye view image. This function only requires two inputs, the bird’s eye variable “*view*” and the original image that will be transformed “*frame*”, the transformed image is saved to a variable “*IBE*”. The original input and transformed output images are shown in Figure 97.



Figure 97 – Original Camera Image (Left) & Transformed Bird's Eye View (Right)

The purpose of the transformed image “*IBE*” was to detect the lane markers on the road ahead of the ego vehicle, unfortunately, this image needed to be manipulated further to achieve this goal. “*IBE*” was converted to a greyscale image using the function ‘*rgb2gray*’ to “convert standard RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance” (Mathworks, 2019), save as variable “*IGray*”.



Figure 98 – Bird's Eye Image Converted To Greyscale

This helped the next function '*segmentLaneMarkerRidge*' to return a binary black and white image and by means of a lane ridge detector using the greyscale image "*IGray*", to convert point locations from vehicle coordinates to image coordinates. The function requires a minimum of three inputs; the greyscale image "*IGray*", the transformed bird's eye image "*IBE*", and an approximate lane marker width in metres. This function then displays the white lane markers on the road on a black background outputted into a variable "*BW*" (Figure 99).

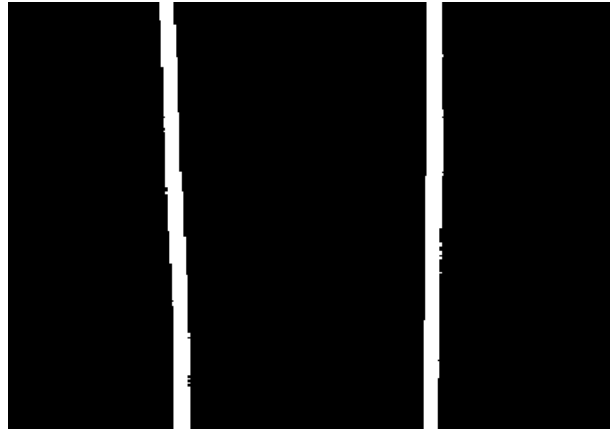


Figure 99 – Bird's Eye View Transformed Into A Binary Black & White Image

To gain the full potential and optimise this function, a sensitivity factor ranging between zero and one was incorporated. This came with its own pros and cons, if the sensitivity value was too low, barely anything would be detected, but if the value was too high, false detections and 'noise' would occur depending on the road surface, its condition and the weather. If the surface is shiny or wet, reflections may occur and cause 'noise' within the image, misleading the algorithm that detects the lane markers, severely impacting the accuracy of the vehicles steering. The effects of this can be seen within Figure 100.



Figure 100 – Black & White Bird's Eye View Displaying 'Noise'

Because "*BW*" is a binary image, the lane marker points can be extracted from the image using the '*find*' function. This treats the image as a matrix and outputs the lane marker points along the 'X' & 'Y' dimensions into a matrix of rows and columns "*[y,x]*".

```
[y,x] = find(BW);
% Lane Marker Points For X & Y Dimensions

pointsVCS = imageToVehicle(view,[x,y]);
% BE -> Vehicle (Transform Image Co-ordinates To Vehicle Co-ordinates)

lanes = findParabolicLaneBoundaries(pointsVCS,0.013);
% Create Model For Parabolic Lane Boundaries Using Vehicle Co-ordinates System
```

Figure 101 – MATLAB Code For Retrieving Lane Marker Points And The Conversion To Vehicle Coordinates And Parabolic Lane Boundaries

The lane markers points can now be converted from bird's eye view image coordinates to vehicle coordinates with the function '*imageToVehicle*' using the variable "view" and the matrix "[y,x]" saved within the variable "pointVCS". This transforms the image pixels to real-world units, paving the way for the next function '*findParabolicLaneBoundaries*' to take the coordinates categorised within the matrix "[y,x]", along with the approximate width of the lane markers and link them together using a random sample consensus (RANSAC) algorithm and outputs an array called "lanes", containing the coefficients of its second-degree polynomial equation. (Mathworks, 2019)

"RANSAC is a resampling technique that generates candidate solutions by using the minimum number observations (data points) required to estimate the underlying model parameters." (Derpanis, 2010)

To finish the computation of the lane boundaries, the '*computeBoundaryModel*' function computes the X-axis world coordinates from specified X-axis coordinates which are saved as a row vector variable "xIn". The X-axis coordinates relate to the real-world location of the lane boundaries, for this, it was easier to link the pre-defined ROI dimensions, specifically the minimum and maximum X-axis. This creates the output "yOut" to the same dimensions as the bird's eye view image "IBE".

```
xIn = roi(1,1):0.01:roi(1,2);
% Create X Axis Coordinates Variable Using ROI Dimensions
yOut = computeBoundaryModel(lanes,xIn);
% Compute Boundary Model Using Parabolic Lanes & X Axis
```

Figure 102 – MATLAB Code To Computation Of Boundary Model

Now, to observe the results and therefore the accuracy of the previous functions, the detected lane boundary model "yOut" can be plotted onto a graph showing the parabolic lanes. This is a visual representation of the direction the vehicle will take, this is a simplistic but vital validation tool to have, if the road is curving left, but the graph displays straight lines, then the previous steps require debugging to solve the problem that has occurred, Figure 104.

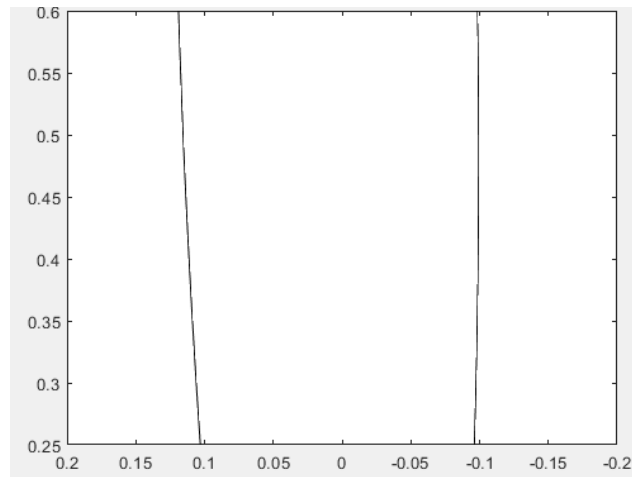


Figure 104 – 'yOut' Variable Plotted To Determine Accuracy

Should the graph be considered a success, the lanes can be extracted from “yOut” and defined separately into the both the left and right lanes. To begin, a new variable “horz” was created to gather the lane boundary information from “yOut”, this was then compared against the original parabolic lanes (“lanes”) where the numbers greater than zero were designated to the left lane, as opposed to all the numbers less than zero, allocated to the right lane. Each lane was saved as its corresponding variable either “left” or “right”.

```
% Defining Left & Right Lanes
horz = yOut(:,1);
left = lanes(horz > 0);
right = lanes(horz < 0);
```

Figure 103 – Defining Left & Right Lanes

Each parabolic lane can be annotated onto the original bird’s eye view image, this makes a significant improvement in how easy it is to compare the accuracy of the detected lane boundaries to the actual road markers it is interpreting. The function ‘insertLaneBoundary’ allows this, each lane is computed separately so a different colour can be applied to help with identification.

The left lane was computed first, using the original bird’s eye view “view”, the transformed image “IBE”, the left lane variable “left” and the designated start and end length of the annotations saved as a variable “lanesBE”, colour can then be dictated by adding the ‘Color’ parameter and then designating a desired colour. The right lane used the same inputs as the left, but this time “IBE” was replaced with “lanesBE” to overlay the previous iteration, then using the “right” lane variable along with defining a different colour for the annotation (Figure 105).

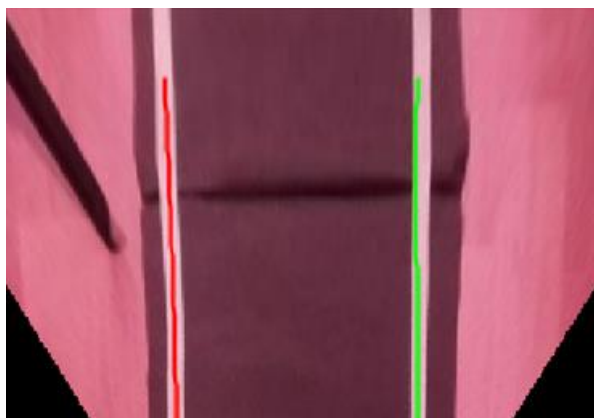


Figure 105 – Left & Right Lanes Annotated Onto The Original Bird's Eye View

These annotations can also be inserted into the original snapshot taken from the Pi camera using the same *'insertLaneBoundary'* function. This time however, it is more appropriate to add the lanes onto to the snapshot at the same time in the same colour. The inputs for this are the original snapshot *"frame"*, the parabolic lanes array *"lanes"*, the monoCamera *"sensor"* and finally the designated start and end length of the annotations. This is saved as a variable *"lanesCam"*. The *"player"* function displays the *"lanesCam"* image, showing the user what the camera can see with an annotated overlay of the computational lane boundaries (Figure 106).



Figure 106 – Parabolic Lanes Annotated Onto The Original Cameras' Image

This completed the perception portion of the code. The next section links what the camera and code perceive of the road ahead, to the control of the vehicle so that the appropriate action can be taken to follow the course of the road.

Before the vehicle control can be implemented, the path along the road will need to be calculated, to do this, the centre of the lane needs to be determined. By using the specified X-axis coordinates designated within the variable *"xIn"* transposed to a column vector, saved as *"xWorld"*, this is inputted into both the left and right lanes using the *'computeBoundaryModel'* function.

```
% Segment Lanes For Control Of Car
xWorld = xIn';
yLeft = computeBoundaryModel(lanes(1,2),xWorld);
yRight = computeBoundaryModel(lanes(1,1),xWorld);
```

Figure 107 – MATLAB Code To Associate Lanes For Steering Control

Two new variables can now be created for the left and right lane markers, the parabolic array “lanes” can be inputted into the function, this time however, separate terms within the array can be used to define each lane. The left lane “yLeft” uses the second column, while the right lane “yRight” uses the first. Once the function has computed both lanes, the direct centre can be found using equation 8-5 and entered into the variable “yCentre”.

$$yCentre = \frac{yLeft + yRight}{2}$$

8-5

With all three lane variables, another validation can be created into a plot to show both the lane markers and the path of the ego vehicle, using five functions that overlay one another. ‘birdsEyePlot’ creates a top down view plot of the road ahead within set limits in the ‘X’ & ‘Y’ dimensions, ‘laneBoundaryPlotter’ creates the parameters for the lane boundaries, which themselves are overlaid onto the plot with the ‘plotLaneBoundary’ function with the “xWorld” variable to determine the X-axis, which the “yLeft” and “yRight” plot their respective lane boundaries. The ego vehicles path along the road follows the same setup as the lane boundaries, ‘pathPlotter’ creates the parameters for the vehicle’s path which is overlaid with the ‘plotPath’ function, using the variables “xWorld” & “yCentre” variables. Put together, these functions create an incredibly useful validation tool in the same way as the “yOut” plot to check that the path is taking the vehicle in the correct direction. (Figure 108)

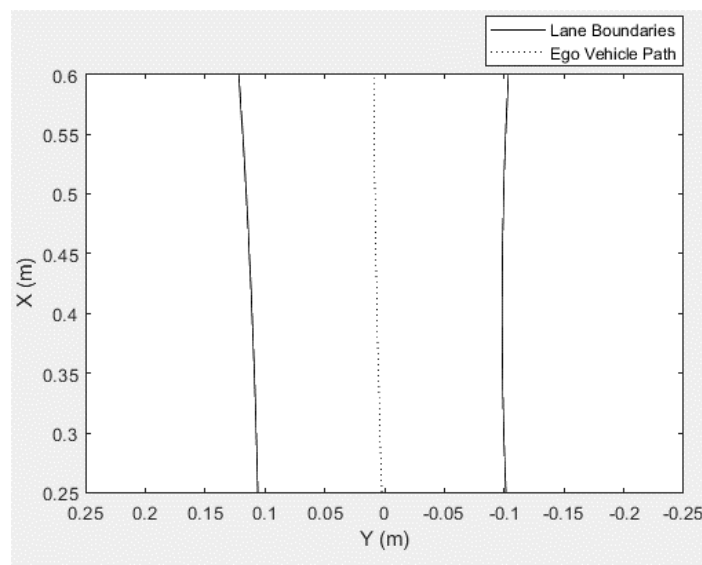


Figure 108 – Plot Displaying Lane Boundaries & Ego Vehicle Path

An equation was concocted to apply the appropriate amount of steering in the correct direction, this took a lot of trial and error to perfect, equation 8-6 shows the final iteration of the variable “Steer”.

$$\text{Steer} = 90 + (800 * y\text{Centre}(10,1))$$

8-6

“yCentre” will either provide a positive or negative value dependant on which direction the road ahead of the ego vehicle takes. The value from the tenth row of “yCentre” was used so that the code created enough steering angle before the vehicle turned into the corner. Before this method was implemented, the vehicle understeered every time it entered the corner until it had distanced itself far enough away from the centre of the lane to apply more, but not enough, steering angle. After some experimentation, the value of the tenth row was to be an adequate balance between over and understeer when the magnification factor was applied.

Unfortunately, this value is so small (± 0.05) that it has no effect on the steering direction, therefore, a magnification factor was implemented into the equation. During the code validation (section 3.1) the value of ‘50’ was used, but once the physical testing began (section 0) this value was again too small, henceforth, trial and error occurred until a more appropriate value was found which eventually became ‘800’. Since 90^0 is the central position of the servo motors, the resultant answer from “yCentre” and the magnification factor is either added or subtracted from this to follow the path correctly depending the sign of the value.

To help debug the code when errors occurred, the values from “Steer” were logged into another variable “SteerLog” so they could be viewed and interrogated if abnormal steering values appeared to help trace the issue. Several sessions were screen recorded during the debugging process, displaying each step of the perception process, with the steering angle displayed in MATLABs command window. Once the session was over, the recording could be inspected frame by frame to help determine the cause of the issue, providing a means to go back to the code, analyse that particular frame and how the problem could best be solved.

If abnormal and erratic values did appear, an attempt to cancel out these issues was undertaken. The initial step was to create a variable “OldSteer” before equation 8-6 takes place within the script, as shown below, to log the value of the previous loop iterations “Steer” value.

`OldSteer = Steer;`

Figure 109 – Redefining The Value Of Steer From The Previous Iteration

The manipulation of the Region of Interest (ROI) used to create a bird's eye image has been discussed within section 3.2.1. This is where it affected the steering value that was applied. If the maximum X-axis was reduced, the how far forward the value taken from "yCentre" would need to reduce as well. The further forward the value from "yCentre" could be taken away from the front of the ego vehicle, the more prepared it was to apply adequate steering around a corner. Figure 110.

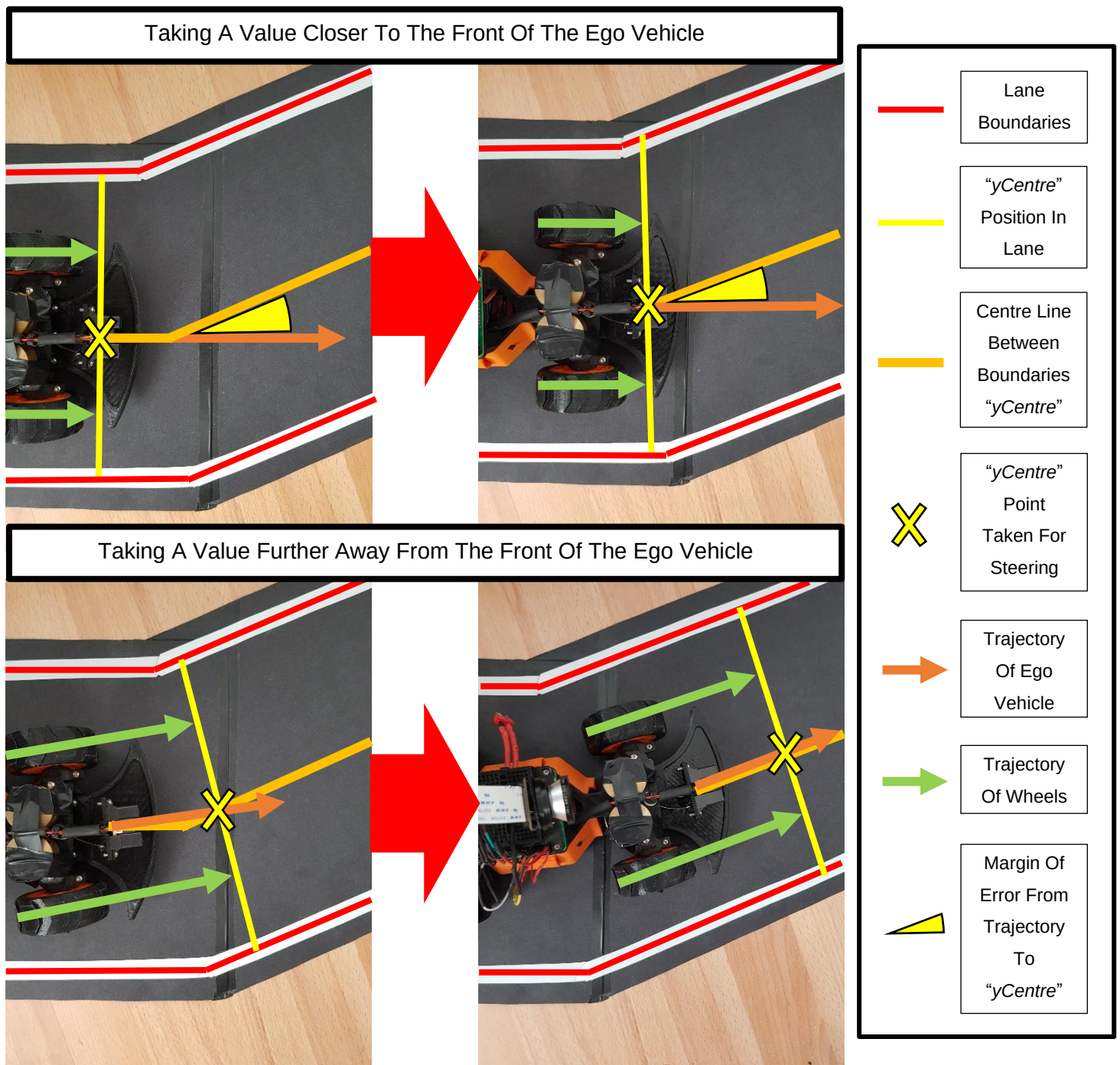


Figure 110 – Comparison Showing the Effect Of Taking Different "yCentre" Values; Closer To The Ego Vehicle (Top), Then Further Away From The Ego Vehicle (Right)

The issue arose when the maximum ROI length was reduced, there were not enough values with the “yCentre” variable would incur errors for trying to access a non-existent row value. Thus, the row value obtained, reduces to take a value closer to the front of the ego vehicle. Figure 111.

```

if roi(1,2) <= 0.25
    Steer = 90 + (900 * yCenter(1,1));
elseif roi(1,2) <= 0.35
    Steer = 90 + (900 * yCenter(5,1));
elseif roi(1,2) <= 0.4
    Steer = 90 + (900 * yCenter(10,1));
else
    Steer = 90 + (900 * yCenter(15,1));
end
SteerLog(c,1) = Steer;

```

Figure 111 – MATLAB Code For Steering Control Operation to Calculation & Steering Value Log

Using a ‘if’ statement, and a logical expression asking if the absolute value of the new calculated steering value, minus the old steering value, is greater than 10. If this expression was met, then an average over the previous steering iteration was ascertained by adding the new “Steer” value and the “OldSteer” value, dividing by ‘2’, for the average. This was found to be incredibly effective at neutralising erratic steering commands and creating a smoother journey for the ego vehicle. This can be seen within Figure 112.

```

% Cancel Out Erratic Steering Commands For Smoothness
if abs(Steer - OldSteer) > 10
    Steer = (Steer + OldSteer) / 2;
    SteerLog(c,1) = Steer;
end

if 0 < Steer && Steer < 180
    writePosition(s,Steer)
elseif Steer < 0 || Steer > 180
    Steer = OldSteer
    writePosition(s,Steer)
end

```

Figure 112 – MATLAB Code For Steering Control Operation to Neutralise Adverse Steering Values

The function ‘writePosition’ communicates with the servo motors onboard the ego vehicle, referencing the respective variable for either the steer or driving motor and the desired angle between 0° and 180°, where 90° is the central ‘off’ position. Both motors operate within these parameters, for the steering servo, 180° steers the vehicle left and 0° steers to the right, whereas with the driving motor, 180° applies forward velocity, 0° applies reverse motion to the vehicle.

Now by applying the “*Steer*” value to the steering motor using the ‘*writePosition*’ function, the vehicle can be directed correctly. Sadly, when the vehicle is in motion, the ‘noise’ previously discussed that occurs on the “*BW*” imagery, causes the parabolic lanes to be incorrectly placed on the road, making the steering experience incredible amounts of ‘unexpected course deviations’. When this happens, the “*Steer*” value applied to the motor, exceeds either 0° or 180° , because this is outside of the servo’s parameters, MATLAB encounters an error causing the script’s operation to halt.

To counteract this, another ‘*if*’ statement was brought into effect, seen in Figure 112, if the steering was between the values of 0° and 180° , that value is written to the motor, but if the “*Steer*” value is outside of these parameters, the “*OldSteer*” value is used instead, ensuring that the vehicle continues in the same direction calculated from the previous iteration, minimising the adverse effects.

8.2. Appendix B – Object Detection

It is common practice within automotive companies striving to achieve level 5 autonomy, to integrate some form of Artificial Intelligence (AI) that can be created and trained to recognise obstructions or potential dangers in front of the ego vehicle. This data can be passed on to other relative control systems that could build a three-dimensional image of the surrounding environment then classifying known or predicted labels of any noteworthy or potentially hazardous objects. (Vm, 2019)

The system can learn and understand when events occurring around the ego vehicle, build up from no threat, to instant need for an emergency brake or avoidance procedure, aiding in its ability to prevent a collision. (Brownlee, 2019)

To enable this system to be as accurate as possible, object detection was implemented to detect objects in view of the front mounted camera. By using a Convolutional Neural Network (CNN), an image can be classified against known ground truth. A course provided by Mathworks Academy (Mathworks, 2019) was undertaken to gain knowledge of the functions and the ability to construct a custom detector.

“To confirm or validate directly (information or data derived indirectly), especially (in remote sensing) by direct observation on the ground, rather than by interpretation of remotely obtained data; to make observations of (land, an area, etc.) directly on the ground, especially in order to confirm or validate data obtained indirectly.” (Lexico, 2020)

There are two types of CNN's, an image classifier and an object detector. A classifier takes an input image, detects objects within the image, and returns the labels it predicts fits best to those objects. Whereas, an object detector, identifies objects within an image, creates boundary (bounding) boxes around the regions containing those objects, and predicts the best fitting label for those objects, annotating both the bounding box and the corresponding label onto the original image.

For this project, an object detector would be the best fitting neural network to fulfil the purpose of its implementation. There are three types of object detectors; Regions with Convolutional Neural Networks (R-CNN), Fast R-CNN and Faster R-CNN, each with their own pro's and con's. A comparison of accuracy vs training time is seen in Figure 113 between different R-CNN models and structures.

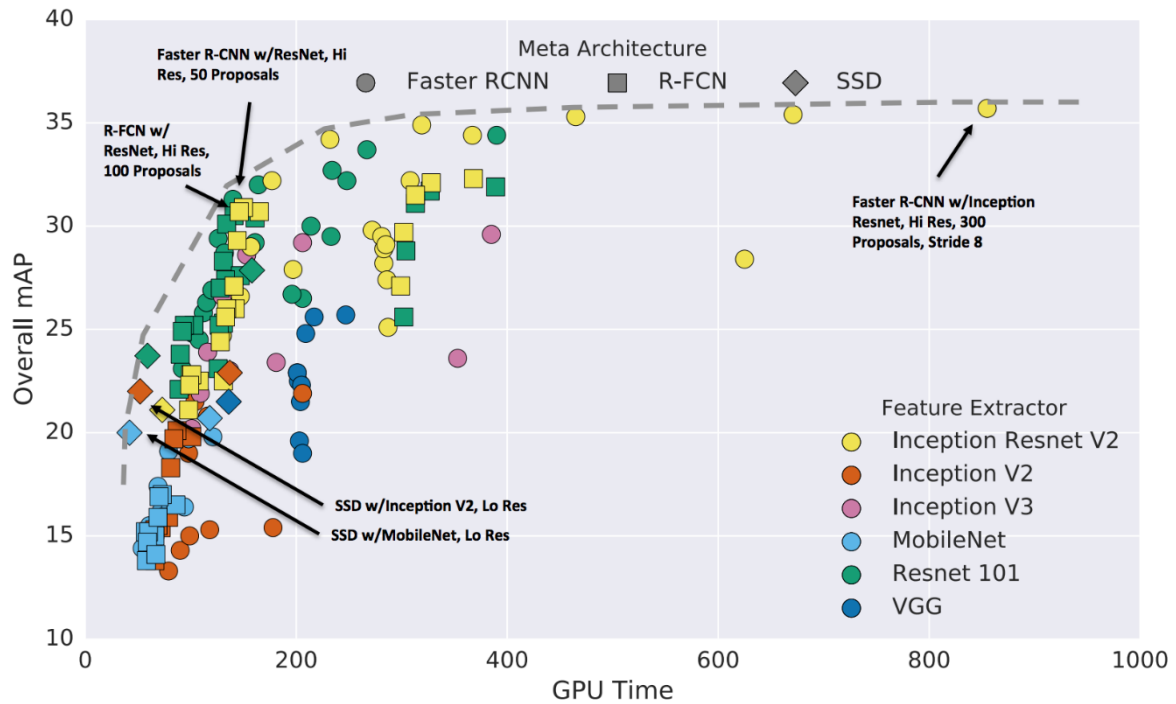


Figure 113 – Comparison of Accuracy (Mean Average Precision (mAP)) vs Training Time In milliseconds (GPU Time) (Hui, 2018)

The other main comparing feature between each R-CNN is the computation time to detect the objects within an image. Along with accuracy of the network, this was the main attribute and deciding factor to select the most appropriate R-CNN. To provide proof of concept and that this was a viable option to pursue, the basic R-CNN was selected and trained first.

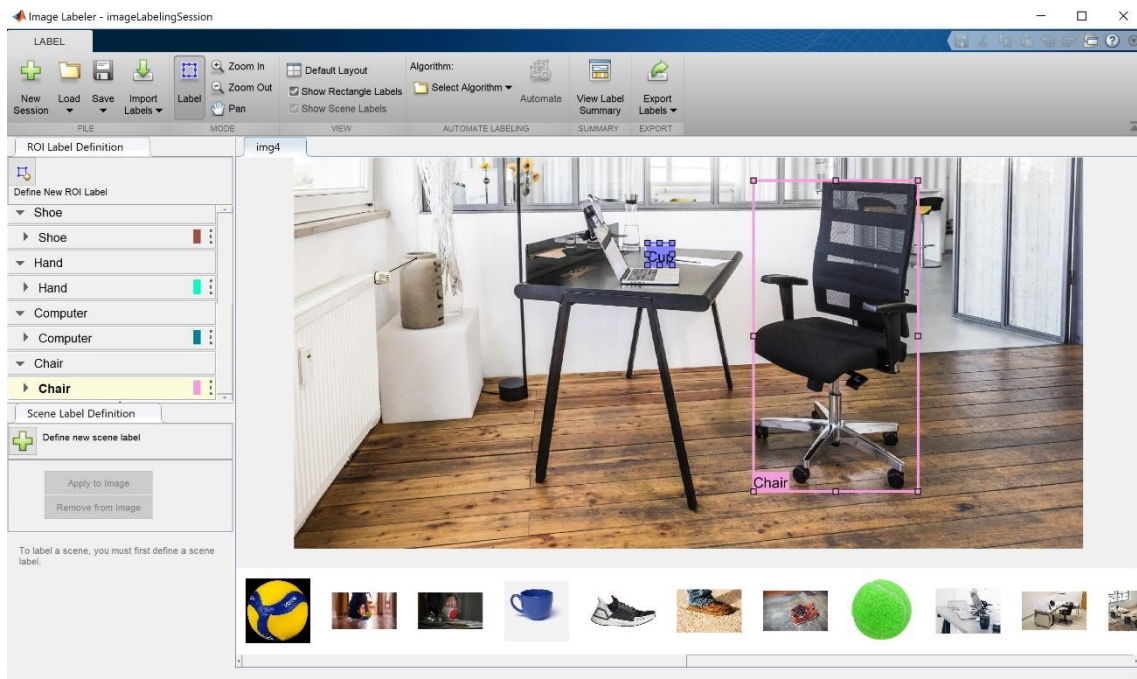


Figure 114 – Applying Ground Truth Using The Image Labeller Application Within MATLAB

To begin, 16 images were gathered from online sources, then using the image labeller application supplied with MATLAB, ground truth was applied to the images to train an object detector (Figure 114). Categories were selected based on the objects the ego vehicle would encounter on a regular basis; Ball, Chair, Computer, Cup, Shoe and Hand.

```
*****
Training an R-CNN Object Detector for the following object classes:

* Ball
* Cup
* Shoe
* Hand
* Computer
* Chair

--> Extracting region proposals from 16 training images...done.
--> Training a neural network to classify objects in training data...

Training on single CPU.
Initializing input data normalization.
=====
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Base Learning |
|       |          | (hh:mm:ss)  | Accuracy   | Loss        | Rate          |
|=====|=====|=====|=====|=====|=====|
|   1   |    1    | 00:00:38    | 17.19%     | 2.2819      | 1.0000e-05    |
|   13  |   50    | 00:21:52    | 89.60%     | 0.2976      | 1.0000e-05    |
|   20  |   80    | 00:33:29    | 92.97%     | 0.2731      | 1.0000e-05    |
|=====|=====|=====|=====|=====|=====|

Network training complete.

--> Training bounding box regression models for each object class...100.00%...done.

Detector training complete.
*****
```

Figure 115 – Basic R-CNN Training On A CPU With 16 Training Images

The detection results proved the concept well. The detections took longer than anticipated, where depending on the image, the computation time for detection could take up to 30 seconds. Should this be implemented into an autonomous vehicle, this would not be efficient. The other issue is, although the objects were correctly categorised, chair objects proved problematic, displayed as two objects, which in turn, increased computation time.



Figure 116 – Images Used Within The R-CNN Detector Annotated With The Predicted Labels

To improve the computational time taken, more images were used to train the detector. This time 36 images were used and the initial learning rate, was increased from ' $1 * 10^{-5}$ ' to ' $1 * 10^{-4}$ ' to reduce the time taken to train the network and to determine how it would affect the accuracy.

"If the learning rate is too high, then training might reach a suboptimal result or diverge."
(Mathworks, 2020)



Figure 117 – Images The R-CNN Had Been Trained On

Once training had completed, it was observed that the R-CNN performed better with images it had been train on than those it had never seen before, and the image size directly impacted the accuracy (Figure 117).

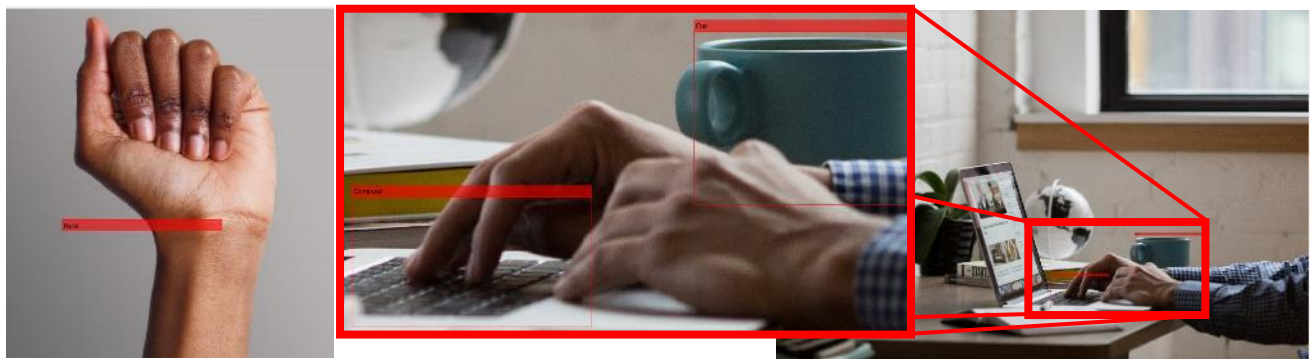


Figure 118 – Images The R-CNN Had Never Seen Before

The calculation times for detections were still too long to be considered adequate during a live feed with an average computational time of 15 seconds and several images were unable to be detected. Another attempt to reduce calculation time was to resize the images prior to detection, regular CNN's required images to be exactly 227 x 227 pixels in size, so the function '*imresize*' was used to achieve this. The network was retrained using the images it had never seen before.

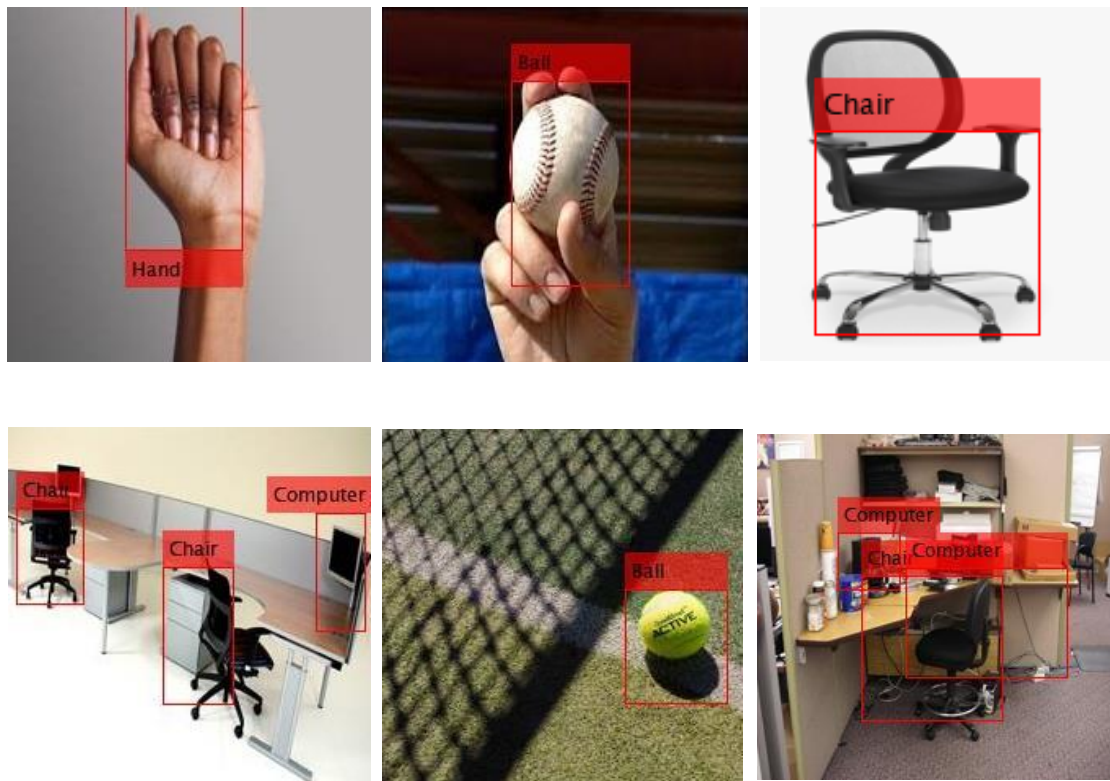


Figure 119 – Images The R-CNN Has Seen (Top), Images the R-CNN Has Never Seen Before (Bottom)

The computation time had decreased to an average of 2 seconds (Figure 120), this was closer to what was expected and could be adequate for its purpose but not during a live camera feed. The detection rate for the images had decreased to less than when at their original size.

To counteract the loss of detections, the images were resized to 480 x 640, this is the standard image size used for the Pi camera throughout the lane detection section. When this is used as an input, the images wouldn't require resizing from their original size, which may improve the success rate of detections. The network was not retrained for this for an accurate comparison.

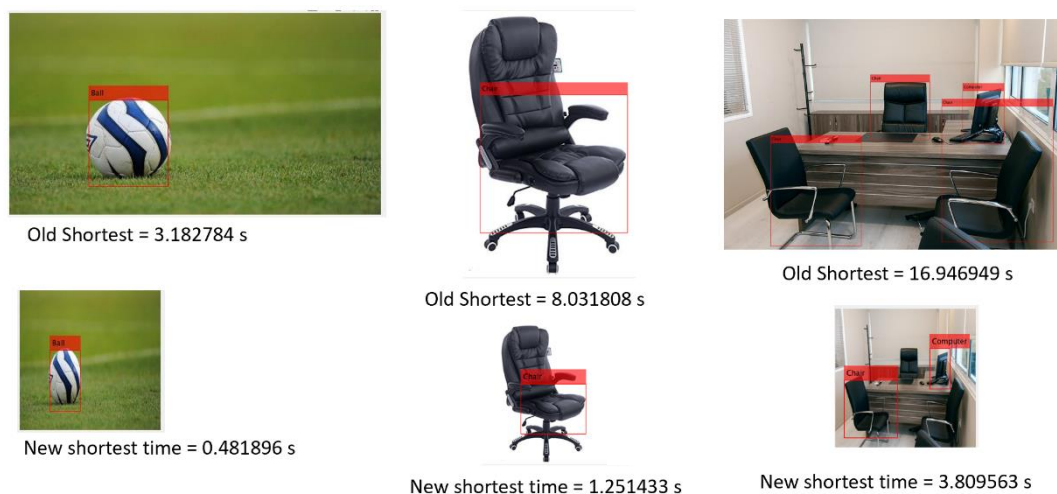


Figure 120 – Comparison Of Computation Time Between Original Size And 227 x 227 Resized Images



Figure 121 – Comparison Of Computation Time Between Original Size and 480 x 640 Resized Images

Figure 121 displays the computation time for detecting objects within images that the detector has already seen. When compared to the time taken for the 227 x 227 image sizes, they had exponentially increased during the 480 x 640 image size experiment, where most had increased past the original image size computational time.

More images were able to be detected than through the previous iterations, but the time taken to achieve these detections were not suitable for any application, thus, the network was retrained with 58 images to improve both the computation time and detection rate.



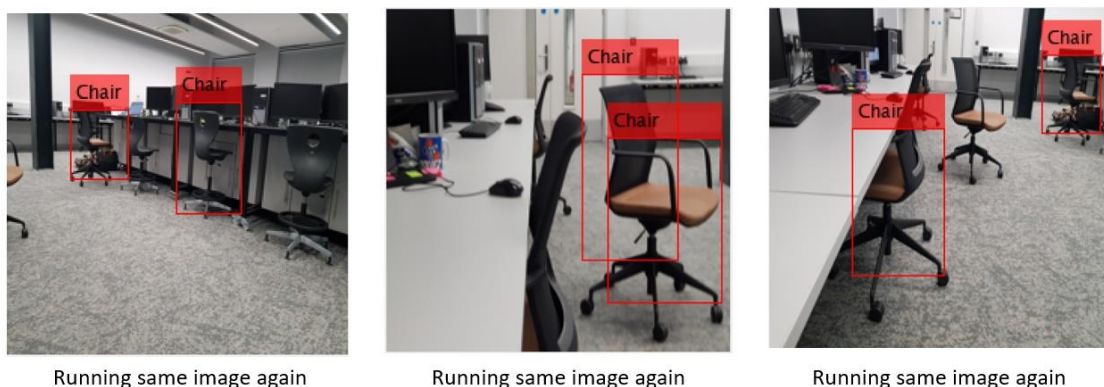
Figure 122 – Detections Of Images The Network Has Never Seen Before Using 480 x 640 Image Size

Even during busy images, the detection rate had increased, but the computation time had not. Although there were still images that could not be detected, this number was less than previous. The trade-off between the detection of images and the time taken to compute them was not acceptable. 227×227 was deemed to be the most appropriate for the task of implementing this into the ego vehicle to reduce the time taken.



Figure 123 – Detections Of Images The Network Has Never Seen Before Using 227×227 Image Size

96 images were used to train/retrain the R-CNN, alongside the 'imresize' function as this previously reduced the computation time dramatically, while accuracy decreased. Although in practice, this was not evident. As seen in Figure 123 and Figure 124, the accuracy wasn't as bad as previously noticed, but the computation time differed greatly.



Elapsed time is 35.472925 seconds.
Elapsed time is 18.819043 seconds.
Elapsed time is 13.228310 seconds.
Elapsed time is 13.157916 seconds.

Elapsed time is 16.558976 seconds.
Elapsed time is 15.884227 seconds.
Elapsed time is 16.324960 seconds.
Elapsed time is 15.455178 seconds.

Elapsed time is 17.131079 seconds.
Elapsed time is 13.996590 seconds.
Elapsed time is 10.996361 seconds.
Elapsed time is 11.017063 seconds.

Figure 124 – Computation Time Of 227×227 Resized Images Ran Through The R-CNN Repeatedly

The Raspberry Pi was powered up, with the Pi camera used as an input for the detector to judge how efficient the system would be as a whole in its current state.



Figure 125 – Initial Images Captured With The Raspberry Pi Camera And Processed Through the Detector

As demonstrated in Figure 125 and Figure 126, the computation times varied significantly, this was disappointing, even more so when it could not classify a hand and took several rotations before it could detect a shoe. It detected the majority of objects within the images that it was trained to find, but it is clear to see the range of times before the objects were detected.

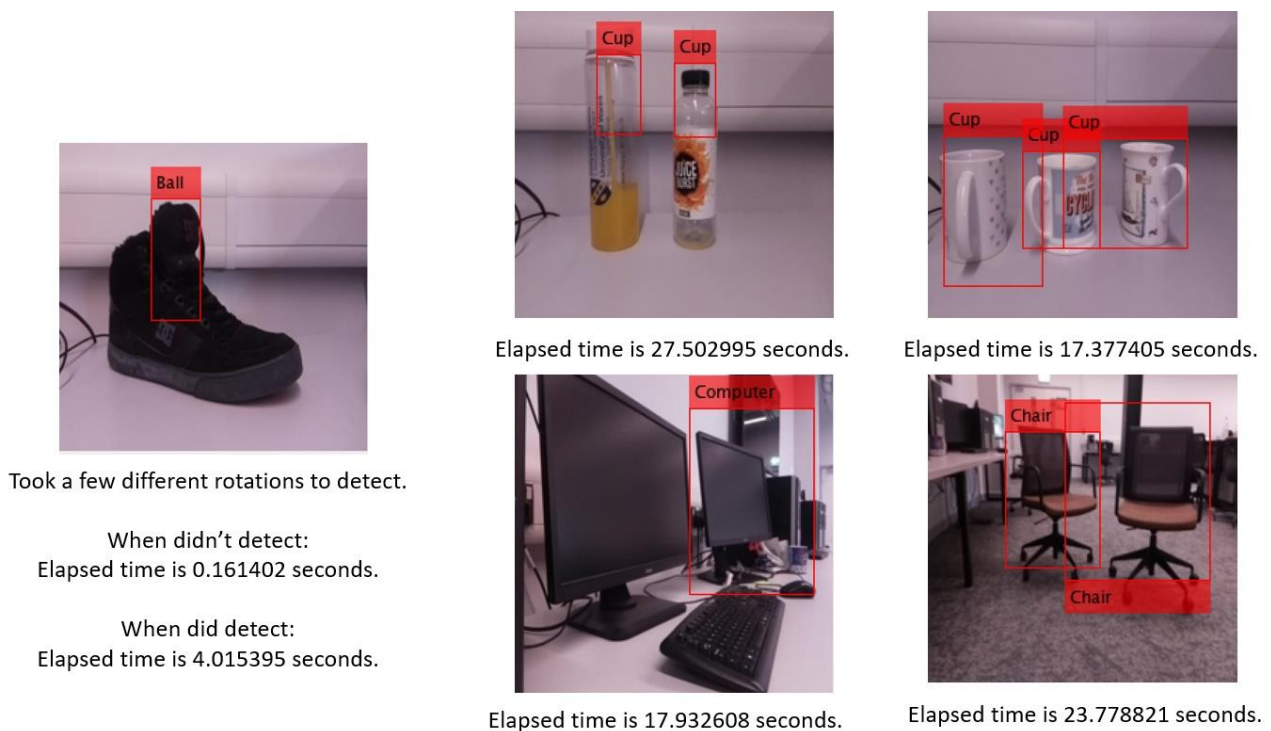


Figure 126 – Images Captured With the Raspberry Pi Camera And Process Through The R-CNN

A 'while' loop was created in the same manner the lane detection works in, where a snapshot was taken from the Pi camera, passed through to the detector, with the annotated image being returned in a 'figure' window, as before.

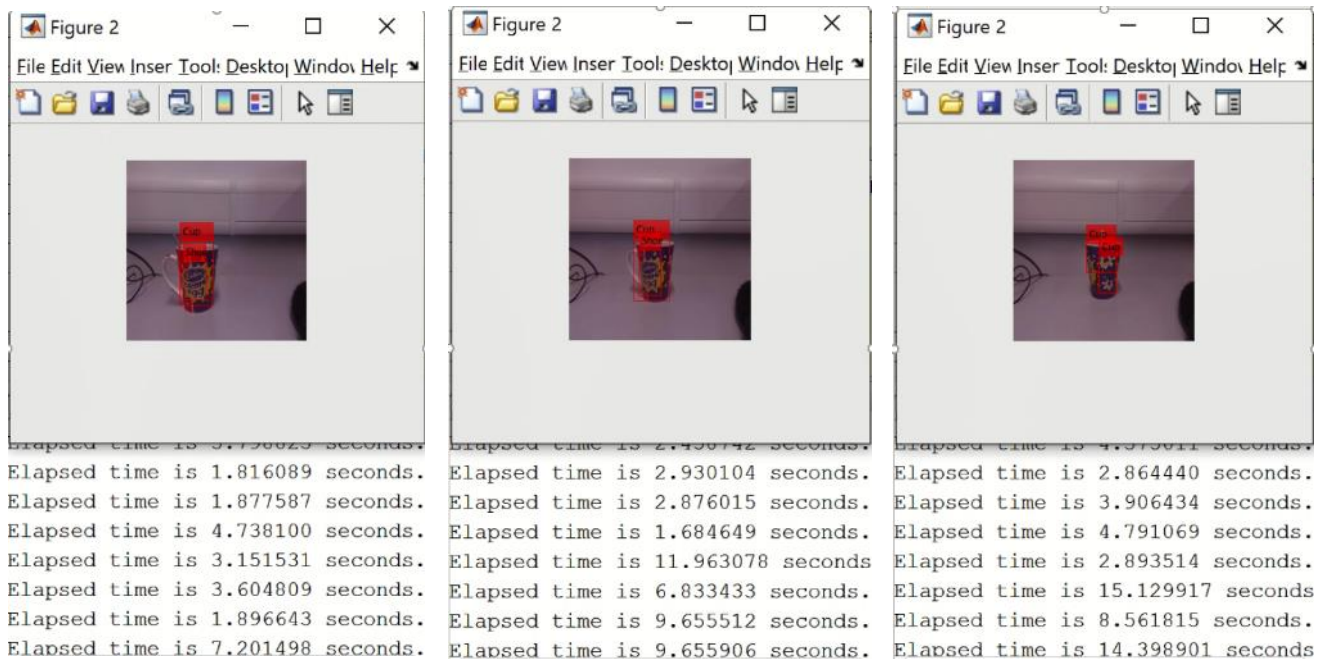


Figure 127 – Live Feed Of Images From The Raspberry Pi Camera Passed Through To The Detector

Figure 127 shows three points throughout a recorded video of the live feed from the Pi camera where the cup was rotated around to see the effect. It is evident that the detector is capable of a computational time of 1.68 seconds, but this is outweighed by the dramatic jump to 15.13 seconds. It is due to this inconsistency that the basic R-CNN was determined to not be a viable option for the implementation to the ego vehicle, although it did a good job in proving that the overall concept was feasible.

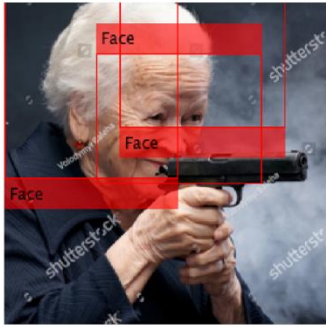
To increase productivity, a Faster R-CNN detector was created and trained. Unfortunately, unlike before, where a 'miniBatch' of images, used for training, can include images of multiple sizes with the basic R-CNN, a Faster R-CNN can only use a 'miniBatch' of one, should this be the case.

To begin with, the detector was trained on all 121 images that had been accumulated to this point. This training process has four stages. First, a Region Proposal Network (RPN) was trained on the images defined within the ground truth labeller, this generates region boxes, called anchors, within an image. The RPN proposes the anchors most likely to contain an object, based on the possibility of being either background or foreground and uses this to refine the anchor, generating a boundary box around the object and compares this to the ground truth.

The second step involves training a Fast R-CNN network using the RPN previously created. This stage is the same as training a basic R-CNN but by using refined ground truth model. Once finished, the RPN is re-trained using the Fast R-CNN from step 2 to adapt the weights, which aid the network with classifications of objects, within the network and ensure correct detections are made. The final step is to then re-train the Fast R-CNN using the updated RPN, improving the accuracy and speed of the network. It is these final two steps that distinguish a Fast R-CNN from a Faster R-CNN.

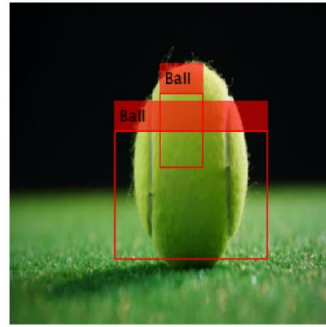
Figure 128 – Detections Of Images By The Faster R-CNN Detector From Online Sources And Pi Camera With Their Computation Time

Overall, this was a huge improvement over the basic R-CNN, the computational times were a fraction of those previously, compared within and the accuracy



F-RCNN:
Elapsed time is 0.631801 seconds.

RCNN:
Elapsed time is 6.259978 seconds.



F-RCNN:
Elapsed time is 0.603257 seconds.

RCNN:
Elapsed time is 2.820298 seconds.



F-RCNN:
Elapsed time is 0.610602 seconds.

RCNN:
Elapsed time is 6.747460 seconds.

Figure 129 – Comparison Of Images Processed Through Both The Basic R-CNN And The Faster R-CNN Detectors

This is more like what was expected from the detector at the start of this venture. Even if it mistakenly grouped objects together, it still correctly identified them and fairly quickly compared to previous iterations of up to 30 seconds.

Unfortunately, when the rest of the automated driving system is active, the time taken per iteration can be up to a second or more, this is not a viable option within the real world to react to situations that can arise very quickly. If the system could manage at least ten Frames Per Second (FPS), it would be more applicable, but unless this could happen, it would do more harm than good by reducing its reactive capability too greatly.

8.3. Appendix C – Parts List

Table 2 – Hardware Used For This Project (Anderson, 2020)

Hardware	Price	Source
Creality Ender 3 3D Printer	£200	Source
Raspberry Pi 3 B+	£36	Source
Raspberry PiNoir Camera Module V2	£24.4	Source
Sharp GP2Y0A02YK0F Infrared sensor	£20	Source
HC-SR04 Ultrasonic sensor	£2	Source
MCP3008 ADC	£2.75	Source
MG90S Metal Gear Micro Servo	£5	Source
LBA2040-22T 2600KV Brushless Motor & 25A ESC	£20.90	Source
45C 4000mAh LiPo 2S 7.4V Battery Pack	£26.19	Source
Samsung INR18650-30Q Lithium-ion Cells	£12.98	eBay – Product Since Removed By Seller “ Moon-Love-Mars ”
AA Battery Contact Leaf Spring	£2.80	Source
Power Bank Module Board	£3	Source
15A Capable Wire	£1	Swansea Indoor Market Electrical Stall – “ Tom Whitehouse Ltd ”
Jumper/Dupont Wires – Male to Male, Female to Male, Female to Female	£5.99	Source
Heat Shrink Tubing Sleeves	£6.99	Source
Mini Breadboard	£5.99	Source
Breadboard	£3	Source
180° Fisheye Lens		
16mm x 5mm x 5mm 625ZZ Ball Bearings	£5.99	Source
Multimeter	£9.99	Source

Table 3 – Software Used For This Project (Anderson, 2020)

Software	Source
Mathworks MATLAB	https://uk.mathworks.com/products/matlab.html?requestedDomain=
MATLAB Automated Driving Toolbox	https://www.mathworks.com/products/automated-driving.html
MATLAB Deep Learning Toolbox	https://www.mathworks.com/products/deep-learning.html
MATLAB Hardware Support For Raspberry Pi	https://www.mathworks.com/help/supportpkg/raspberrypiio/index.html
MATLAB AlexNet Infrastructure	https://www.mathworks.com/help/deeplearning/ref/alexnet.html
Python	https://www.python.org/

8.4. Appendix D – Samsung INR18650-30Q Lithium-ion Cell Specifications

Specification

PROPRIETARY AND
CONFIDENTIAL

Type		Spec.(Tentative)	Typical INR18650-30Q
Chemistry		NCA	NCA
Dimension (mm)	Diameter	18.33 ± 0.07	18.33 ± 0.07
	Height	64.85 ± 0.15	64.85 ± 0.15
Weight (g)		Max. 48.0	45.6
Initial IR (mΩ AC 1kHz)		≤ 18	13.13 ± 2
Initial IR (mΩ DC (10A-1A))		≤ 30	19.94 ± 2
Nominal Voltage (V)		3.60	3.61
Charge Method (100mA cut-off)		CC-CV (4.2±0.05V)	CC-CV (4.2±0.05V)
Charge Time	Standard (min), 0.5C	180min	134min
	Rapid (min), 4A	70min	68min
Charge Current	Standard current (A)	1.5	1.5
	Max. current (A)	4.0	4.0
Discharge	End voltage (V)	2.5	2.5
	Max. cont. current (A)	15	15
Rated discharge Capacity	Standard (mAh) (0.2C)	3,000	3,040
	rated (mAh) (10A)	2,700	2,983

3/17

SAMSUNG SDI 

Figure 130 – Samsung INR18650-30Q Lithium-ion Cell Specifications (Samsung SDI Co., Ltd. - Energy Business Division, 2014)

1.0. Scope

This product specification has been prepared to specify the rechargeable lithium-ion cell ('cell') to be supplied to the customer by Samsung SDI Co., Ltd.

2.0. Description and model

2.1 Description lithium-ion rechargeable cell

2.2 Model name INR18650-30Q

3.0. Nominal specifications

Item	Specification
3.1 Minimum discharge capacity	2,950mAh Charge: 1.50A, 4.20V, CCCV 150mA cut-off, Discharge: 0.2C, 2.5V discharge cut-off
3.2 Nominal voltage	3.6V
3.3 Standard charge	CCCV, 1.50A, 4.20 ± 0.05 V, 150mA cut-off
3.4 Rated charge	CCCV, 4A, 4.20 ± 0.05 V, 100mA cut-off
3.6 Charging time	Standard charge : 180min / 150mA cut-off Rated charge: 70min (at 25 °C) / 100mA cut-off
3.7 Max. continuous discharge (Continuous)	15A(at 25 °C), 60% at 250 cycle
3.8 Discharge cut-off voltage End of discharge	2.5V
3.9 Cell weight	48.0g max
3.10 Cell dimension	Height : Max 65.0mm Diameter : Max 18.4mm
3.11 Operating temperature (surface temperature)	Charge : 0 to 50 °C (recommended recharge release < 45 °C) Discharge: -20 to 75 °C (recommended re-discharge release < 60 °C)
3.12 Storage temperature (Recovery 90% after storage)	1.5 year -30~25 °C (1*) 3 months -30~45 °C (1*) 1 month -30~60 °C (1*)

Note (1): If the cell is kept as ex-factory status (40±5% SOC, 25 °C),
the capacity recovery rate is more than 90% of 10A discharge capacity
100% is 2,900mAh at 23 °C with SOC 100% after formation.

Figure 131 – Samsung INR18650-30Q Lithium-ion Cell Specifications (Samsung SDI Co., Ltd. - Energy Business Division, 2015)

8.5. Appendix E – MATLAB Script To Operate The Ego Vehicle

```
%% Connections to Pi, CameraBoard and Servos

% Connection To Raspberry Pi
rpi = raspi('[Local Device Address]','[Device Name]','[Device Password]');
% Connection To Pi Camera
picam = cameraboard(rpi,'Resolution','640x480','FrameRate',30);
% Rotate Camera If Upside Down
picam.Rotation = 180;
% Connection To Driving Motor
d = servo(rpi,12);
% Configure GPIO Mode To PWM
modeD = configurePin(rpi,12,'PWM');
% Connection To Steering Motor
s = servo(rpi,18);
% Configure GPIO Mode To PWM
modeS = configurePin(rpi,18,'PWM');
% Enable SPI On Raspberry Pi
enableSPI(rpi)
% Create Connection To IR Sensor
IRSensor = spidev(rpi,'CE0');

%% Get Sample Frames

for k = 1:20
    % Snapshot From Pi Camera
    img = snapshot(picam);
    figure(1000)
    % Display Snapshot
    imshow(img)
end

%% Centralise Servos

writePosition(d,90);
writePosition(s,90);

%% Camera Setup

% Birds Eye Parameters
roi = [0.25 0.6 -0.2 0.2];
% Region Of Interest [BottomX, TopX, LeftY, RightY]
imgSize = [NaN 300];
% Define BE Size [NoDefinedX, 300PixelsY]

% Camera Intrinsics & Sensor Setup0
FL = [500 500];
% Define Focal Length
PP = [320 240];
% Define Principle Point
IS = [480 640];
% Define Image Size

intrinsics = cameraIntrinsics(FL,PP,IS);
% Combine Intrinsics
```

```

height = 0.1415;
% Define Camera Height From Ground
pitch = 11.25;
% Define Camera Pitch
roll = 0;
% Define Camera Roll

% Create MonoCamera Sensor
sensor = monoCamera(intrinsics,height,'pitch',pitch,'roll',roll);

c = 1;
Drive = 90;
BrakeLog = Drive;
DriveLog = Drive;
Steer = 90;
SteerLog = Steer;
Continue = 1;
EmergencyStop = 0;
Slow = 0;
Stop = 0;
FurtherPotentialDanger = 0;
DistanceLog = 0;
FPS = 0;
Video = 0;

% Setup Player For Loop
player = vision.DeployableVideoPlayer;

% Running loop
while 1 > 0

    tic
    OldDrive = Drive;
    OldSteer = Steer;

    % Snapshot From Pi Camera
    frame = snapshot(picam);
    figure(1)
    imshow(frame,'InitialMagnification','fit')

    % Create Birds Eye View
    view = birdsEyeView(sensor,roi,imgSize);

    % Apply Birds Eye View
    IBE = transformImage(view,frame);
    figure(10)
    imshow(IBE,'InitialMagnification','fit')

    % Segment Lane Markers In Black & White
    BW = segmentLaneMarkerRidge(rgb2gray(IBE),view,0.01,'Sensitivity',0.35);
    figure(20)
    imshow(BW,'InitialMagnification','fit')

    % Lane Marker Points For X & Y Dimensions
    [y,x] = find(BW);

    % BE -> Vehicle (Transform Image Co-ordinates To Vehicle Co-ordinates)
    pointsVCS = imageToVehicle(view,[x,y]);

```

```

% Create Parabolic Lane Boundarie Model With Vehicle Co-ordinate System
lanes = findParabolicLaneBoundaries(pointsVCS,0.01);

if abs(lanes(1,1).XExtent(1,2) - lanes(1,2).XExtent(1,2)) > 0.2
    if (lanes(1,1).XExtent(1,2) > lanes(1,2).XExtent(1,2))
        roi(1,2) = lanes(1,2).XExtent(1,2);
    elseif (lanes(1,1).XExtent(1,2) < lanes(1,2).XExtent(1,2))
        roi(1,2) = lanes(1,1).XExtent(1,2);
    end
    if roi(1,2) < 0.4
        Stop = 1;
        text_str = ['Distance To Nearest Object: ' num2str(Distance,
'%4.2f'), ' mm', ' ! Stop !'];
    end
elseif lanes(1,1).XExtent(1,2) || lanes(1,2).XExtent(1,2) < 0.6
    roi(1,2) = abs(lanes(1,1).XExtent(1,2) + lanes(1,2).XExtent(1,2))/2;
    if roi(1,2) < 0.55
        roi(1,2) = roi(1,2)*1.05;
        if roi(1,2) >= 0.8
            roi(1,2) = 0.8;
        end
    end
end
if roi(1,2) < 0.5
    if roi(1,1) < 0.25
        roi(1,1) = roi(1,1) * 0.9;
    end
    if roi(1,1) <= 0.2
        Stop = 1;
        writePosition(d,90)
        text_str = ['Distance To Nearest Object: ' num2str(Distance,
'%4.2f'), ' mm', ' ! Stop !'];
    end
    if roi(1,1) <= 0.1
        roi(1,1) = 0.1;
    end
elseif roi(1,2) >= 0.55
    roi(1,1) = roi(1,1) * 1.1;
    if roi(1,1) >= 0.25
        roi(1,1) = 0.25;
    end
end

if (lanes(1,1).XExtent(1,2) > lanes(1,2).XExtent(1,2))
    MaxWorldLength = lanes(1,2).XExtent(1,2);
elseif (lanes(1,1).XExtent(1,2) < lanes(1,2).XExtent(1,2))
    MaxWorldLength = lanes(1,1).XExtent(1,2);
end

% Test Model For Visual Inspection On Plot (Comment Out Plot For Speed)
xIn = roi(1,1):0.01:MaxWorldLength;
yOut = computeBoundaryModel(lanes,xIn);
figure(40)
plot(yOut,xIn,'k')
xlim([-0.2 0.2])
set(gca,'xdir','reverse')

if isempty(yOut)
    player(frame)
end

```

```

else
    % Defining Left & Right Lanes
    horz = yOut(:,1);
    left = lanes(horz > 0);
    right = lanes(horz < 0);

    % Insert Left & Right Lanes Onto Birds Eye View
    lanesBE = insertLaneBoundary(IBE,left,view,
(roi(1,1)*1.2):0.01:(lanes(1,2).XExtent(1,2)*0.95),'color','red');
    lanesBE = insertLaneBoundary(lanesBE,right,view,
(roi(1,1)*1.2):0.01:(lanes(1,1).XExtent(1,2)*0.95),'color','green');
    figure(50)
    imshow(lanesBE,'InitialMagnification','fit')

    % Segment Lanes For Control Of Car
    xWorld = xIn';
    yLeft = computeBoundaryModel(lanes(1,2),xWorld);
    yRight = computeBoundaryModel(lanes(1,1),xWorld);

    % Find Centre of Lane
    yCenter = (yLeft + yRight)/2;

    % Plot Lane Markers and Lane Center / Path Of Ego Vehicle
    bep = birdsEyePlot('XLimits',[0 roi(1,2)],'YLimits',[roi(1,3)
roi(1,4)]);
    lanePlotter = laneBoundaryPlotter(bep,'DisplayName','Lane
plotLaneBoundary(lanePlotter,{[xWorld,yLeft],[xWorld,yRight]});
    egoPathPlotter = pathPlotter(bep,'DisplayName','Ego Vehicle Path');
    plotPath(egoPathPlotter,{[xWorld,yCenter]});

    % Insert Lane Boundaries Into Camera Frame
    lanesCam = insertLaneBoundary(frame,lanes,sensor,
(roi(1,1)*1.2):0.01:(MaxWorldLength*0.95),'color','red');
    figure(50)
    imshow(frame,'InitialMagnification','fit')

    % Steering Control
    % 180 = left, 0 = right
    if roi(1,2) <= 0.25
        Steer = 90 + (900 * yCenter(1,1));
    elseif roi(1,2) <= 0.35
        Steer = 90 + (900 * yCenter(5,1));
    elseif roi(1,2) <= 0.4
        Steer = 90 + (900 * yCenter(10,1));
    elseif roi(1,2) <= 0.45
        Steer = 90 + (900 * yCenter(12,1));
    else
        Steer = OldSteer;
    end

    SteerLog(c,1) = Steer;

    % Cancel Out Erratic Steering Commands For Smoothness
    if abs(Steer - OldSteer) > 10
        Steer = (Steer + OldSteer) / 2;
        SteerLog(c,1) = Steer;
    end

```

```

if 0 < Steer && Steer < 180
    writePosition(s,Steer)
elseif Steer < 0 || Steer > 180
    Steer = OldSteer;
    writePosition(s,Steer)
end

% Read Voltage From MCP3008 Connected To IR Sensor
Data = uint16(writeRead(IRSensor,[1, bin2dec('10000000'), 0]));
Highbits = bitand(Data(2), bin2dec('11'));
Voltage = double(bitxor(bitshift(Highbits, 8), Data(3)));
Voltage = (3.3/1024) * Voltage;

% Calculate Distance To Nearest Object From Voltage (mm)
Distance = (16.2537 * Voltage^4 - 129.893 * Voltage^3 + 382.268 *
Voltage^2 - 512.611 * Voltage + 301.439) * 10;
DistanceLog(c,1) = Distance;

% Apply Adequate Warning Based On Distance To Nearest Object (mm)
if 550 < Distance && Distance < 1000
    Distance;
    Slow = 1;
    text_str = ['Distance To Nearest Object: ' num2str(Distance,
'%4.2f'), ' mm', ' ! Slow Down !'];
elseif 400 < Distance && Distance < 550
    Distance;
    Stop = 1;
    text_str = ['Distance To Nearest Object: ' num2str(Distance,
'%4.2f'), ' mm', ' ! Stop !'];
elseif Distance < 400
    Distance;
    EmergencyStop = 1;
    text_str = ['Distance To Nearest Object: ' num2str(Distance,
'%4.2f'), ' mm', ' !!! Emergency Stop !!!'];
else
    Distance;
    Continue = 1;
    text_str = ['Distance To Nearest Object: ' num2str(Distance,
'%4.2f'), ' mm'];
end

% Annotate Frame With Distance Measurement And
position = [320 480];
lanesCam = insertText(lanesCam,position,text_str,'FontSize',18,
'AnchorPoint','CenterBottom');

% Take Appropriate Action Based On Distance Measurements
if Slow == 1
    if Drive > 110
        Drive = Drive / 1.005;
    end
    if Drive <= 110
        Drive = 110;
    end
    BrakeLog(c,1) = Drive;
    DriveLog(c,1) = Drive;
    writePosition(d,Drive)
    Slow = 0;

```

```

elseif Stop == 1
    if Drive > 90
        Drive = 60;
    end
    writePosition(d,Drive)
    BrakeLog(c,1) = Drive;
    Drive = 90;
    DriveLog(c,1) = Drive;
    writePosition(d,Drive)
    FurtherPotentialDanger = 1;
    Stop = 0;
elseif EmergencyStop == 1
    if Distance < 100 && Drive > 90
        Drive = 0;
    elseif Distance < 150 && Drive > 90
        Drive = 20;
    elseif Distance < 200 && Drive > 90
        Drive = 30;
    elseif Distance < 250 && Drive > 90
        Drive = 40;
    elseif Distance < 300 && Drive > 90
        Drive = 50;
    elseif Distance < 350 && Drive > 90
        Drive = 60;
    elseif Distance < 400 && Drive > 90
        Drive = 70;
    elseif Distance < 400 && Drive == 90
        Drive = 90;
    end
    writePosition(d,Drive)
    BrakeLog(c,1) = Drive;
    Drive = 90;
    DriveLog(c,1) = Drive;
    writePosition(d,Drive)
    FurtherPotentialDanger = 1;
    EmergencyStop = 0;
elseif Continue == 1
    if FurtherPotentialDanger == 1 && OldDrive == 90
        Drive = 90;
    elseif FurtherPotentialDanger == 0 && Drive >= 110 && Drive <130
        Drive = Drive * 1.001;
    elseif FurtherPotentialDanger == 0 && Drive > 130
        Drive = 130;
    elseif FurtherPotentialDanger == 0 && Drive < 110
        Drive = 110;
    end
    writePosition(d,Drive)
    DriveLog(c,1) = Drive;
    Continue = 0;
end

if FurtherPotentialDanger == 0
    Drive = Drive;
elseif DriveLog(c,1) < 90
    Drive = 90;
    FurtherPotentialDanger = 1;

```

```

elseif Distance > 550
    if length(DistanceLog) > 30
        Detections = DistanceLog(c-30:c,1);
    elseif length(DistanceLog) > 20
        Detections = DistanceLog(c-20:c,1) < 550;
    elseif length(DistanceLog) > 10
        Detections = DistanceLog(c-10:c,1) < 550;
    else
        Detections = DistanceLog(c,1) < 550;
    end
    StillDetections = Detections <= 550;
    if any(StillDetections == 1)
        Drive = 90;
        FurtherPotentialDanger = 1;
    elseif all(StillDetections == 0)
        Drive = Drive;
        FurtherPotentialDanger = 0;
    end
end
DriveLog(c,1) = Drive;
writePosition(d,Drive)

SPF = toc;
FPS = (1/SPF); % Because Frame Per Time = (1 / Time Per Frame)
position2 = [0 0];
text_str2 = ['FPS: ' num2str(FPS,'%4.2f')];
lanesCam = insertText(lanesCam,position2,text_str2,'FontSize',18,
'AnchorPoint','LeftTop');

% Display Final Frame
player(lanesCam)

% Record Frames For Later Use
% Saves To Current Folder, Otherwise Designate Directory
imwrite(lanesCam, sprintf('%i.jpg',c));

% Increase Counter
c = c + 1;
end
end

%% Centralise Servos

writePosition(d,90);
writePosition(s,90);

%% Plots Log Files

plot(BrakeLog)
plot(DistanceLog)
plot(DriveLog)
plot(SteerLog)

%% Close Figures & Shutdown Pi

close all

h = raspberrypi;
h.system('sudo shutdown -h now')

```