Modular Self-Driving and Sensor Packages for R/C Cars

A Major Qualifying Project Report:

Submitted to the Faculty of the:

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the degree of Bachelor of science

by

Thomas Kim (RBE/ME)

Anthony Marge (ECE/ME)

Joshua Rondon (ME)

Kyle Wood (ECE)

Date: May 18th, 2020

Approved by:

Professor Pradeep Radhakrishnan (RBE/ME)

Professor Kaveh Pahlavan (ECE)

Copyright Notice

Table of Contents

List of Figures

List of Tables

Abstract

Mechatronic systems are an amalgamation of mechanisms, sensors, controls, and actuators. Self-driving cars are a growing class of mechatronic systems and research into related areas has been surging. Many companies are investing large sums of money into the research and development of autonomous vehicles. The next generation of mechanical engineers must be capable of handling these developments and should be able to design systems around them. In this paper, we describe our efforts to develop a modular self-driving and sensors kit that can be integrated into any scale or remote-controlled car as a way to provide students in mechanical engineering exposure to autonomous driving and real-time data collection. Details on the related research, selection of appropriate self-driving models and sensors, testing and validation of these kits, development of user-guides for use in a course and evaluation with test users will be discussed in this paper.

Keywords: Artificial Intelligence, Remote Controlled Cars, Scale Cars, Self-Driving Scale Cars, Self-Driving Kits, Sensors, Neural Networks

Nomenclature

AI	Artificial Intelligence
AWS	Amazon Web Services
CMG	Control Moment Gyroscope
CSV	Comma Separated Value
ESC	Electronic Speed Controller
IMU	Inertial Measurement Unit
MQP	Major Qualifying Project
NN	Neural Net
PWM	Pulse-Width Modulation
RasPi	Raspberry Pi
RC	Remote Controlled
RPM	Rotations Per Minute
WPI	Worcester Polytechnic Institute

Authorship

Chapter	Primary Author(s)	Editor(s)
Introduction	Joshua Rondon	Anthony Marge
Background	Joshua Rondon	Anthony Marge, Kyle Wood
Project Goals and Requirements	Joshua Rondon, Thomas Kim	Anthony Marge, Kyle Wood
Self-Driving Implementation	Anthony Marge	Joshua Rondon
Sensors Implementation and Combining Self-Driving & Sensor Kits	Kyle Wood, Thomas Kim	Joshua Rondon
Creating Testing Mechanisms for Subcomponent Testing	Thomas Kim	Joshua Rondon
Results and Discussion	Anthony Marge, Kyle Wood	Thomas Kim
Conclusion	Anthony Marge	Kyle Wood, Thomas Kim
References	Anthony Marge	Joshua Rondon
Appendices	Thomas Kim, Joshua Rondon	Anthony Marge, Kyle Wood

Acknowledgements

We would like to thank our two advisors: Professor Pradeep Radhakrishnan and Professor Kaveh Pahlavan. Without their support and direction, we would not have been able to achieve as much as we did. We would like to thank Michael DeFrancesco, Michael Pierce, Alex Boggess, Julia Davenport, Richard Mohabir, and Zack Orbach for making the 3D-printed RC cars for us to test our sensor and self-driving packages on. We would also like to thank Jessie Kablik and Brian King for testing out our user guides. We would also like to thank Worcester Polytechnic Institute, most importantly all of the faculty and staff who have helped with advice, planning, and purchasing of crucial parts of our MQP. Specifically, Ms. Payton Wilkins, Ms. Barbara Furhman, and Mr. Peter Hefti who all were always there to help and support us.

Executive Summary

Mechatronic systems are an amalgamation of mechanisms, sensors, controls and actuators. Self-driving cars are a growing class of mechatronic systems and research into related areas has been surging. Many companies are investing large sums of money into the research and development of autonomous vehicles. The next generation of mechanical engineers must be capable of handling these developments and should be able to design systems around them. It is therefore important to develop laboratory exercises that can augment content in existing courses in order to equip students to handle these fast-changing technological advancements. Autonomous driving involves neural network models and sensors. In order to develop laboratory exercises around this theme, it was decided to investigate the use of open source self-driving packages that could be integrated into scale (or remote-controlled) cars. Scale cars are cheaper, and the technology required is also low-cost.

At Worcester Polytechnic Institute, there is a senior-level capstone design course called Advanced Engineering Design where student teams design and develop a scale-car with a custom-gearbox and a steering linkage. The teams must test their car at the end of the seven-week course. While the mechanical design of the system may be robust, the sensors and control are all manual. Given that the students may not have enough practice driving scale-cars, the end-of-term race might not fully evaluate the car's ability to tackle varied driving conditions. So, for our project, we created a self-driving package and a sensor package that can be integrated into any scale or remote-controlled car as a way to provide students in mechanical engineering exposure to autonomous driving and real-time data collection (Figure I).

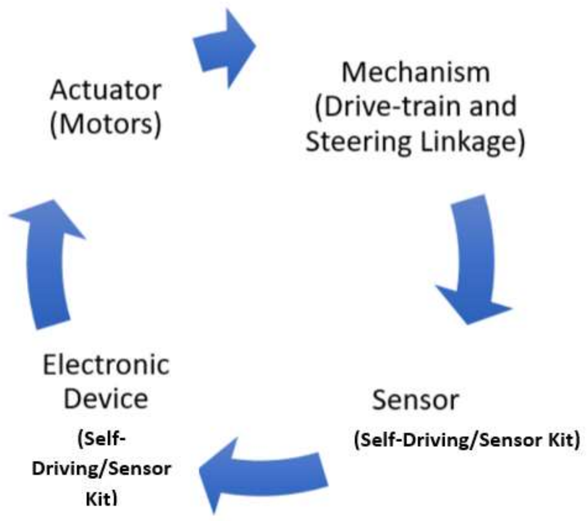


Figure I: ME4320 Mechatronic Diagram once the Self-Driving/Sensor Kits Become Integrated into the Class.

To develop a modular self-driving kit, we first had to select a commercially available self-driving platform for us to use, and we selected DonkeyCar. Next, we had to test this self-driving platform. Our first goal with the self-driving package was to have a remote controlled car drive four laps around a track without human intervention using our self-driving package, and to achieve this goal, we tested our self-diving package on seven different tracks. First, we tested the self driving package on a recommended car on simple tracks. Next, we tested the self-driving platform on a student-made car on simple tracks. Next, we tested the self-driving platform on a 3D printed car on simple tracks. Next, we tested the self-driving platform on a student-made car on complex tracks. Lastly, we tested the self-driving platform on a 3D printed car on complex tracks. With each test, we gained experience creating neural network models and learned techniques on how to make a better neural network model, which in turn improved the self-driving performance of the remote controlled cars we tested on until we were able to achieve our goal of a remote controlled car self-driving four laps around a track.

After running tests on the self-driving platform, we needed to create a user guide that is easy for students in ME 4320 and ME/RBE 4322 to follow so that they can implement our self-driving kit on a scale car to create a self-driving scale car. We created a first draft of our user guide by buying all of the components necessary to make a second self-driving kit and taking notes as we constructed the second self-driving kit. We formalized these notes to make our user guide. Next, we tested our user guide on two Mechanical Engineering students, Jesse Kablik and Brian King, so we could evaluate our guide. We used the feedback that they gave us to improve our guide, completing our development of the modular self-driving kit and user guide. The kit is now ready to be deployed in ME 4320 and ME/RBE 4322.

We had to make a sensor package and a user-guide on how to use the package. As per our objective on the modular sensor kit, we needed to determine what sensors should be used for data collection, as well as how we should go about implementing the sensors in an easy to understand and follow manner. This was accomplished with an Ardiuno Mega and RasPi; both fairly common and simple to use electronics. For our sensor package, we wanted to provide students with various sensors that could record useful data about the function of their scale cars. To do this, we had to wire various sensors into an Arduino, and then program that Arduino to send the data to a Raspberry Pi where the data could be saved to a useful filetype for later analysis.

The authors also developed a modular sensor kit that could be integrated into any scale-car to record speed of drive shafts, rotations around pitch and roll axes, and temperatures of motors, batteries and sensor boards. The sensor kit records data as the car is being driven around the race track. The sensor kit and the collected data can provide students exposure to real-time data collection and enhance their learning and understanding of the behavior of mechatronic systems, as well as serve for analysis of their cars performance. As with the self-driving package, the sensor kit was also tested on the same two Mechanical Engineering students, Jesse Kablik and Brian King, so they could evaluate our guide. With their limited experience in using an Arduino and Raspberry Pi, they provided valuable feedback for this second user guide, and like the self-driving user guide, this one is also ready to be deployed in ME 4320 and ME/RBE 4322 (Figure II).

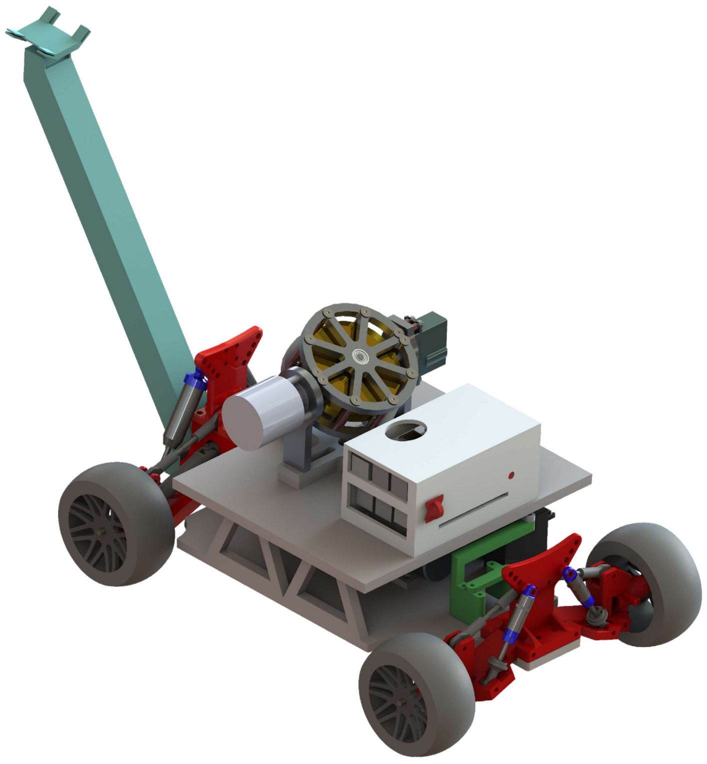


Figure II: Scale Car with Combined Self-Driving and Sensor Package

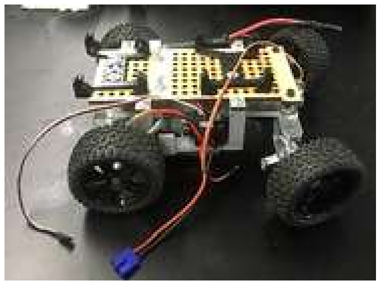
In the end, we were successfully able to create our self-driving and sensor packages to our specifications and were able to create user guides and test them on Mechanical Engineering graduate students, who were able to implement the self-driving and sensor packages on a car that they built. The total cost of the kit costs just over $400.

1 Introduction

Level 5 autonomy as defined by the SAE J3016 specification is an objective that automakers like General Motors, Ford, Tesla, Honda, Toyota, Renault-Nissan, and others hope to achieve within the next 10 years. Recent advancements in wireless technologies has enabled communication between vehicles and the surrounding infrastructure [1]. Semi-autonomous vehicles like the Tesla Model S, Cadillac CT6, and Audi A8 are already on the market [9]. The shift towards autonomy is not only due to the novelty of it all, but also with the possible benefits that come with it. 94% of car crashes nationally are due to driver error [2]. According to the National Vital Statistics Report, Motor Vehicle accidents were the highest contributor to accident related deaths [3]. Higher levels of autonomy have the potential to create safer road environments by reducing risky behavior behind the wheel like impaired or drunk driving. The world is heading towards a new era of vehicle navigation; therefore, the next generation of mechanical engineers must be capable of handling these developments and should be able to design systems around them.

Mechatronic systems are an amalgamation of mechanisms, sensors, controls and actuators. Self-driving cars are a growing class of mechatronic systems, and it is evident that many companies are investing large sums of money into the research and development of autonomous vehicles. It is therefore important to develop laboratory exercises that can augment content in existing courses in order to equip students to handle these fast-changing technological advancements. Autonomous driving involves neural network models and sensors. In order to develop laboratory exercises around this theme, it was decided to investigate the use of open source self-driving packages that could be integrated into scale (or remote-controlled) cars. Scale cars are cheaper, and the technology required is also low-cost. At Worcester Polytechnic Institute, there is a senior-level capstone design course called Advanced Engineering Design (ME4320) where student teams design and develop a scale-car with a custom-gearbox and a steering linkage. The remote-controlled cars that the students design in the course are about 1/10th the size of a normal car. They are made from plastic and metal components that are either custom made or prefabricated at the students' discretion. An electric motor allows the cars to drive forwards and backwards; and a steering servo allows the cars to turn left and right.

The teams must test their car at the end of the seven-week course in an end-of-term race. A demonstration of this event can be found in the following link: https://youtu.be/vFhJ4NV6oDo.While the mechanical design of the system may be robust; the sensors and control are all manual. Given that the students may not have enough practice driving scale-cars, the end-of-term race might not fully evaluate the car's ability to tackle varied driving conditions.



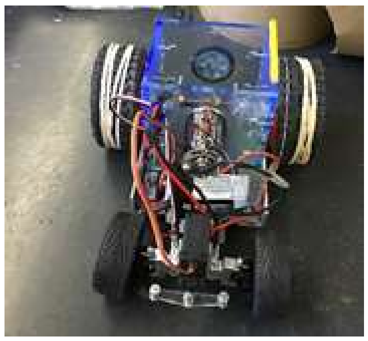


Figure 1.1: Top and Bottom of Typical Cars Designed in ME4320 (student built cars). The boxes house their custom made transmissions.

Figure 1.2: Images of the End of the Year Race for ME4320. This is where students test their driving ability along with their car design. The race can be found at https://youtu.be/vfhj4nv6odo. This image is reproduced from[5].

For our project, our goal was to enhance the Mechatronic System Design Education at WPI. We wanted the final product that students create to be one that could be driven either autonomously or manually. In order to remove the human (manual) element, Figure 1.3, we wanted to develop a kit that would be integrated into any scale car; especially considering that student's mechanical designs varied. We also wanted students to be more familiar with sensor capabilities in analyzing mechatronic systems.

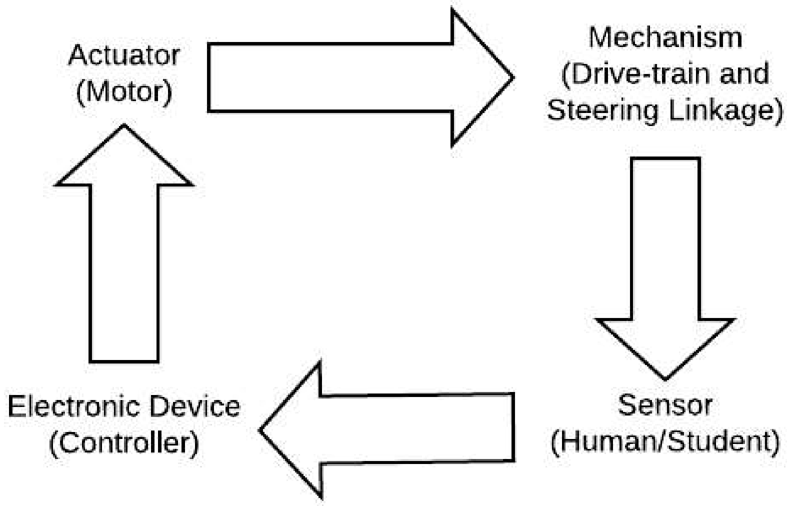


Figure 1.3: ME4320 Mechatronic System Diagram that Represents the RC Cars that Are Driven Manually in ME 4320. In this System the Students are The Sensors that Navigate the track.

To meet our project goal, we created three main objectives. Our first objective was to develop a modular sensor package. Our second objective was to develop a modular self-driving package. Our third objective was to design an installation procedure for the system.

Our first goal with the self-driving package was to have an RC car drive 4 laps around a track without human intervention using our self-driving package. To achieve this goal, we tested our self-diving package on seven different tracks. With each test, we gained experience creating neural network models and learned techniques on how to make a better neural network model, which in turn improved the self-driving performance of the RC cars we tested on until we were able to achieve our goal of an RC car self-driving 4 laps around a track.

The small knowledge gap needed to use the application allows students to be able to acquire a high-level/working understanding of how the system functions in order to create a working product, possibly within a day if their vehicle is already assembled. This is especially helpful for ME4320 since a vast majority of the students that take the class do not have experience coding or working with AI. The knowledge gap is small enough to be integrated into the class without overwhelming students with extra information on top of their current coursework.

The mechatronic system of the RC car can be broken down into the following components actuator(motor), mechanism(drive- train and steering linkage), sensor(student), and electronic device(Raspberry Pi). In this project our goal was to build on ME4320 to better equip Mechanical Engineering students to deal with mechatronic systems. This takes the student out of the equation and replaces them with sensors to navigate the track, and collect car data.

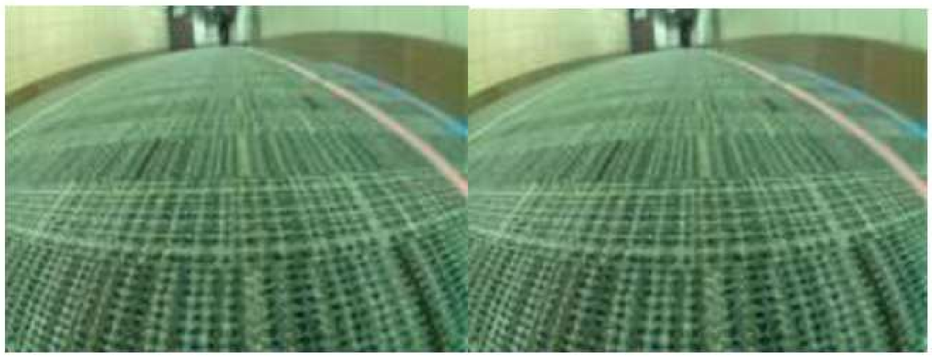
The paper will be organized as follows: background, implementation, results, conclusion, and acknowledgements. In the next chapter, we will describe the necessary background information related to the project.

2 Background

This chapter will describe the necessary background information related to the project. In this section AI Application, Available Self-Driving Kits, Available Sensor Kits, and the scale cars that we tested on will be discussed.

2.1 AI Application

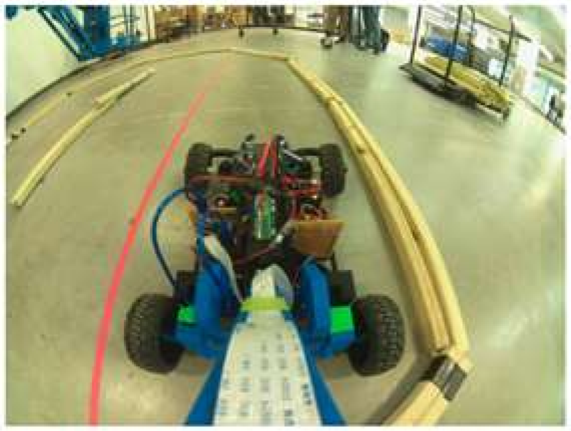
The type of software that can give a car the ability to drive autonomously is called Artificial Intelligence (AI). AI is the name given to any technique that enables a machine to mimic human behavior. Neutral Networks (NNs) are an application of AI[19]. They are a weighted system that takes inputs and then puts those inputs through a black box, also known as the hidden layers, to get an output [6]. In order to set the weights of the working NN model; the system must go through multiple iterations of running through data collected from the environment that the AI system would be operating in. After each iteration, the weights are adjusted in order to improve performance. When the NN's performance starts to plateau, the training is complete. This means the final weights that will be used during operation of the working NN model are set. The performance of a NN is heavily dependent on the training data used to create the working model [19]. If the training data does not properly represent the operating conditions, then the working model is more likely to have an undesirable outcome once it must navigate the target environment. [6] In our case the desired outcome is a remote-controlled car navigating a track without human intervention.



{"cam/image array": "36586_cam-image_array_.jpg", "user/angle": 0.0, "user/throttle": -0.09485152745139927, "user/mode": "user", "milliseconds": 2485334}

Figure 2.5: The Above Images are Examples of training data Collected Using DonkeyCar Software.

This project we are undertaking is the second iteration of this project. The previous team's project statement was broken down into five components [35]. First, test the feasibility to manufacture a 1: 10th scare car that is equipped with front and rear suspension. Second, test the performance of a modular sensor package capable of producing live data updates. Third, test the ability of a trained RC car to drive itself around an indoor track without the use of mapping. Fourth, the car will navigate the track while racing against other human driven manually controlled RC cars. The fifth goal was for the team to create a modular AI system that could provide RC cars with autonomous ability just by attaching the system to the vehicle[35].



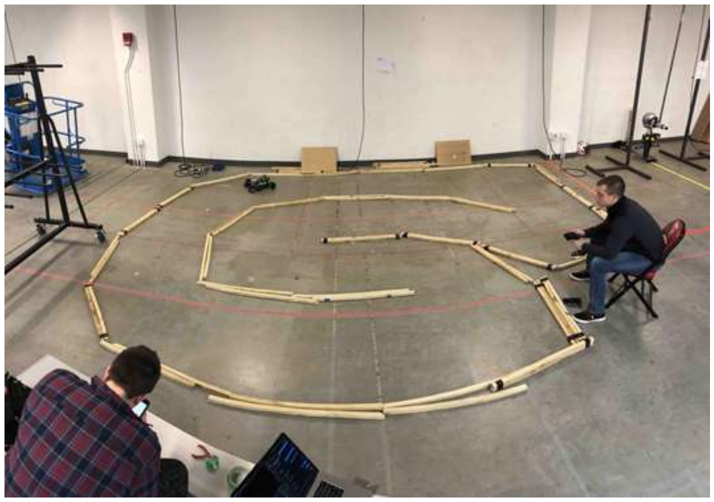


Figure 2.4: Top and Bottom of a Previous Iteration of the Projects's Car (2019) and Testing Track

The AI created by the previous iteration of the project operated under the workflow represented in Figure 2.2. Image data is collected from the camera and then processed by the Raspberry Pi. The image data goes through an OpenCV masking function that reduces the dimensionality of the image in order to improve the performance of the Neural Network The masked data is input into the Neural Network that is on the Raspberry Pi. The output of processed image data going through the Neural Network are driving instructions sent to the Arduino through a serial connection. The Arduino then sends those instructions to their specific mechanical systems. A flow chart of the workflow is shown in Figure 2.2.

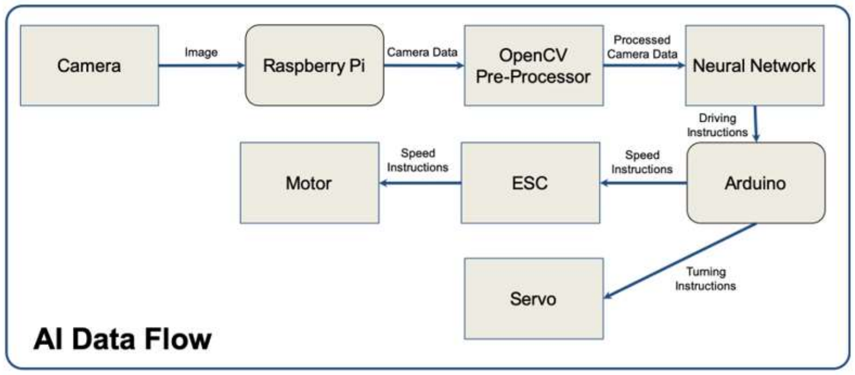


Figure 2.2: Work Flow of the AI System from Previous Iteration of the Project. Reproduced as in from [35].

Figure 2.1 shows an example of a car camera image going through masking in order to reduce the complexity of an image. The right side is the raw image data and the left side is the masked image. This mask function converts the pixels that are the color of the wall to be white scale, and all other colors are filtered to be black.

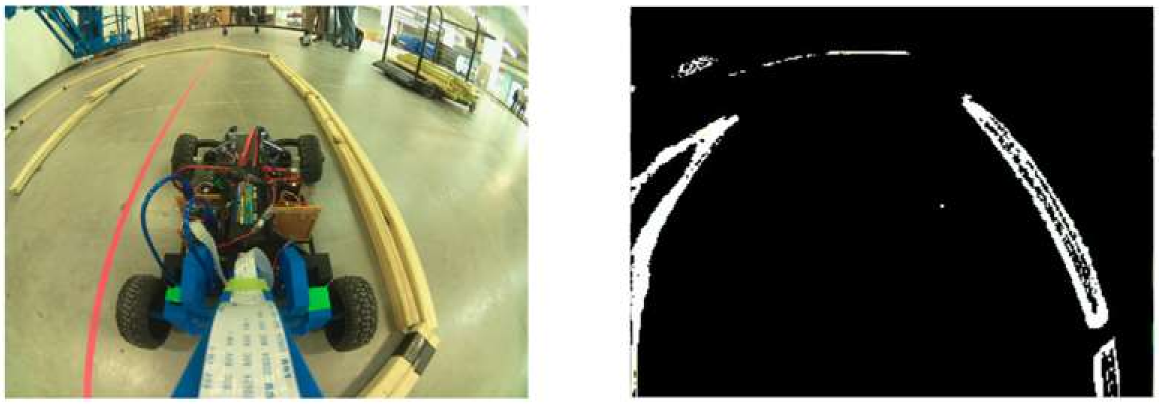
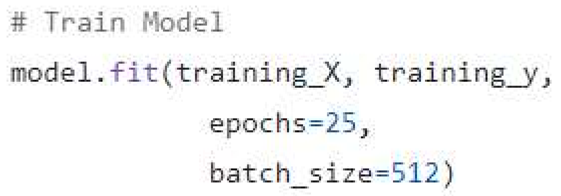


Figure 2.1: Image Masking Example. This particular masking function is meant to highlight the borders of the track to help the Neural Network find a pattern in the data.

Their team collected training data by driving the car using a radio controller while the Raspberry Pi camera collected video. Their video frames were labeled with the ESC and servo values that were sent from the radio controller. It took between 5 to 20 minutes to train their models. In the end of their project their car was only able to navigate 50% of the track. This can be attributed to the fact that their models took such a short time to train. Meaning, the Neural Network model was not able to go through enough iterations to get the optimum performance. In his code the training goes through 25 epochs. An Epoch is one cycle through the full training data set. Software that we used to train our NN had 100 epochs. We think added cycles of training would have improved their performance.Their NN was also never trained for vehicle avoidance.



```
# Train Model
model.fit(training_X, training_y,
          epochs=25,
          batch_size=512)
```

Figure 2.3: A Clip of Code from Last Year. Epoch Stands for One Cycle through the Full Training Data Set. Reproduced as in from [35]

2.2 Available Self-Driving Kits

AWS Deep Racer [10], OSOYOO Robot Car [9], and Yahboom [11] are all self-driving kits that are currently on the market. The issue with some of these kits is that they come designed as standalone products for creating a self-driving car rather than add-ons to an already built scale car. Cost is also a contributing factor for these kits. The AWS Deep Racer has a list price of $399.00 along with the additional cost that is added on for the amount of time that one trains the model using AWS [9]. The OSOYOO Robot Car achieves self-driving in its line tracking mode but cannot avoid obstacles when it is in this mode, since that feature is in a separate mode. This is because it uses an Arduino for its processing, which can only run one program at a time [9]. Apart from the market, there are currently many DIY self-driving projects that are showcased online. Websites like Medium and YouTube have content that guide individuals through the process of building a self-driving RC car. Some examples of this are *DEEPPICAR* Series [12], *Ryan Zotti: How to Build Your Own Self Driving Toy Car* [13], and *I built my own Self-Driving*

(RC)! [14]. The common electronics that can be found in both market kits and DIY projects to give their system autonomous ability are Raspberry Pi's, Raspberry Pi camera modules, and Arduinos [9][11][12][13]. Python and its packages were commonly used in developing autonomous models. According to GitHub, python is the most popular language when it comes to machine learning [15].

Looking at Table 2.2, cost was a hard factor to determine because these companies offered different products that had self-driving ability, and the cost would change per product. Also, for open source applications ones cost changes depending on where they buy materials and what materials the user currently possesses. This creates a large range of possible pricing that makes it difficult to generalize. Open source was important because it allowed for collaboration outside the community at hand. This is important when it comes to troubleshooting, because odds are someone has faced a similar problem that a user may be facing. That added resource on top of solid documentation is what our team found helpful in the past when we are getting acquainted with new software.

Table 2.2: Self-Driving Kits

	Donkey® Car	Osoyoo® Robot Car	LEGO® MINDSTORMS ®	Yahboom	AWS Deep Racer
Cost	-	-	-	-	$349
Open Source	Yes	Yes	No	No	No
Modular: Can easily go from one custom designed car to the next	Yes	No	No	No	No
Documentation	Yes	Yes	Yes	Yes	Yes

Ultimately, we decided to use Donkey Car software. It is a free open source software that assists in collecting training data and developing NN models. The primary factors that drove our decision was that it was well documented, had an active community, and small knowledge gap to overcome. The Donkey Car website has many guides that navigate students through the process selecting hardware and electronics. It also has in-depth instructions into how to install/utilize the Donkey Car software along with suggestions as to how to optimize it. There is an active Discord and multiple blog posts that can serve as resources for students to get a better understanding of the software. Donkey Car is the first part of our Self-Driving/Sensor Kit that will be integrated into the ME 4320 course as shown in Figure 2.6.

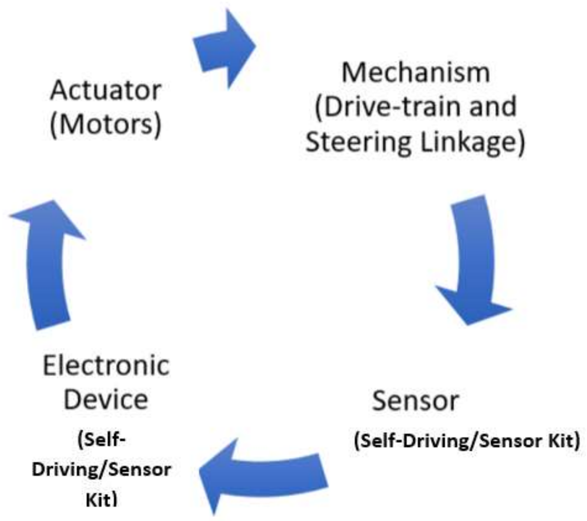

Actuator (Motors)
Mechanism (Drive-train and Steering Linkage)
Sensor
(Self-Driving/Sensor Kit)
Electronic Device
(Self-Driving/Sensor Kit)

Figure 2.6: Future ME4320 Mechatronic Diagram once the Self-Driving/Sensor Kits Become Integrated into the Class.

When it came to deciding which development board to choose the two options were the Raspberry PI and the Jetson Nano Dev Board. The specifications of the two side by side can be found in **Table 2.1**. The Jetson Nano has a more powerful GPU which is very pivotal when it comes to processing and training Neural Network Models. They are specialized in matrix operations, and these repetitive operations are used in Neural Network Models[19]. They also have higher AI performance. GFLOPs is a standard measure of computing power [21]. However the Jetson Nano does have some down sides. It cannot connect to the internet Wifi without an add on. It also has much larger board dimensions than the Raspberry Pi. This could create a challenge to design around. It is also more than doubles the price of the Raspberry Pi. Another factor we considered is that many of the students in the ME4320 class would be novices when it came to using these electronics; meaning the possibility of one of these card-sized computers breaking due to misuse is a substantial factor. Ultimately, we ended up choosing the Raspberry Pi because the most heavy GPU activity would have been training the Neural Network models. Which we did on our personal computers. The boards would either be collecting data or running the H5 file of the finished Neural Network model. The Raspberry was also the most cost effective in terms of creating a class set to get them aware of this new technology.

Table 2.1: Specificationations of Jetson Nano Dev Board vs. Raspberry Pi 3B+

	Jetson Nano Dev Board	Raspberry Pi 3B+
AI Performance	472 GFLOPs	21.4 GFLOPs
CPU	1.4 GHz 64-bit Quad- Core Arm Cortex-A57 MPCore	1.4 GHz 64-bit quad core ARM Cortex-A53
GPU	128-Core Nvidia Maxwell	Broadcom VideoCore IV
RAM	4GB LPDDR4	1GB LPDDR2 SDRAM
GPIO Header	40-pin	40-pin
Board Dimensions	100 X 79mm	85 X 56mm
Wireless	None	Dual-band 802. 11ac wireless LAN, Bluetooth 4.2
Ports	Ports 4X USB 3.0, wire ethernet 10/100/1000Mbps	4 USB 2.0, Wired Ethernet up to 330 Mbps
Multimedia	2160p30 (H.264)	1080p30 (H.264)

Video Output	HDMI, Displayport (4K)	HDMI, Display Serial Interface (DSI)
Camera Serial Interface	YES	YES
M.2 Key E Slot	YES	NO
Price	$99	$35

**Kimari, K. (2019, June 24). Nvidia Jetson Nano vs. Raspberry Pi. Retrieved May 18, 2020, from https://www.maketecheasier.com/nvidia-jetson-nano-vs-raspberry-pi/*

Apart from our MQP WPI has other projects that are connected to autonomous driving taking place on campus. A MQP from 2019 developed a modular system that used mapping software to give off-road vehicles Level 3 autonomy according to SAE's standard[22]. A MQP from 2015, created an in-depth recommendation packet for the use of subsystems against the real-time challenges required for driving an autonomous, wireless, and vision-based system[23]. Apart from WPI, the growing interest in autonomy has created opportunities elsewhere. DIY Robocar races are planned and scheduled by Donkey Car community members. In these events Donkey Car hobbyists come together and race their AI's head to head[11]. The Figure #, shows a snapshot of one of those races. Amazon also has a self-driving league. Developers , novices and experts, train their own self-driving model for the AWS DeepRacer. Once a Developer thinks they have a strong model that can enter the model into virtual races and scheduled real time competitions[11].

Figure 2.7, This image is an example of a DIYRobocars Races that took place in May 2017. Competitors come together to see which of their car designs and AI models work the best. Reproduced from [11].This is image was taken from the Donkey Car Website: https://www.donkeycar.com/examples/may-2017-diyrobocars-race

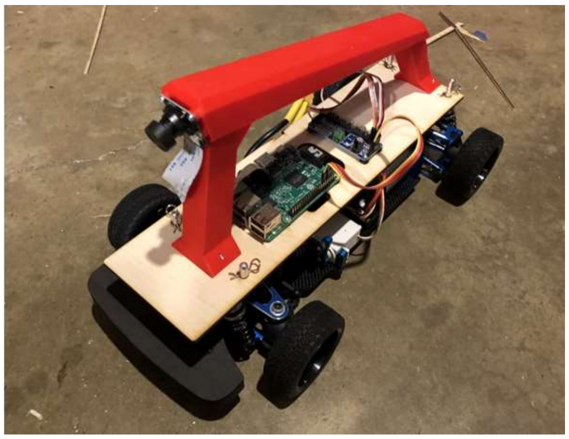
2.3 Available Sensor Kits

There are a wide variety of sensor kits that are currently on the market. Examples of this are the Elegoo Sensor Kit[16], Elec Freaks Sensor Kit[17] and the Hiletgo Sensor kit[18]. We wanted students to be able to acquire information on the orientation, speed, and temperature variance on their custom designed vehicles. These kits come with unnecessary sensors for our needs; these kits do contain a plethora of valuable sensors, however we did not find a need for extra impractical sensors. We only needed sensors to record the orientation, speed, and temperature variance that was taking place in the scale vehicles. Many of the kits we found required some prior knowledge to assemble into a system [25][17][18]. We wanted to create a kit that could be plugged into any of the systems that students designed so that they could acquire data in a matter of minutes rather than spending an excessive amount of time trying to install the sensors. By working on developing a modular system that incorporates simple-to-install individual sensors, the goal of reinforcing the skills in mechatronic systems learning will be applied across the ME 4320. Additionally, buying sensors individually is significantly cheaper than purchasing them in kits, as we need many duplicates, we would have to buy at least 20 kits, as each kit contains one of each sensor. To add on, many sensors from the kits would go to waste on our project, we would never need to utilize a water level sensor [25], flame sensor [25], or heartbeat sensor [18]. As the kits are a great learning tool, they should not be completely dismissed, however for our application where it was known to us what we wanted to record, it made no sense to purchase a commercially available kit.

The sensors chosen for implementation in the sensor package for the purposes of the project were thermal sensors, hall-effect sensors, and an inertial measurement unit. The previous MQP team that had been working on the project before us had worked on implementing these sensors in the years prior, with the exception of the hall-effect sensor. The inclusion of these sensors accomplishes a few tasks; the sensors have varying degrees of complexity for the users to work with when developing the sensor package, as well as monitor the most important key statistics of remote travel: orientation and velocity. With the combination of these sensors, and their compatibility with Arduino software, it made deciding which sensors to implement on the first iteration of the sensor package quite simple. The BNO055 9-DOF Absolute Orientation IMU Fusion Breakout Board, Hall Effect Magnetic Sensor Module for Arduino, and ASAIR AM2302 Temperature and Humidity Sensors all serve the purpose of being Arduino accessible, simple electronic sensors that will work in tandem to accomplish the task of monitoring the vehicle during travel.

2.4 Cars Tested on

There were three types of cars that we used in our development of our modular kits: The Mission D car, student-made cars, and 3D printed cars. The Mission D car is a commercially available car that costs about $95 from the DonkeyCar store. The kit comes with 3D printed laser cut chassis, screws, OV5647 camera for Raspberry Pi or a IMX219 for Jetson Nano, Servo Driver, and jumper cables. The kits do not come with an RC car, Raspberry Pi, SD card, or battery. It is designed for 1/10 Scale cars; specifically HobbyKing Trooper Pro, Tamaya TT01, and HobbyKing Mission D. The student-made cars are scale cars that were made by students in previous years of the Advanced Engineering Design course at WPI. The 3D printed cars are cars that incorporate many 3D printed parts. The assembly is pretty straight forward, but can take 3 hours total to make sure everything is assembled correctly. This time can go up or down depending on the skill of the assembler. The end product should look something like the cars in Figure 2.8.



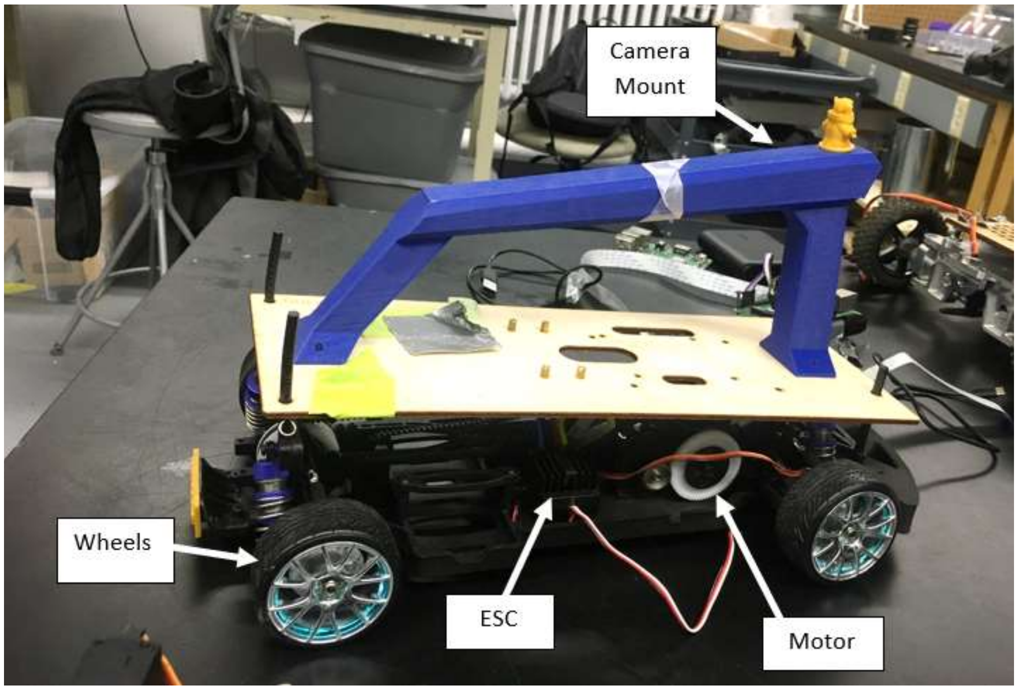


Figure 2.8: Top and Bottom Pictures are of the Mission D car. The Donkey Car Kit is assembled onto the HobbyKing Mission D car. Top image is reproduced from[11].

Figure 2.8 is a picture of a Mission D car. As previously mentioned, this is a scale car that is commercially available for purchase. It has rubber wheels, has a plastic chassis, and is made of mostly aluminum and plastic parts. It has a brushed DC motor to drive forwards and backwards and has a servo to steer left and right.We first used the Mission D car so we could learn how the Donkey Car platform and Neural Network implementations work on a scale car.

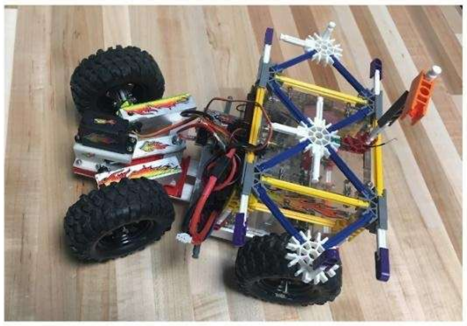


Figure 2.9: Example Student-Made Advanced Engineering Design Car

Figure 2.9 is a picture of a car that a student made in a previous Advanced Engineering Design course at WPI. Student made cars are made using mostly plastic and aluminum parts. They have rubber wheels, a brushed DC motor, and a steering servo. After testing our self-driving system on the Mission D car, we started testing on the student-made cars. Student made cars are not commercially manufactured, meaning their designs can vary greatly from one another. This way, if we could get the self-driving system to work on a student-made car, then it would be possible for students to be able to get the cars that they build to self-drive.

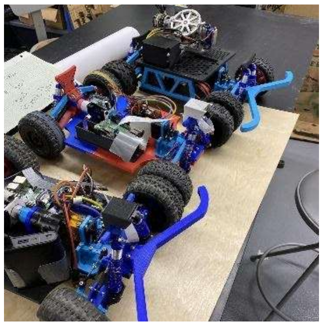


Figure 2.10: Example 3D Printed RC Cars

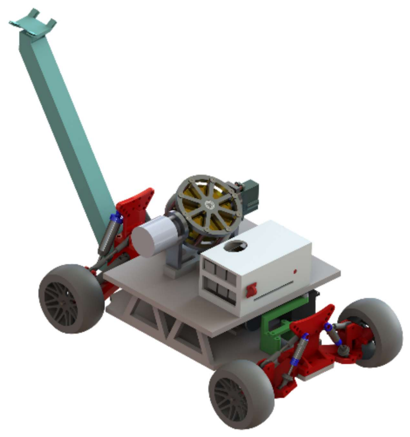


Figure 2.11: CAD Render of a 3D Printed RC Car

Figures 2.10 and 2.11 show 3D printed RC Cars. These cars were made by our partner MQP team who designed these 3D printed RC cars to be used in WPI's Advanced Engineering Design course. They are made using mostly plastic 3D printed parts, with some aluminum elements. They have rubber wheels, a brushed DC motor, and a steering servo. So, we tested our self-driving package and sensor package on these 3D printed cars as these cars would also be used in the Advanced Engineering Design course.

Table 2.3: Car Specifications for Figures 2.10 and 2.11

Part	**Name**
Wheels	PRO1190-013
Brushless DC Motor	JUSTOCK 3650SD G2 Brushless Motor
Steering Servo	SAV-SC-1256TG

The previous iteration of the project's MQP team was not able to create an AI that could navigate a track completely on its own. They were also not able to find a way to successfully display the sensor data that they acquired. There were a lot of possible options for components to use to achieve our objectives of creating a modular self-driving/sensor package. However, after weighing our needs we were able to narrow down the options to our current software and electronic components. Next, is the Project Goals and Requirements chapter where we will go into our processes of how we are going to achieve those goals.

2.5 Summary

This chapter we described the necessary background information related to the project. We discussed AI Applications, Available Self-Driving Kits, Available Sensor Kits, and the scale cars that we tested on.

In the next chapter, we will discuss our project goal, the objectives we made to meet that project goal, and requirements for each objective for us to evaluate our progress.

3 Project Goals and Requirements

In this chapter, we will discuss our project goal, the objectives we made to meet that project goal, and requirements for each objective for us to evaluate our progress.

The goal of our project is to design and implement a modular self-driving system with sensors that can be integrated onto any scale RC car. This will allow students in the ME4320 course, who build their own RC car as part of that course, to turn their RC cars into self-driving RC cars with sensors. To meet this goal, we have five main objectives. Our first objective is to develop a modular sensor package. Our second objective is to develop a modular self-driving package. Our third objective is to design an installation procedure for the system (the two packages). Our fourth objective is to improve the self-driving package to the point that it qualifies as level five autonomous. Our fifth objective is to integrate the CMG into our system.

3.1 Sensors

To develop a modular sensor package, we will first select the sensors that will be part of our sensor package and the location that the sensor will be on the car. The first type of sensor that we have selected are thermal sensors. There will be a thermal sensor on the ESC and the motor of the car, but also one in the sensor box. The sensor box, as seen below in Figure 3.1 will be a box that contains the Raspberry Pi and arduino, as well as the extra wiring from the sensors. These three locations were chosen for the thermal sensor as it is important to know if the motor, ESC, or Raspberry Pi are overheating as that will degrade the performance of the car. The second type of sensor that we have selected is a tachometer/hall effect sensor. If we attach this sensor to the rear axle, we will get a reading of how fast the rear axle and therefore how fast the motor and the wheel are rotating. This sensor can be useful for determining the velocity, torque, and acceleration of the car. The third sensor that we have selected is an IMU. The IMU will go on the sensor box, and will be useful for determining the speed, acceleration, and direction of the car. The fourth sensor will be a strain gauge attached to the center of the chassis to measure the flex of the car's chassis, and the last sensor will be a camera on the back of the car that will be used for the self-driving package.

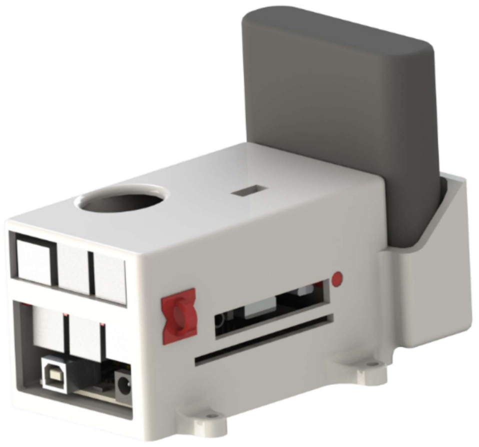


Figure 3.1: Render of Sensor and Self-Driving Housing

After selecting the sensors for our sensor package, we will have to choose the data that we want to collect from each sensor. The thermal sensor will be used to collect temperature data of the Raspberry Pi, the electronic speed control (ESC), and the motor on the car. The IMU will be used to collect data on the roll, pitch, yaw, acceleration, and velocity of the car. The tachometer will be used to collect data on the RPM of the wheel and motor, and the velocity and acceleration of the car. The strain gauge will be used to determine the flex of the car's chassis, and the camera will be used to send images to the Raspberry Pi for the neural network used in the self-driving package.

After determining the data that we will collect from each sensor, we will then have to figure out how to connect all of the sensors and determine if we can add more sensors to the system. The sensors will be connected to the Arduino, which will then send the data to the Raspberry Pi, which will display the data on a web server. This web server will be locally hosted on the Raspberry Pi, which can be reached by connecting to the Raspberry Pi remotely. As mentioned earlier, there will be a sensor box that will hold the Raspberry Pi and Arduino. This box will be a 3D printed enclosure that will protect the microprocessors. To make the sensor package modular, we will pre-code the sensor package to support all of the selected sensors, but also to run with a fewer number of sensors. Each sensor will have its own code package, allowing each sensor to work independently from the other sensors, allowing for interchangeability. The webpage will be displayed on a Flask web server, allowing students to get real-time updates on their cars. We will also create an option for the students to download a csv file of all of the data from the current run of the sensors.

Next, we determined the best way to go about getting the data from the sensor package. As previously mentioned, the data will be displayed on a local host (the Raspberry Pi), and can be connected to remotely using any laptop. After that, we determined how to display the data on the dashboard. On this webserver, we can graphically display all of the data from the sensors that are plugged in, and display it in a manner that is easy to visualize. The data can also be downloaded from a csv file too. Lastly, we will program the car to stop if the Raspberry Pi, ESC, or motor's manufacturer's maximum specified temperature is exceeded.

After we have made our sensor package, we will test it to make sure it operates as expected. The first set of tests are designed to make sure that we get the data that we expect from the sensors. For the thermal sensors, this means verifying that each of the three thermal sensors accurately and quickly responds to rapid changes in temperature and that they can send the temperature information to the Raspberry Pi. This test can be done by placing each of the sensors in an extreme thermal condition and checking the responses as relayed to the server. For the IMU, we will need to make sure that the IMU properly displays the roll, pitch, yaw, and acceleration of the car. One way to do this is to move the sensor to easily defined angles, such as 90 degrees and 180 degrees to verify the direction of the IMU. Then, to test acceleration, we can drop the IMU from a short height to make sure that the acceleration is 9.8 m/s^2. If these tests are successful, we will then design a test where we drive the car around a set turn at a set speed, and then measure the data from the IMU. We will do theoretical calculations to verify that the roll sensor is working correctly. We can also drive the car on a sloped surface to make sure that the roll of the car is equal to the slant of the slope. Lastly, we can put the car on its side (a 90 degree angle) and verify that the IMU reads a roll of 90 degrees.

For the tachometer, we will verify that the sensor can detect the changes that coincide with the rapid rotation of the magnet attached to the axle. First, we will test that the tachometer can record and react to simply the presence of the magnet. Second, we will test that the magnet can be detected repeatedly by rotation that axle at varying speeds and recording that the tachometer can detect each passing of the car. Finally, we will attach the sensor/axle combination onto the car itself. Then, we will have the car drive a set distance and record the time it takes to travel said distance. Lastly, we will compare the recorded data with the theoretical calculations by dividing distance over time to ensure that the Tachometer is working as expected.

For the strain gauge, we will verify that the strain gauge can detect and report changes in strain on the chassis so the operator can know the flex in the car's chassis. We can ensure that the strain gauge has the correct readings by adding known weights to a chassis and performing theoretical calculations as to what the strain gauge should read. We can also look up the force required to break the car's chassis and then monitor the strain gauge to make sure the chassis does not break during operation.

The next test will be to ensure that the sensor package is truly modular. We can verify this by trying to run the system with only one sensor at a time, or with various subsets of all of the available sensors. The goal is to make each sensor independent of one another so as to allow the students in the ME4320 course to choose what sensors they want on their car.

The third test will be to verify that a CSV file of all of the data can be downloaded, and that the data is properly displayed to the web server. This involves recording through the use of the camera on the car, and transmitting that data successfully from the Raspberry Pi. The last test for the development of the sensor package is to ensure that the camera works by capturing some of the video that the camera takes. After this test, the development of the sensor package will be finished.

3.2 Self-Driving

While we are developing the sensor package, we will also develop a modular self-driving package. The first step in this process is to obtain a working RC car that has no mechanical issues. Currently, we have a small 1/16th scale car that was built by one of the students in last year's ME4320 course, but we still have to add some parts before the car can drive. After we have a working car, we will need to attach a Raspberry Pi and a camera to the car so the car can see and react to its surroundings as it drives around a track. The next step will be to ensure that the system can utilize neural networks, machine learning, and computer vision to drive the car around a track. Then, after we can get the car to do a full lap around one track, we will design different tracks for the car to drive on and to test the results. We will also determine various driving conditions that we will want the car to drive in.

After we build the self-driving package, we will run tests on the system. The first test will be to make sure that the car can go around a track without bumping into the sides of the track. We will video record the tests with our own cameras from a vantage point in the testing location that have a good view of the whole track. This way we can verify that the self-driving system works. The next test will be to test the neural network using the Python unit test library. We will check the loss functions (which measure our model's performance) that our model runs to make sure that the loss functions are appropriate for our self-driving system and that the loss functions are properly scaled. We can also check the intermediate outputs and connections of our neural network, as well as the design parameters to make sure the system is behaving as expected. Then, we will check our documentation to make sure that all of our code is documented. The last test that we will do on our self-driving package will be to compare the performance of our self-driving RC car to the performance of other self-driving RC cars, such as the donkey car. We will have to create comparison tests that make sense between the different RC cars in order to highlight which RC car performs best under different conditions. This comparison test will conclude the development of the self-driving package.

For our self-driving package, we wanted to show that we could get a car that would be used in the Advanced Engineering Design course at WPI to self-drive at least four laps around a track. We thought that this goal would demonstrate that a car made in the course could consistently self-drive itself around a track. To meet this goal for our self-driving system, we decided to learn how to use the Donkey Car platform by trying to get scale cars to self-drive on various types of tracks.

3.3 User Guides

After making the self-driving and the sensor packages, we will design an installation procedure for the system. This objective is aimed at designing a process that the students from ME 4320 can follow to install the self-driving and sensor packages onto the cars that they build within one or two class periods. First, we will have to obtain IRB Approval as we will perform user experience testing using the students from the ME 4320 and ME 4322 course as human subjects. This step can be done in advance to streamline the process. The next step will be to develop a procedure for installing our systems on an RC car. This procedure will include a list of all of the parts required for installation and a list of all the tools required. The first step in this procedure will be to attach the sensors to the car (with specific locations), and the next step will be to wire all of the sensors to the Raspberry Pi. We will add more steps to this procedure depending on the final design of our sensor package.

After making the installation procedure, we will identify aspects of our installation procedure that can be optimized by performing the procedure ourselves. We will look at which steps take the longest and which steps are worded in a confusing manner and edit our procedure to adjust these steps. Then, we will give the installation procedure to students in the ME4320 class and see if they can install the system within one class period. If they can, then we have successfully completed this objective. If they can't, we will take notes on how they followed the procedure, what questions they asked, what steps took them the longest, and what steps were hardest for them to complete. Then, we will use this feedback to optimize the Installation Procedure. We will repeat this user experience testing cycle until the entire procedure can be performed by a student in the ME4320 class within one or two class periods. Once a student can complete the procedure in one to two class periods, then we will have fulfilled this objective.

3.4 Level 5 Autonomy

The next objective will be to make the RC self-driving package satisfy level five autonomy. This first requires the self-driving package to be developed. Next, we have to define what level five autonomy is. For our purposes, this means that the car can drive over bumpy surfaces, the car is capable of handling any conditions we put it in, and the car can drive around the track without any human input. To accomplish this, we must program the code to be robust enough to handle all of these different cases. To test that our car is level five autonomous, we will make multiple tracks to test the car on with ramps, tunnels, and hills. We will also change the tracks during runs to try to confuse the car. When issues present themselves, we will fix them by improving our self-driving code. We can also test conditions that are outside of the basic expectations for our self-driving package. This would include throwing objects at the car actively trying to cause a collision with the self driving car and another car, loosening one of the wheels of the car, forcing the sensors to malfunction, and other ideas that we can come up with. If an RC car running our self-driving package can withstand all of these obstacles and pass all of these tests, then we will have achieved level five autonomy.

3.5 CMG

Our last objective will be to integrate the CMG as part of our self-driving and sensor packages. The CMG, a control moment gyroscope, is used to apply torques to a vehicle. This could be used to maintain the balance of a car as it drives on slopes, goes up ramps, or even makes the car flip over after wiping out. First, we will need a CMG small enough to fit onto the car. Team 2 of our project will work on that. After the CMG is built, we will connect the CMG as another sensor for the sensor package. Because we want the CMG to be able to course correct, it will need to be able to take input from the Arduino that the IMU is connected to. Another code package will be devised to allow for direct communication of the car's orientation and the CMG to maintain balance. To test that the CMG is behaving properly, we will first make sure that the CMG can correct orientation. This will involve placing the CMG chassis in different positions and having the CMG move to correct the orientation of the car. Then, the CMG will be connected to the Arduino, and a code package for the CMG will be used to dictate what direction and speed the CMG will reposition to based on the orientation data from the IMU. The same tests from the IMU will be performed again with the car at various degrees of incline so we can verify the accuracy of data from the IMU. Then, the CMG will be monitored to see if it is reacting in the proper manner. After completing this test, we will have accomplished our final objective.

3.6 Summary

In this chapter, we discussed our project goal, and the five objectives that we made to meet that project goal. Our first objective and requirements for each objective for us to evaluate our progress. Our first objective is to develop a modular sensor package. Our second objective is to develop a modular self-driving package. Our third objective is to design an installation procedure for the system (the two packages). Our fourth objective is to improve the self-driving package to the point that it qualifies as level five autonomous. Our fifth objective is to integrate the CMG into our system.

In the next chapter, we will go over how we implemented the self-driving kit, the tests we did on the self-driving kit, and how we made the user guide for our self-driving kit.

4 Self-Driving Implementation

In this chapter, we go over the overall self-driving kit development process. Next, we review the cars that we tested our self-driving kit on and go over the components required to make our self-driving kit. Then, we cover how we collected training data when using our self-driving kit and what tracks we tested our self-driving kit on. After that, we go over the various tests that we performed on our self-driving kit. Lastly, we go over how we made the user guide for our self-driving kit.

To meet our project goal, we had to make a self-driving package and a user-guide on how to use the package. Our overall process is shown in Figure 4.1.

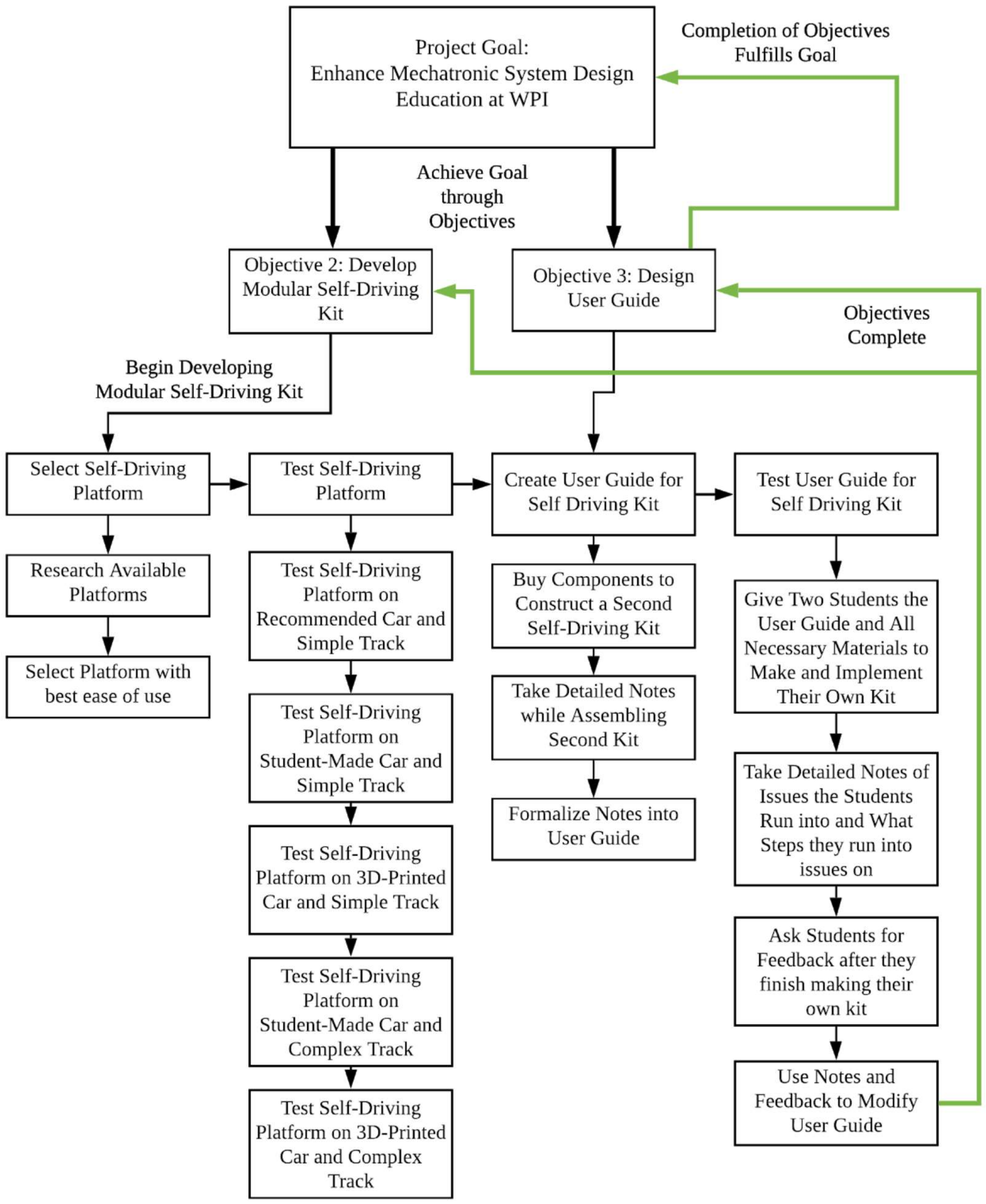

Project Goal: Enhance Mechatronic System Design Education at WPI
Completion of Objectives Fulfills Goal
Achieve Goal through Objectives
Objective 2: Develop Modular Self-Driving Kit
Objective 3: Design User Guide
Objectives Complete
Begin Developing Modular Self-Driving Kit
Select Self-Driving Platform
Research Available Platforms
Select Platform with best ease of use
Test Self-Driving Platform
Test Self-Driving Platform on Recommended Car and Simple Track
Test Self-Driving Platform on Student-Made Car and Simple Track
Test Self-Driving Platform on 3D-Printed Car and Simple Track
Test Self-Driving Platform on Student-Made Car and Complex Track
Test Self-Driving Platform on 3D-Printed Car and Complex Track
Create User Guide for Self Driving Kit
Buy Components to Construct a Second Self-Driving Kit
Take Detailed Notes while Assembling Second Kit
Formalize Notes into User Guide
Test User Guide for Self Driving Kit
Give Two Students the User Guide and All Necessary Materials to Make and Implement Their Own Kit
Take Detailed Notes of Issues the Students Run into and What Steps they run into issues on
Ask Students for Feedback after they finish making their own kit
Use Notes and Feedback to Modify User Guide

Figure 4.1: Flowchart of Self-Driving Kit Development

To meet our objective of developing a modular self-driving kit, we first had to select a commercially available self-driving platform for us to use (Figure 4.1). As previously mentioned, we researched the available self-driving platforms, and selected a platform based on various factors such as ease of use. Next, we had to test this self-driving platform. Our first goal with the self-driving package was to have an RC car drive four laps around a track without human intervention using our self-driving package, and to achieve this goal, we tested our self-diving package on seven different tracks. First, we tested the self driving package on a recommended car on simple tracks. Next, we tested the self-driving platform on a student-made car on simple tracks. Next, we tested the self-driving platform on a 3D printed car on simple tracks. Next, we tested the self-driving platform on a student-made car on complex tracks. Lastly, we tested the self-driving platform on a 3D printed car on complex tracks. With each test, we gained experience creating neural network models and learned techniques on how to make a better neural network model, which in turn improved the self-driving performance of the RC cars we tested on until we were able to achieve our goal of an RC car self-driving four laps around a track.

After running tests on the self-driving platform, we needed to create a user guide that is easy for students in ME 4320 and ME/RBE 4322 to follow so that they can implement our self-driving kit on a scale car to create a self-driving scale car. We created a first draft of our user guide by buying all of the components necessary to make a second self-driving kit and taking notes as we constructed the second self-driving kit. We formalized these notes to make our user guide. Next, we tested our user guide on two Mechanical Engineering students, Jesse Kablik and Brian King, so we could evaluate our guide. We used the feedback that they gave us to improve our guide, completing our development of the modular self-driving kit and user guide. The kit is now ready to be deployed in ME 4320 and ME/RBE 4322.

4.1 Test Cars

There were three types of cars that we used in our development of our modular kits: the "Mission D" car (Figure 4.1.2), student-made cars (Figure 4.1.3), and 3D printed cars (Figure 4.1.4). As previously mentioned, the Mission D car is a commercially available car. The student-made cars are scale cars that were made by students in previous years of the Advanced Engineering Design course (ME 4320) at WPI. The 3D printed cars use many 3D printed parts.

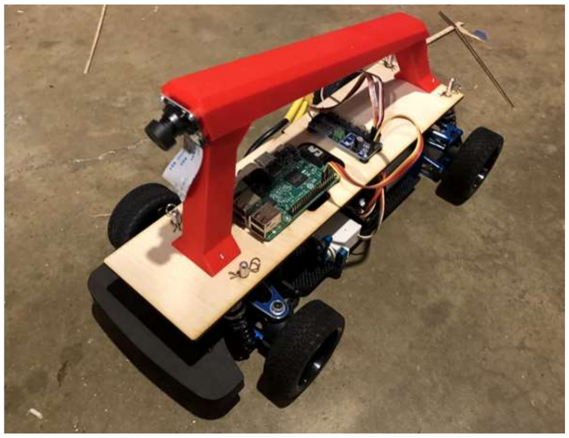


Figure 4.2: Picture of Mission D Car, Reproduce into [8]

Figure 4.2 is a picture of a Mission D car. This is a scale car that is commercially available for purchase that we used in our first tests on our self-driving system as it was a scale car recommended by the Donkey Car website. We first used the Mission D car so we could learn how the Donkey Car platform and Neural Network implementations work on a scale car.

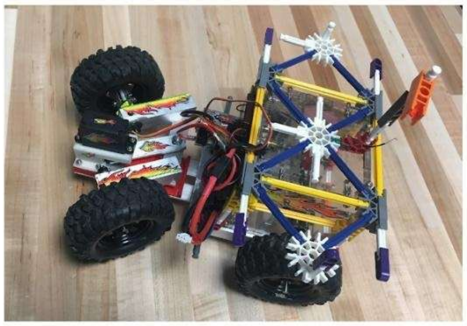


Figure 4.3: Example Student-Made Advanced Engineering Design Car

Figure 4.3 is a picture of a car that a student made in a previous Advanced Engineering Design course (ME 4320) at WPI. After testing our self-driving system on the Mission D car, we started testing on the student-made cars. These student-made cars are not commercially manufactured, so we suspected that there could be challenges in getting our self-driving system to work well on these cars. This is why it was important for us to test our system on these cars. Also, if we could get the self-driving system to work on a student-made car, then it would be possible for students to be able to get the cars that they build to self-drive.

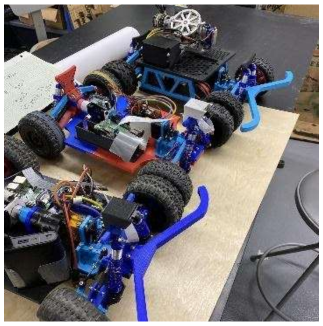


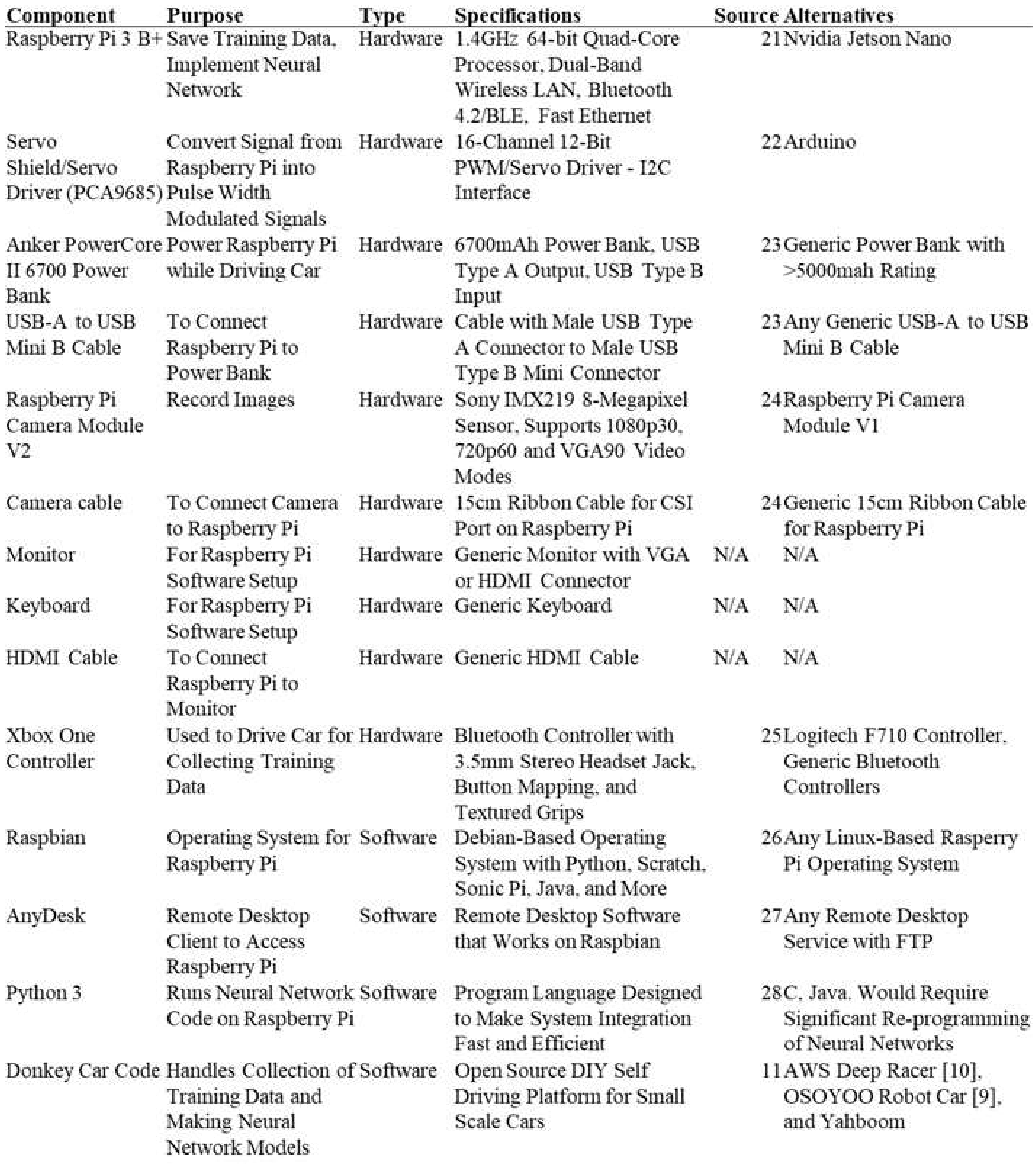
Figure 4.4: Example 3D Printed Scale Cars

Figure 4.4 shows 3D printed scale cars. These cars were made by our sister MQP team (Michael DeFrancesco, Michael Pierce, Alex Boggess, Julia Davenport, Richard Mohabir, and Zack Orbach) for use in the Advanced Engineering Design course (ME 4320). We tested our self-driving package on these 3D printed cars as these cars would also be used in ME 4320 and ME 4322.

4.2 Self-Driving Module Required Components

Our self driving kit is made up of both hardware and software components, each with their own purpose and specifications (Table 4.1).

Table 4.1: Table of Required Components for Self-Driving Package



Component	Purpose	Type	Specifications	Source	Alternatives
Raspberry Pi 3 B+	Save Training Data, Implement Neural Network	Hardware	1.4GHz 64-bit Quad-Core Processor, Dual-Band Wireless LAN, Bluetooth 4.2/BLE, Fast Ethernet	21	Nvidia Jetson Nano
Servo Shield/Servo Driver (PCA9685)	Convert Signal from Raspberry Pi into Pulse Width Modulated Signals	Hardware	16-Channel 12-Bit PWM/Servo Driver - I2C Interface	22	Arduino
Anker PowerCore II 6700 Power Bank	Power Raspberry Pi while Driving Car	Hardware	6700mAh Power Bank, USB Type A Output, USB Type B Input	23	Generic Power Bank with >5000mah Rating
USB-A to USB Mini B Cable	To Connect Raspberry Pi to Power Bank	Hardware	Cable with Male USB Type A Connector to Male USB Type B Mini Connector	23	Any Generic USB-A to USB Mini B Cable
Raspberry Pi Camera Module V2	Record Images	Hardware	Sony IMX219 8-Megapixel Sensor, Supports 1080p30, 720p60 and VGA90 Video Modes	24	Raspberry Pi Camera Module V1
Camera cable	To Connect Camera to Raspberry Pi	Hardware	15cm Ribbon Cable for CSI Port on Raspberry Pi	24	Generic 15cm Ribbon Cable for Raspberry Pi
Monitor	For Raspberry Pi Software Setup	Hardware	Generic Monitor with VGA or HDMI Connector	N/A	N/A
Keyboard	For Raspberry Pi Software Setup	Hardware	Generic Keyboard	N/A	N/A
HDMI Cable	To Connect Raspberry Pi to Monitor	Hardware	Generic HDMI Cable	N/A	N/A
Xbox One Controller	Used to Drive Car for Collecting Training Data	Hardware	Bluetooth Controller with 3.5mm Stereo Headset Jack, Button Mapping, and Textured Grips	25	Logitech F710 Controller, Generic Bluetooth Controllers
Raspbian	Operating System for Raspberry Pi	Software	Debian-Based Operating System with Python, Scratch, Sonic Pi, Java, and More	26	Any Linux-Based Rasperry Pi Operating System
AnyDesk	Remote Desktop Client to Access Raspberry Pi	Software	Remote Desktop Software that Works on Raspbian	27	Any Remote Desktop Service with FTP
Python 3	Runs Neural Network Code on Raspberry Pi	Software	Program Language Designed to Make System Integration Fast and Efficient	28	C, Java. Would Require Significant Re-programming of Neural Networks
Donkey Car Code	Handles Collection of Training Data and Making Neural Network Models	Software	Open Source DIY Self Driving Platform for Small Scale Cars	11	AWS Deep Racer [10], OSOYOO Robot Car [9], and Yahboom

As you can see in Table 4.1, our self-driving kit is made of many components. A Raspberry Pi 3 B+[21] is used to save training data and to implement the neural network model.

The Raspberry Pi 3 B+ has a 1.4GHz 64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE, and fast ethernet. A possible alternative to the Raspberry Pi 3 B+ is the Nvidia Jetson Nano as it is a micro computer designed to run neural networks.

A servo shield/servo driver (PCA 9685)[22] is used to convert the signal from the Raspberry Pi into pulse width modulated (PWM) signals that the steering servo and electronic speed control of scale cars used. This servo driver has 16 channels and can send 12-bit PWM signals based on the communications with the Raspberry Pi using the I2C interface. A possible alternative to this servo driver would be to use an Arduino to program and send PWM signals to the electronics that control scale cars.

We used the Anker PowerCore II 6700[23] power bank to power the Raspberry Pi while we were driving our scale cars. We could have used any generic power bank with more than a 5000 mah rating, as this capacity was always enough to do an entire day's worth of testing with the Raspberry Pi. We used a USB-A to USB mini B cable that came with the power bank to connect the power bank to the Raspberry Pi, but any generic USB-A to USB mini B cable could have been used.

We used the Raspberry Pi Camera Module V2[24] to record video images at 160 by 120 pixels that would be sent to the Raspberry Pi. This camera module uses the Sony IMX219 8-megapixel sensor that actually supports video images of up to 1920 by 1080 pixels at 30 frames per second, but this was greater than the resolution that we needed. An alternative to this camera module could be the V1 camera module, but this camera module worked for us without any issues. The camera module came with a camera cable to connect the camera to the Raspberry Pi, but any generic 15cm ribbon cable for Raspberry Pi's would have worked as well.

We used an Xbox One controller to drive the scale cars while collecting training data for our neural network models. The Xbox One controller connected to the Raspberry Pi via Bluetooth. We used the Xbox One controller as we already owned some, but any generic bluetooth controller could have worked. Additionally, we started using a Logitech F710 controller in C term as they are cheaper than Xbox One controllers and connect to the Raspberry Pi via a USB dongle, removing the need to pair the controller with the Raspberry Pi.

We used a generic monitor and keyboard to set up the software on the Raspberry Pi. The keyboard plugged into a female USB type A port on the Raspberry Pi, whereas the monitor connected to the Raspberry Pi using a generic HDMI cable.

We used Raspbian as the operating system for the Raspberry Pi. It comes with python already installed, which we needed to run the self-driving code, but we think other linux-based Raspberry Pi operating systems could be used. Raspbian worked for what we needed it to do, and it's free, so we saw no need to test other operating systems. We used AnyDesk to remote desktop into the Raspberry Pi so we would no longer need to use the keyboard and monitor to access the Raspberry Pi. Any remote desktop service with FTP that works on Raspbian could be used.

In order to achieve autonomous driving, we used an open source software called Donkey Car[11]. As previously mentioned, it is open source and designed for scale cars. Alternatives such as AWS Deep Racer, OSOYOO Robot Car, and Yahboom could be used to achieve autonomous driving.

To use the Donkey Car software, we first had to drive the car around the track manually. As we drove the car, the software recorded images from the camera along with their corresponding servo angle and throttle value. This data, called training data, is then used to create a NN model. The neural network model will mimic the driving behaviors present in the training data, allowing the car to navigate the track without human input.

The Donkey Car software runs on python 3, so we needed to install the necessary dependencies in order to run the Donkey Car software. C, Java, and many other programming languages could be used to create a self-driving scale car, but Donkey Car is not compatible with these programming languages.

Figure 4.5 shows how all of the components connect together.

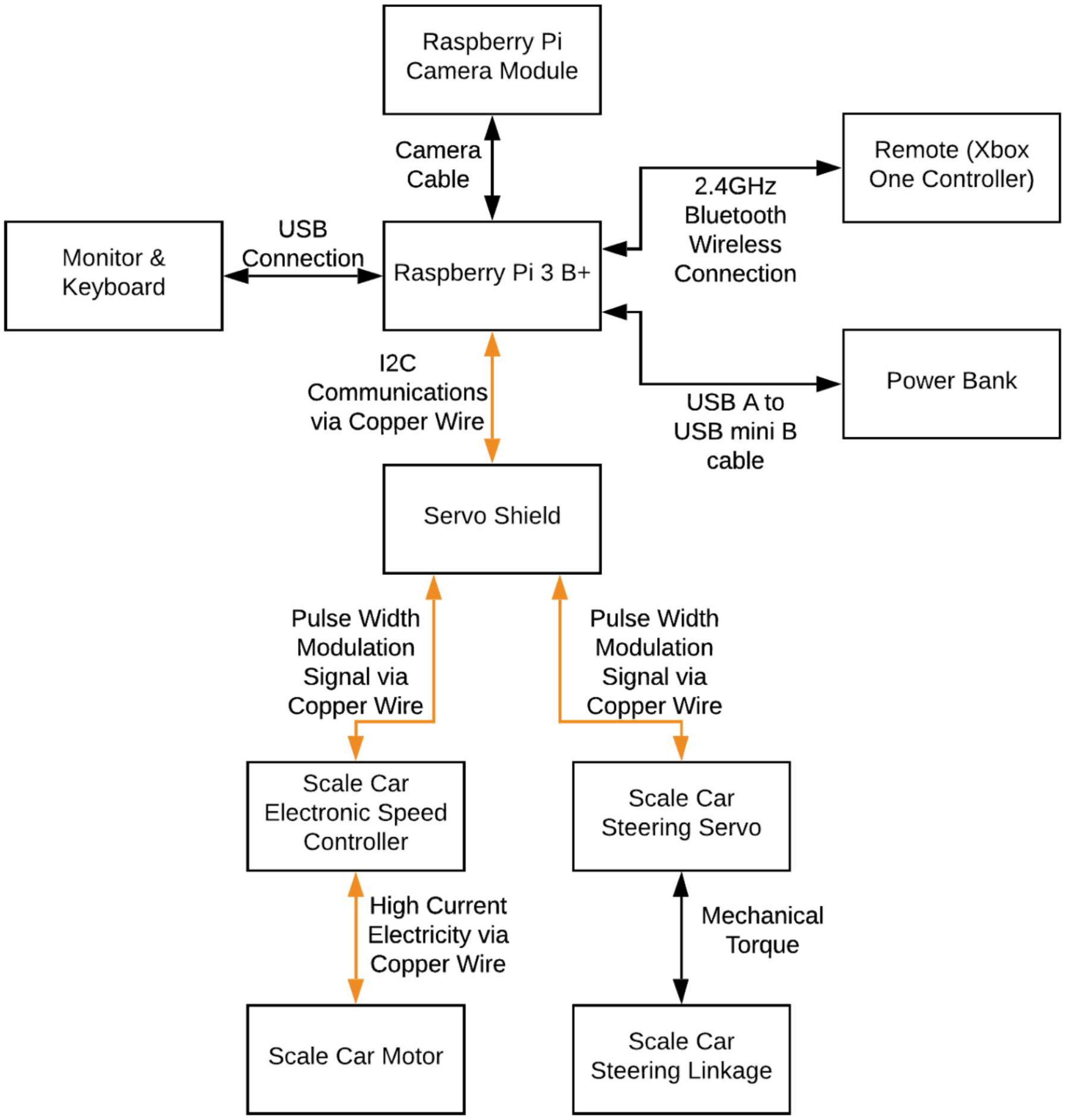

Raspberry Pi Camera Module
Camera Cable
Remote (Xbox One Controller)
2.4GHz Bluetooth Wireless Connection
Monitor & Keyboard
USB Connection
Raspberry Pi 3 B+
Power Bank
I2C Communications via Copper Wire
USB A to USB mini B cable
Servo Shield
Pulse Width Modulation Signal via Copper Wire
Pulse Width Modulation Signal via Copper Wire
Scale Car Electronic Speed Controller
Scale Car Steering Servo
High Current Electricity via Copper Wire
Mechanical Torque
Scale Car Motor
Scale Car Steering Linkage

Figure 4.5: Self-Driving Kit System Diagram

As you can see from our system diagram, the Raspberry Pi 3 B+ is the center of our self-driving kit. This is because it collects the training data and is then used to run the trained neural network models to self-drive scale cars. A monitor and keyboard are connected to the Raspberry Pi while setting up the software on the Raspberry Pi, but are disconnected for collecting training data and self-driving the scale cars. The power bank is an external battery that powers the Raspberry Pi independently from the battery used to power the scale car, adding to the self-driving kit's modularity. A remote (we mostly used an Xbox One controller) is used to drive the car during the training data collection process by sending bluetooth signals to the Raspberry Pi. The Raspberry Pi camera module is connected to the Raspberry Pi to collect images for training data and to collect images for the neural network to process while the car is self-driving. A servo shield is connected to the raspberry pi to control the steering servo and electronic speed controller (ESC) of the scale car, which in turn control the scale car's steering linkage and motor, respectively.

4.3 Training Data

Before we started testing our self-driving package, we had to learn how to collect training data. Training data is collected by manually driving the RC car around the track, during which time the Raspberry Pi is saving pictures from the camera on the front of the RC car along with the motor and servo levels, which control the car's speed and steering angle, respectively. This training data is then used to train a neural network model. When this neural network model is run on a car, the Raspberry Pi will take images from the camera and then process them through the neural network model to determine what speed and steering angle should be set, causing the car to self-drive. The neural network model will cause the car to try and mimic behaviors present in the training data, so the quantity and quality of training data used to create the neural network model significantly impacts the self-driving performance of the RC car. Therefore, to improve the likelihood that a neural network will perform well, we followed a procedure when collecting training data, and worked to improve that procedure with each test.

As a starting point, we used the Donkey Car website to come up with our first set of training data collection methods [7]. We drove the car in four different styles while collecting training data.

4.3.1 Driving Style 1: Accurate Laps

We called the first driving style “accurate laps”. For these accurate laps, when the user drives the car, they are most concerned about driving the car in the center of the track as accurately as possible, even if at a slower speed than when the user normally drives the car around the track. The theory behind these types of laps is that it will teach the neural network model that the car should try to stay in the center of the track, and that these accurate laps will provide a baseline for the car’s behavior. Donkey Car[11] recommends that these accurate laps make up roughly 10% of a training data set. ut, in practice, these accurate laps made up about 20% of our data.

4.3.2 Driving Style 2: Small Oscillation Laps

We called the second driving style “small oscillation laps”. For these laps, as the user drives the car around the track, make small oscillations around the center of the track. In other words, the user drives the car such that it zigzags left and right as it goes around the track, but the user makes sure the car stays mostly in the center of the track. The theory behind these laps is that this data will let the car see other parts of the track and learn to correct itself back to the centerline of the track. However, in practice, we found that the car would sometimes mimic this zig-zag behavior as it self-drove itself. To fix this, we eventually used a button on the controller we were using to drive the car when collecting training data to toggle when the car was recording training data by changing a setting in the Donkey Car[11] code. This way, we could control which of our driving behaviors were recorded as training data. For these small oscillation laps, we only recorded from when the car was closer to one of the edges of the track and drove back to the center of the track. This way, the car would not learn to deviate from the center of the track if it were driving in the center of the track. This second style of training data made up 20% of our total training data.

4.3.3 Driving Style 3: Extreme Laps

We called the third driving style “extreme laps”. For these laps, the car was driven back and forth between the extremes of the lanes. However, we were careful to not overcorrect the car. We would have the car stay on one side of the track, and then drive it to the other side of the track. The goal of this training data is for the car to learn how to drive back to the center of the track after driving off course. However, similarly to the small oscillation laps, we believe that this training data taught the car to drive away from the center of the track instead of training it how to drive back to the center of the track if it goes off course. So, when we eventually changed the system so that we could control when the car was recording training data, we made sure to only record the car driving back to the center of the track from when it was at the side of the track, or even when it was off of the track completely. This change significantly improved the reliability and consistency of our car's self-driving capabilities. This third style of training data made up about 30% of our training data.

4.3.4 Driving Style 4: Fast Laps

We called the fourth and final driving style "fast laps". For these laps, we would just drive the car around the track normally, but briskly. The goal of this training data is for the car to learn how to go around the track faster and not be too slow. This made up about 30% of our training data. Although the Donkey Car website recommended 5k records of training data, by our last test, we would try to get 20k records of training data before creating a neural network model.

4.4 Training Tracks

We tested our Self-Driving package on seven different tracks. Our first couple of tracks were made out of brightly colored tape in a classroom (Figures 4.6, 4.7, 4.8 and 4.9).

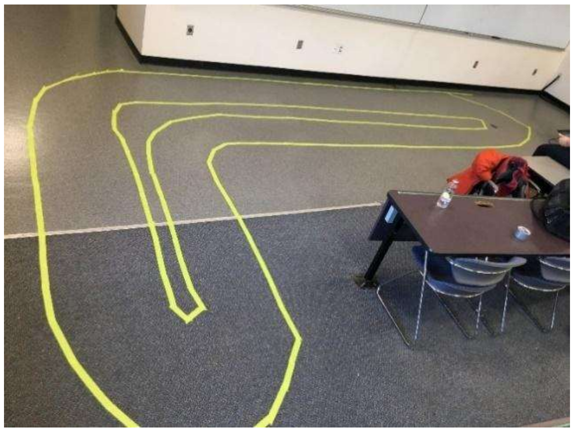


Figure 4.6: Picture of Track 1

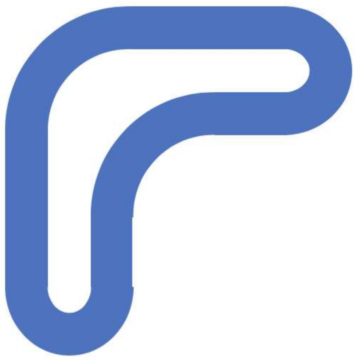


Figure 4.7: Schematic of Track 1

Figures 4.6 and 4.7 show the first track that we tested on. We made the track out of green tape, and it had two 90 degree turns and two 180 degree hairpin turns, so there would be different types of turns for the car to try and drive through.

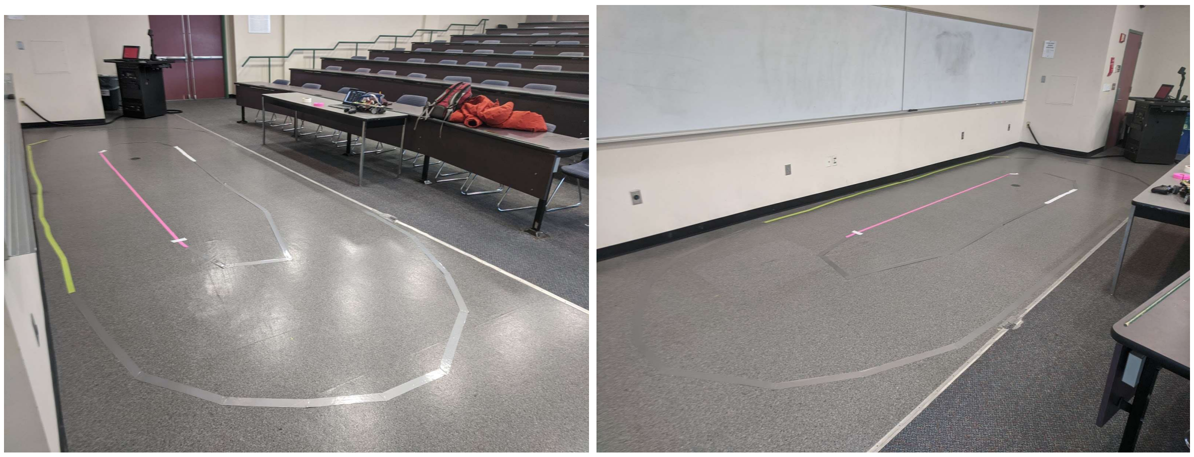


Figure 4.8: Pictures of Track 2

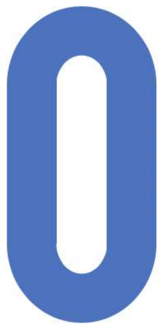


Figure 4.9: Schematic of Track 2

Figures 4.7 and 4.8 show the second track that we tested on. It was also made out of tape, but since we were running low on green tape, we used green tape, silver duct tape, some white tape, and some pink tape. We made the track to be a simple oval shape in hopes that the simple shape would increase the probability that we would be able to successfully train a neural network model on this track.

One of the biggest issues with our first two tracks is that we had to remove the track at the end of the day, as the track was in a classroom. This meant that we would have to collect training data, train the neural network model, and then upload that model to the car to self-drive it all in the span of one day. We wanted a track that could stay set up for long periods of time so we could practice collecting different training data for the same track, which would allow us to better analyze the best methods for collecting training data. It is important to note that the training data and neural network models that we made were fairly track specific (we couldn't take the training data from one track and use it to self-drive a car on a different track), so each time we created a new track, we had to collect new training data. This is another reason why we wanted to have a permanent location for us to set up our track to test our self-driving package. So, for our third track (Figures 4.10 and 4.11), we used a simple hallway as we could expect the hallway to relatively maintain the same appearance from day to day, providing us with a place to permanently train and test our self-driving package.

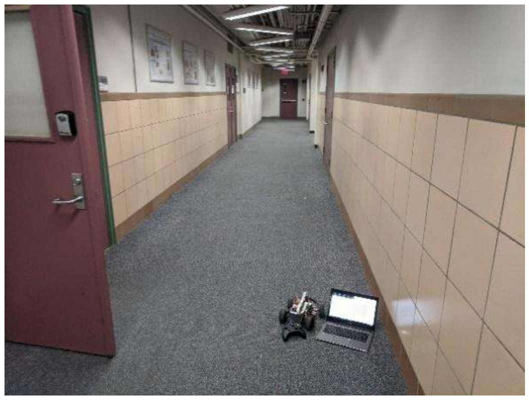


Figure 4.10: Picture of Track 3

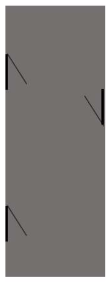


Figure 4.11: Schematic of Track 3

Figures 4.10 & 4.11 show Track 3. As previously stated, this track was just a simple hallway. This allowed us to test if a hallway could be used as a track to train self-driving cars.

Our tests on Track 3 were successful, so we decided to use a loop of hallways as our 4th track. Track 4 was made up of four 90-degree right turns and had the shape of a rectangle (Figures 4.12 & 4.13).

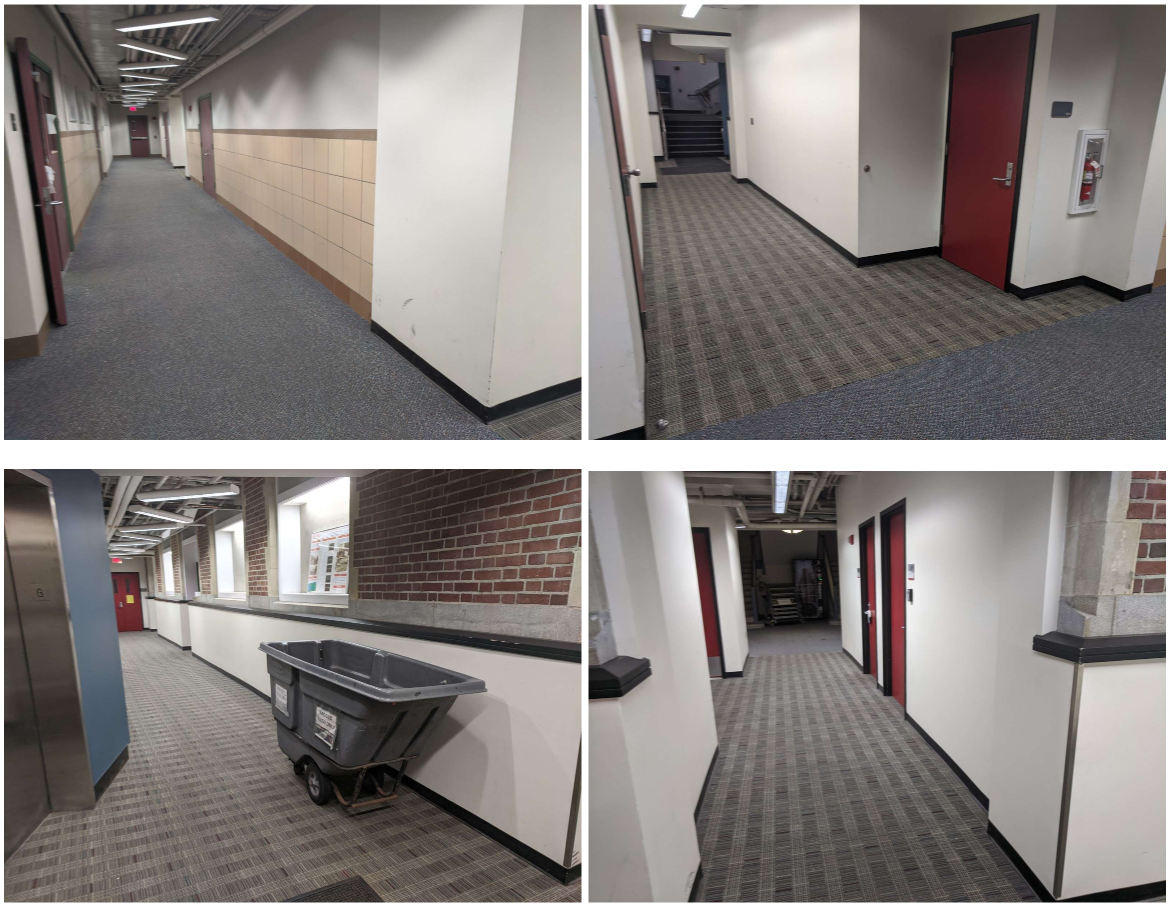


Figure 4.12: Pictures of Track 4.

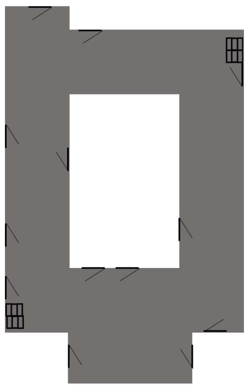


Figure 4.13: Schematic of Track 4

As you can see from Figures 4.12 & 4.13, Track 4 was a simple rectangle shape.

We made our fifth track using brightly colored flagging tape in the same hallways as Track 4 (Figures 4.14, 4.15 & 4.16). We theorized that with added visual cues for the boundaries of the track, the neural network model would be more likely to identify the boundaries of the track and stay within them.

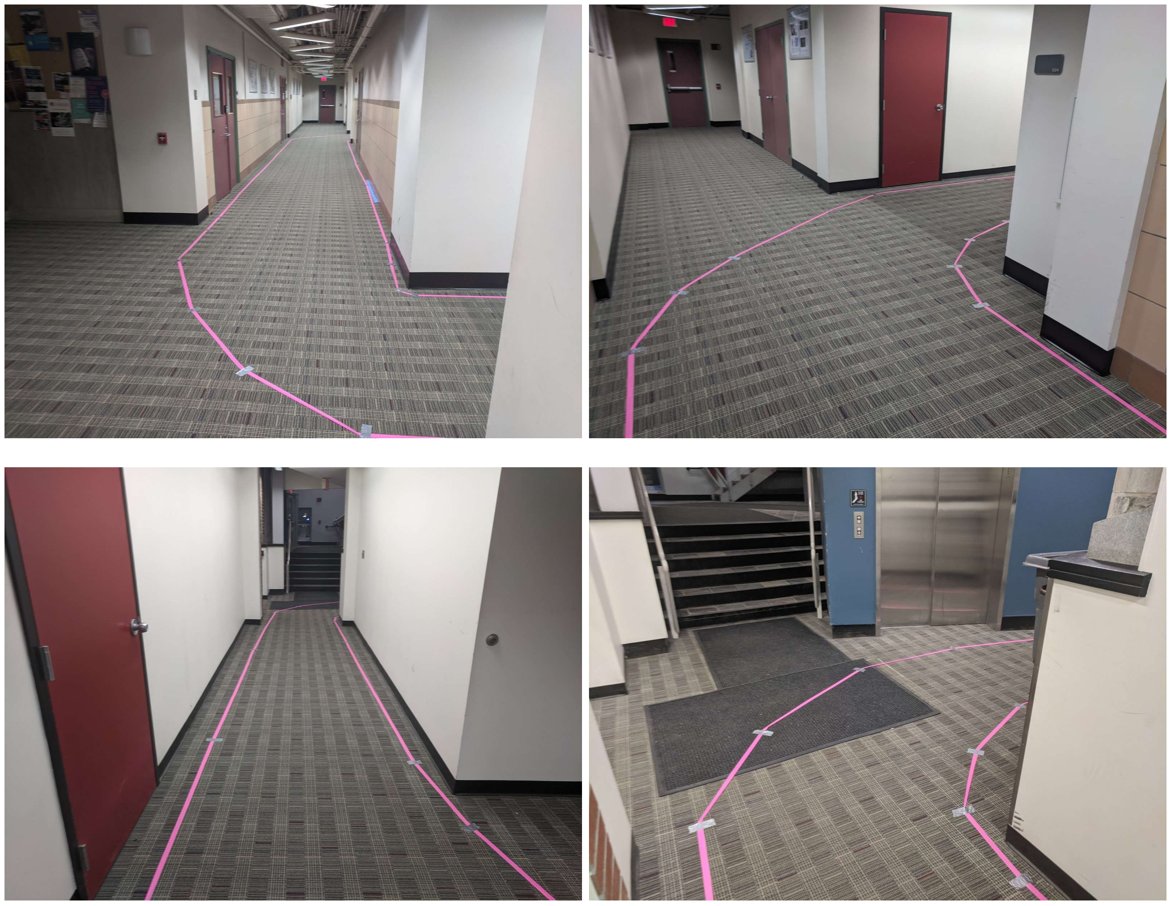


Figure 4.14: Pictures of Track 5

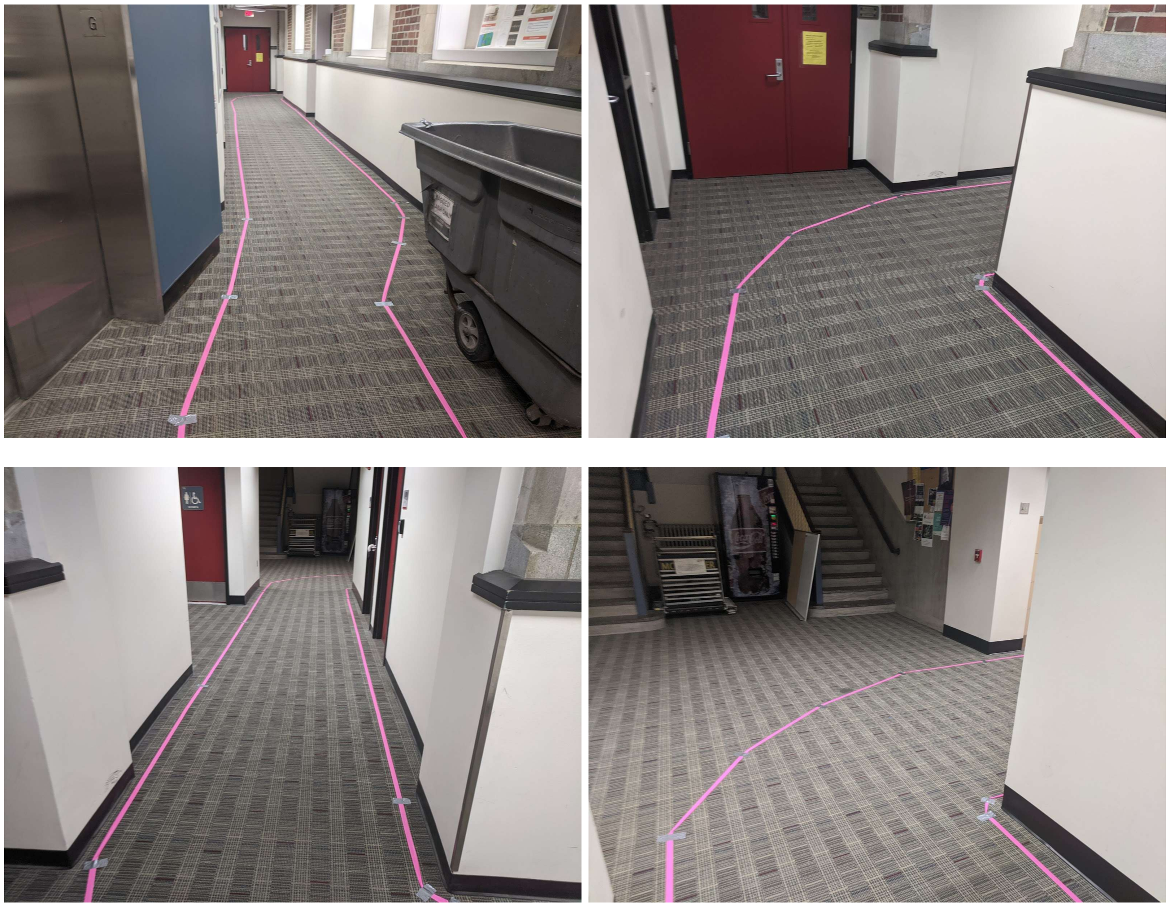


Figure 4.15: More Pictures of Track 5

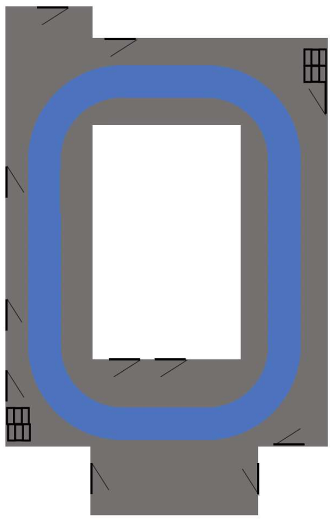


Figure 4.16: Schematic of Track 5

As you can see, Track 5 is in the same hallway as Track 4. It is made of four 90-degree turns, and uses pink flagging tape as the boundaries to the edges of the track. The track is more than five feet wide at some points, and as narrow as two and a half feet at other points.

After some tests, we modified track 5 to include a horseshoe-shaped turn to see if our self-driving package had the capability to deal with turns more complex than a 90-degree turn (Figure 4.17).

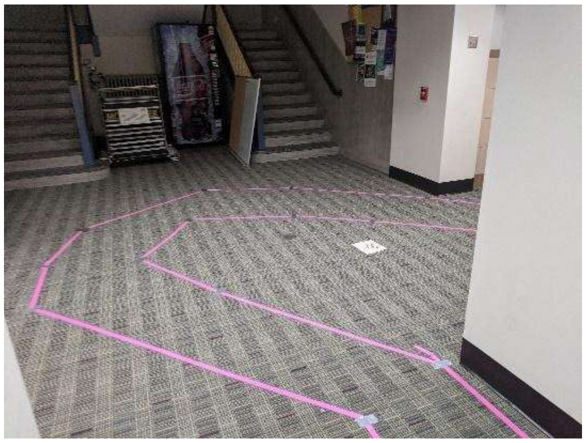


Figure 4.17: Horseshoe Turn Modification

Figure 4.17 shows the complex horseshoe turn modification that was made to Track 5. As you can see, for this turn, the car would have to turn left by about 45-degrees, make almost a 180-degree turn, and then turn left again by about 45-degrees.

For our sixth track, we decided to make a more complex track to try and push the limits of the self-driving package's capabilities. This track consisted of many turns and was also narrower when compared to previous tracks as we theorized that would help the neural network identify the boundaries of the track (Figures 4.18 & 4.19).

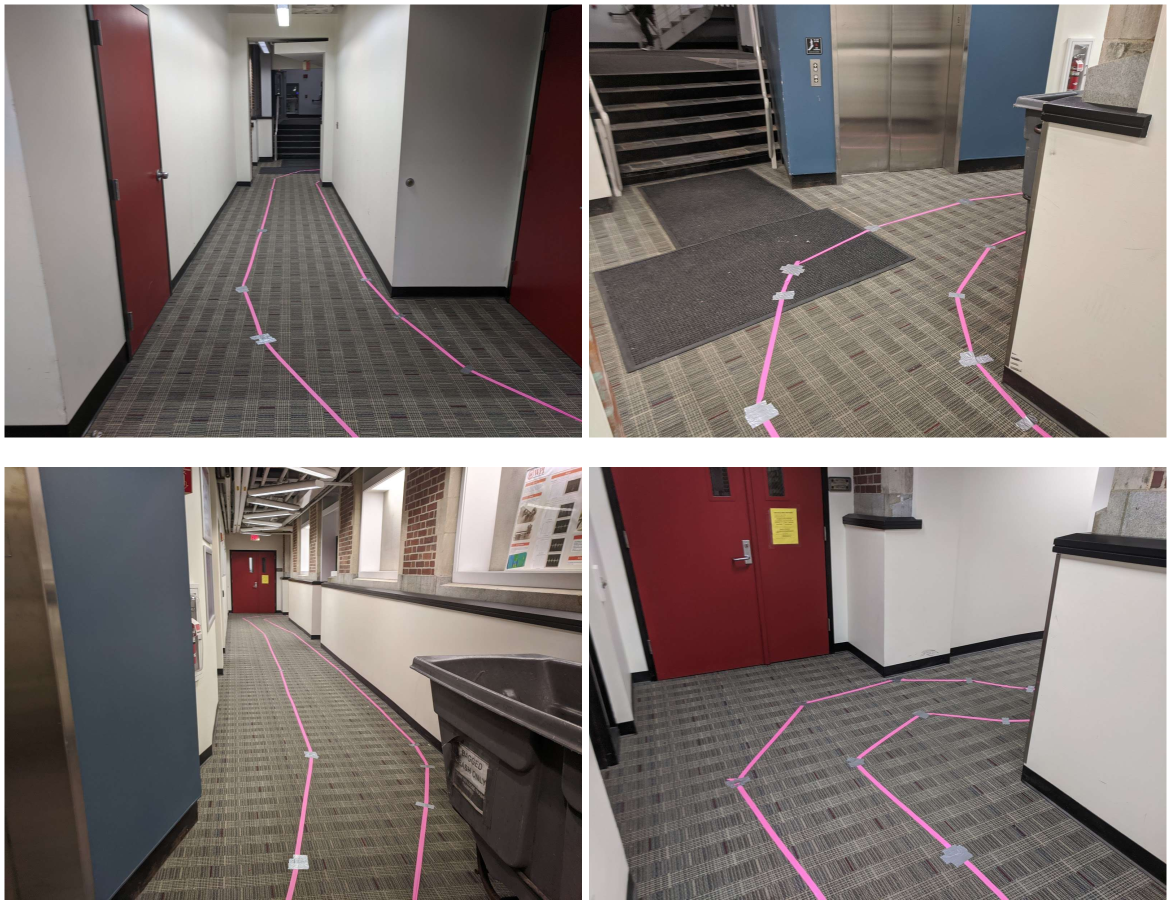


Figure 4.18: Pictures of Track 6

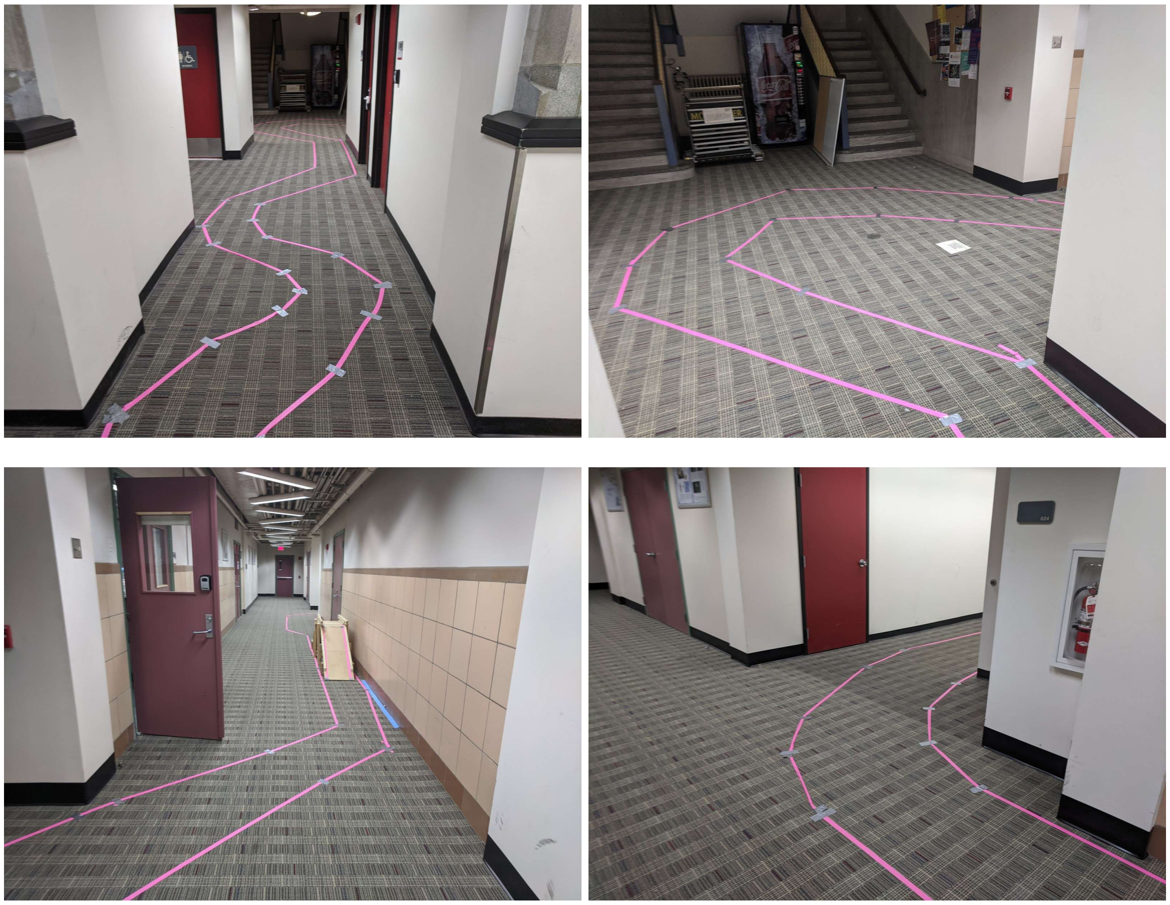


Figure 4.19: More Pictures of Track 6

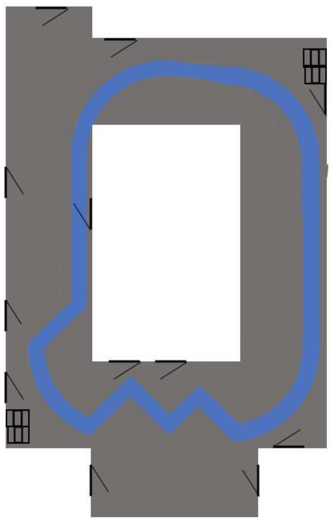


Figure 4.20: Schematic of Track 6

As you can see from Figures 4.18, 4.19, & 4.20, Track 6 was on the same hallway as for Tracks 4 and 5. Track 6 was generally a narrow track with many complex turns, especially with one section with an S-turn.

Unfortunately, we were unable to get a scale car to successfully drive around Track 6, so we made Track 7 to be similar to track 5, but it was three feet wide at all points of the track and had different color tape on the left and right sides of the track (Figures 4.21, 4.22 & 4.23). We theorized that by having a different color for the left and right boundaries of the track, the neural network model would correlate one color to turning left and the other color to turning right, which would in turn improve the reliability of the self-driving performance. Additionally, we had two horseshoe turns instead of one like we had in track 5 to see if the car's neural network could learn to navigate multiple complex turns on the same track.

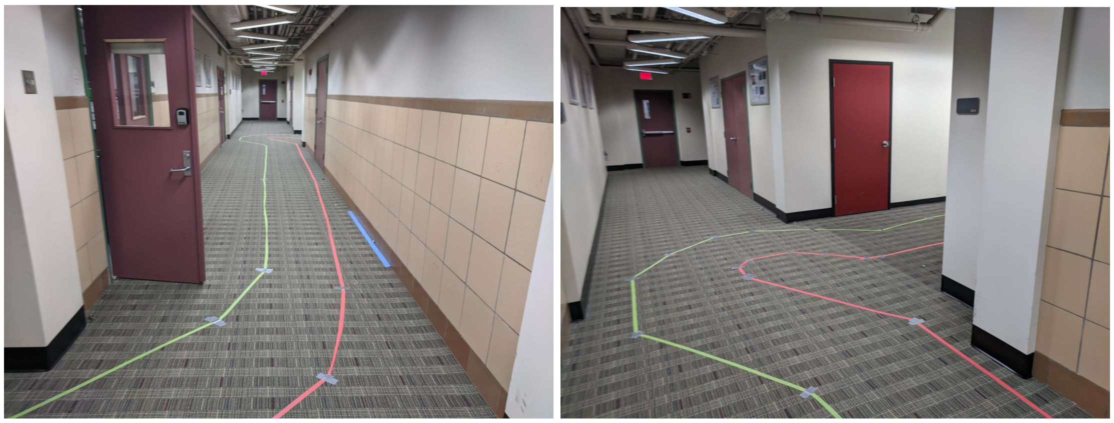


Figure 4.21: Pictures of Track 7

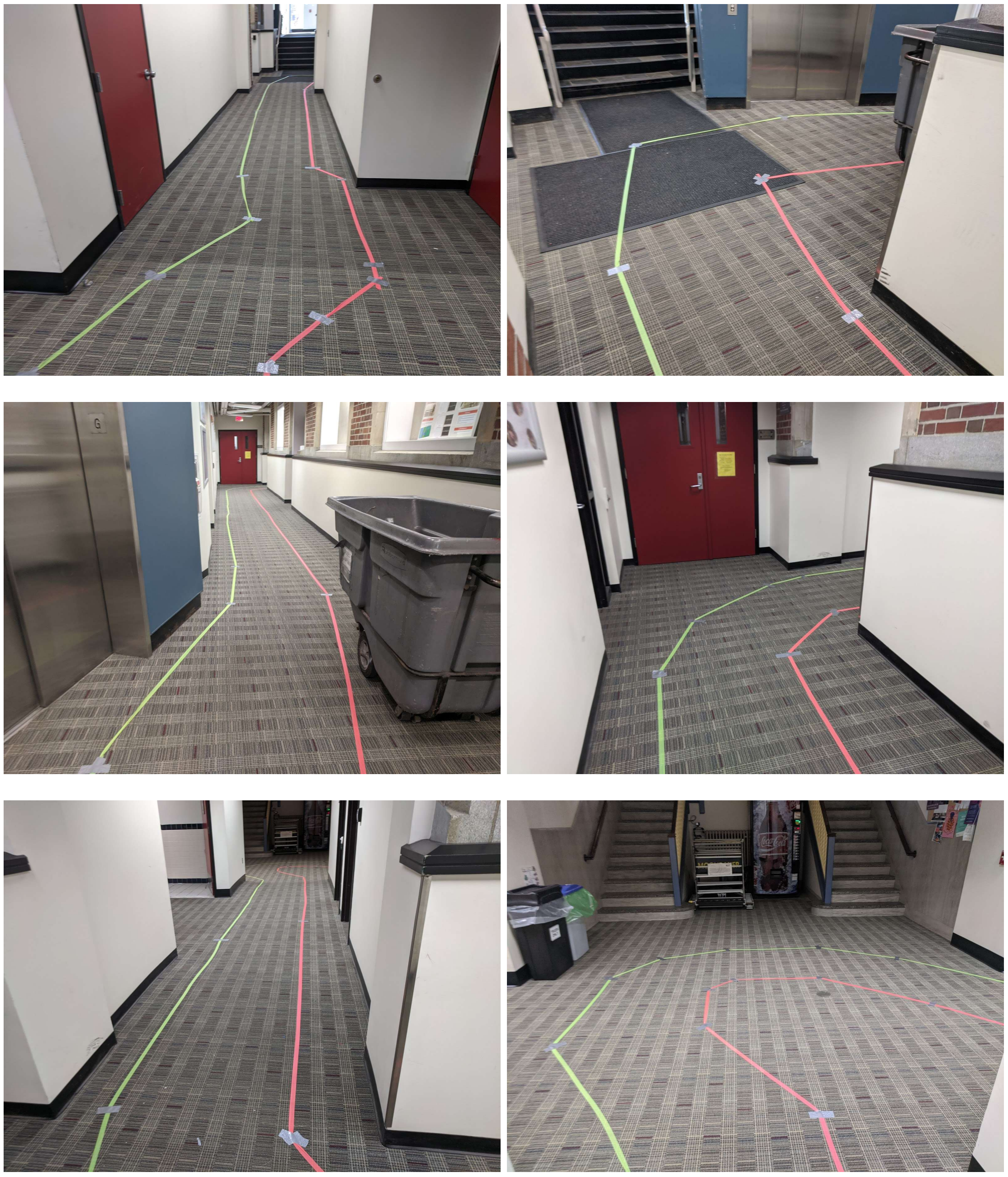


Figure 4.22: More Pictures of Track 7

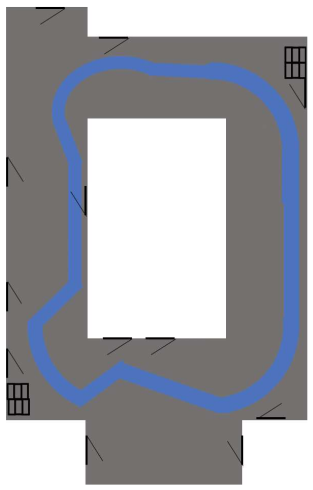


Figure 4.23: Schematic of Track 7

As you can see from Figures 4.21, 4.22 & 4.23, Track 7 was on the same hallway as for Tracks 4, 5, and 6. Track 7 was three feet wide at all points of the track, had two 90-degree turns, and two horseshoe turns, like the modification that was made to Track 5 (Figure 4.17).

Out of our seven tracks, our latter tracks were more complex than the earlier tracks. Our latter tracks started to feature horseshoe turns to test the limits of the self-driving kit.

4.5 Tests Performed on Self-Driving System:

4.5.1 Tests on Track 1

For our first test, we used the Mission D scale car (Figure 4.2) on Track 1 (Figures 4.6 & 4.7). Our main goal was to practice collecting training data. We drove the Mission D car around the track several times. We did some zigzags on the track, and had two examples of the car driving back onto the track if it ever went off the track. The biggest issue that we had with Track 1 is that the hairpin turn in the bottom of the track picture was a tight turn, and so we had to back up the car sometimes when we drove it through that turn. Additionally, since we were still new to the Donkey Car system, we had to disassemble the track before we could finish training the neural network. So, we were unable to test trying to have the car self-drive around Track 1 for this first test.

4.5.2 Tests on Track 2

For our second test, we used the Mission D scale car (Figure 4.2) on Track 2 (Figures 4.8 & 4.9). Our goal with this test was to collect training data, create a neural network, and have the Mission D car drive around the track by itself a few times. For this second test, we followed the instructions on how to properly collect training data. We were able to get the Mission D car to go around Track 2 a few times, but then the car stopped driving and we were unable to get the car to self-drive again using the same neural network. For this first successful model, we only trained the car using 5,400 records of training data, which took about an hour and a half to make the neural network model, and we hypothesized that we did not have enough records of training data for the car to reliably identify the track and self-drive around the track.

4.5.3 Tests on Track 3

For tests on Track 3(Figures 4.10 & 4.11), we tried collecting training data and training a student-made car (Figure 4.3) from WPI's Advanced Engineering Design course. The test on Track 3 was mostly a proof-of-concept test; we wanted to see if the hallway was even a viable track for us to train and self-drive cars on. We also wanted to see if it was possible to train a student-made car to self-drive. So, we collected training data for driving the car halfway down the hallway. After collecting 2,700 training data records, we trained the neural network model, which took a little under an hour, uploaded the model to the car, and it self-drove. It drove down the section of the hallway that we had trained it on, and then continued driving past where we had trained it until it eventually crashed into a wall.

4.5.4 Tests on Track 4

After successfully demonstrating the ability for a scale car to be trained to self-drive down a hallway, we decided to have Track 4 (Figures 4.12 & 4.13) be a rectangle. Our first three tests on Track 4 were all done on the student-made car (Figure 4.3). The third test on track 4 was a success as the car was able to go around the track twice without crashing into a wall or veering off of the track. This model required over 30,000 training data records and took about two to three hours to train the model.

We did a fourth test on track 4, this time with one of the 3D printed cars (Figure 4.4). This car was a lot slower than the student-made car we had previously used (Figure 4.3), which we theorized would make it better for self-driving as the neural network would have more time to react to an obstacle (like a wall) if the car was veering off course. We collected 17,000 training data records, but when we made the model and tried to use it to self-drive the car, the car was only able to make it about halfway around one lap before it ran into a wall.

4.5.5 Tests on Track 5

For our tests on Track 5 (Figures 4.14, 4.15, & 4.16), we increased the amount of training data that we collected as we hypothesized that a lack of training data for our fourth test on track 4 is what caused that model to be unsuccessful. Again, we used a 3D printed car (Figure 4.4) for these tests. Our best neural network model on track 5 was trained on 40,000 records, which took 2.5 hours to train, and the car was able to self-drive for four and a half laps, which met our goal of having the car self-drive for four laps. We then modified track 5 to include a horseshoe-shaped turn to see if our self-driving package had the capability to deal with turns more complex than a 90-degree turn (Figure 4.12).

We were able to create a neural network model that caused the car to self-drive around this modified Track 5 for 6.5 minutes straight without human intervention. This model used 35,000 records of training data and took 3 hours to train. Since the track is wide, we split up our training data into thirds, one third in the center of the track, one third on the right hand side of the track, and one third on the left hand side of the track. We hypothesized that since the track was wide, giving the neural network data from different positions in the track would let it better identify the boundaries of the track and learn behaviors to stay within these boundaries.

4.5.6 Tests on Track 6

We ran a single test on Track 6 (Figures 4.18, 4.19, & 4.20) using a 3D printed car (Figure 4.4). We collected 60,000 records of training data, which took over three hours to train a neural network model, and when put on a car, the car was not able to self-drive. The car was only able to go around the track for one lap before hitting a wall and would sometimes drive outside of the track boundaries. We took this to mean that the track was too narrow, so we decided to make our next track wider.

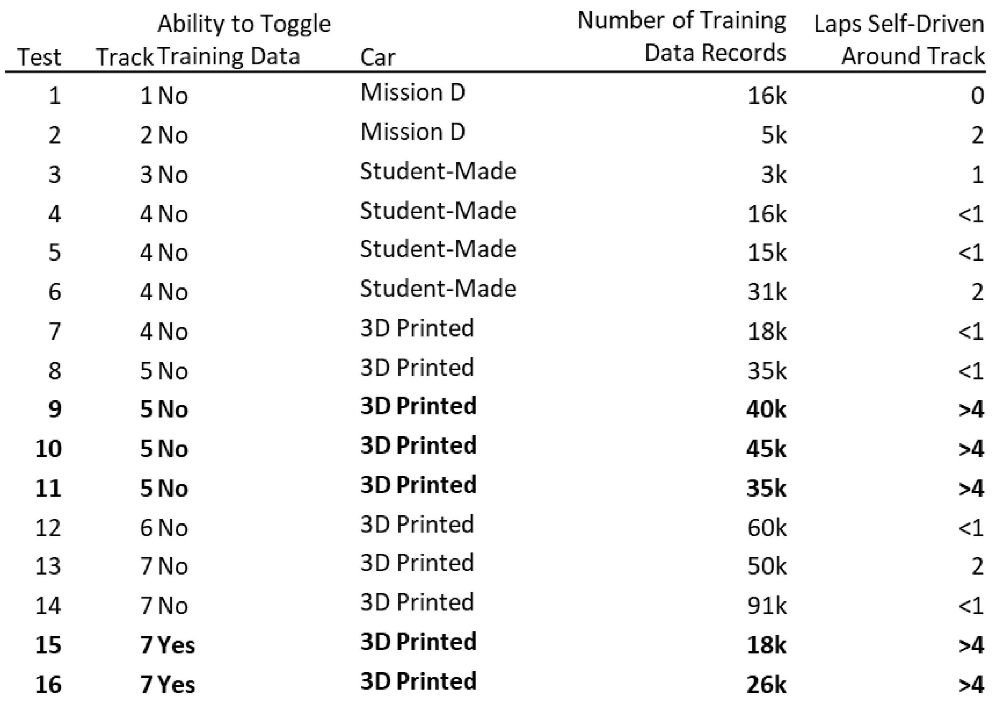
4.5.7 Tests on Track 7

On Track 7 (Figures 4.21, 4.22, & 4.23), we performed tests using a 3D printed car (Figure 4.4). Our first two tests on Track 7 were unsuccessful, even with collecting over 90,000 training data records and training the neural network model for over five hours. The car would miss turns and was unable to self-drive around the track for four laps. This is when we set a button on the remote that we were using to drive the car when collecting training data to toggle when the car was recording training data. After implementing this change, we collected 18,000 training data records, which took one and a half hours to train a neural network model. The car was able to self-drive for over four laps, which was our goal, with the smallest amount of training data records.

4.5.8: Summary of Tests

We tested our self-driving kit on seven different tracks, and were able to get the total number of training data records required to make a neural network that can self drive down to 18,000 records. Table 4.2 summarizes all of the tests that we did.

Table 4.2: Table of Self-driving Package Tests



Test	Track	Ability to Toggle Training Data	Car	Number of Training Data Records	Laps Self-Driven Around Track
1	1	No	Mission D	16k	0
2	2	No	Mission D	5k	2
3	3	No	Student-Made	3k	1
4	4	No	Student-Made	16k	<1
5	4	No	Student-Made	15k	<1
6	4	No	Student-Made	31k	2
7	4	No	3D Printed	18k	<1
8	5	No	3D Printed	35k	<1
9	**5**	**No**	**3D Printed**	**40k**	**>4**
10	**5**	**No**	**3D Printed**	**45k**	**>4**
11	**5**	**No**	**3D Printed**	**35k**	**>4**
12	6	No	3D Printed	60k	<1
13	7	No	3D Printed	50k	2
14	7	No	3D Printed	91k	<1
15	**7**	**Yes**	**3D Printed**	**18k**	**>4**
16	**7**	**Yes**	**3D Printed**	**26k**	**>4**

As you can see from Table 4.2, we had five tests where we were able to create a neural network model that would self-drive a scale car around a track four or more times. The number of records required for these successful neural networks significantly reduced when we started using a button to toggle when the scale car was recording training data. This makes sense, as this toggling ability gave us more control over the behaviors that the neural network models would learn.

4.6 User Guide and Manuals

After testing our self-driving kit on seven different tracks, we made a user guide on how to use our self-driving kit. The purpose of this user guide is for students in the ME 4320 and ME/RBE 4322 courses to be able to use the user guide to set up their own self-driving kit and use it to make their scale-cars self-drive. Therefore, the user guide had to be clear, concise, and require only a reasonable background in college-level Mechanical Engineering. The guide had to be clear in the sense that if it were confusing, students would not be able to follow the user guide. The guide had to be concise in the sense that the total process could not take too long as the students would have a limited amount of time to set up the self-driving kit. The guide could only require a college-level Mechanical Engineering background as that would be the target audience of the user guide.

Our first step to making the guide was ordering the components to construct a second self driving kit. After ordering the components, we took detailed notes of the process that we followed to construct the second self-driving kit. We then formalized these notes into a user guide.

Next, we gave the user guide to two students, along with all of the components of the self-driving kit. We took detailed notes of the issues that they ran into while setting up and using the kit. We also asked them for feedback on what steps they found to be hard and what issues they felt they ran into, and other areas of the user guide that could be improved. We then used these notes and feedback to modify the user guide. Now, our self-driving kit has an accompanying user guide for students in the ME 4320 and ME/RBE 4322 courses to implement the self-driving kit. The user guide can be found at http://tiny.cc/MQPUserGuide

4.7 Self-Driving Implementation Summary

In this chapter, we discussed the self-driving kit development process, we reviewed the cars that we tested our self-driving kit on, and went over the components required to make our self-driving kit. Then, we covered how we collected training data when using our self-driving kit, tracks we tested our self-driving kit on, and the various tests that we performed on our self-driving kit. Lastly, we went over how we made the user guide for our self-driving kit.

In the next chapter, we will discuss the development of our sensor kit and the integration of the sensor kit with the self-driving kit.

5 Sensors Implementation and Combining Self-Driving & Sensor Kits

In this chapter, we will discuss the implementation of the sensor package, how we combined the self-driving and sensor packages, and the further development of our user guides.

To meet our project goal, we had to make a sensor package and a user-guide on how to use the package. As per our objective on the modular sensor kit, we needed to determine what sensors should be used for data collection, as well as how we should go about implementing the sensors in an easy to understand and follow manner. This was accomplished with an Ardiuno Mega and RasPi; both fairly common and simple to use pieces of hardware.

For our sensor package, we wanted to provide students with various sensors that could record useful data about the function of their scale cars. To do this, we had to wire various sensors into an Arduino, and then program that Arduino to send the data to a Raspberry Pi where the data could be saved to a useful filetype.

A sensor user guide was developed by streamlining the process we used to create the package. We then tested this guide on two Mechanical Engineering graduate students so we could evaluate how effective our guide was and what changes needed to be made. As seen below in Figure 5.1, we developed a plan to implement sensors, and verify that the sensors are functioning appropriately.

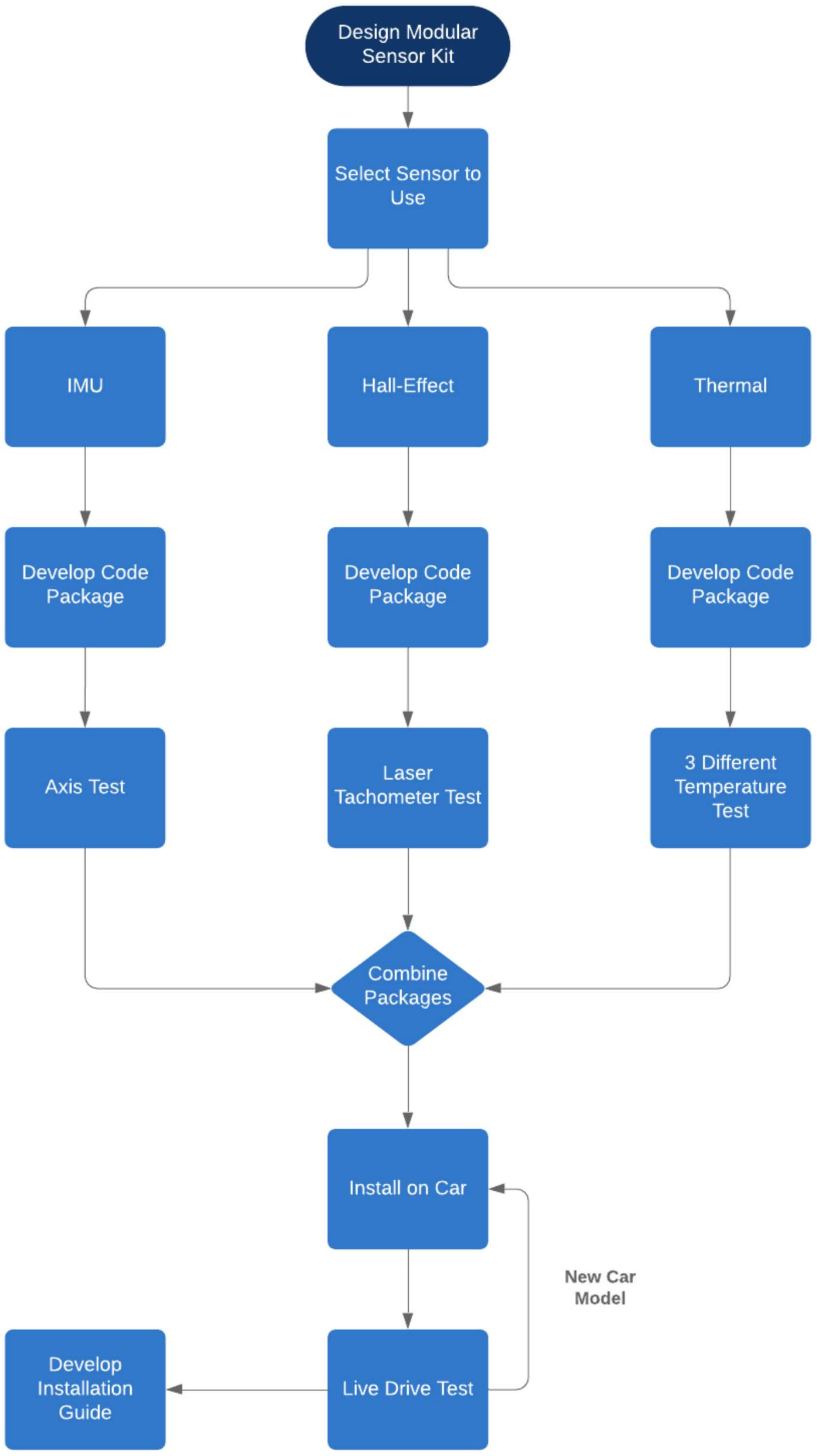


Figure 5.1: Flowchart of Project Plan

The first step for developing a sensor kit is determining what sensors are needed. There are only so many valuable pieces of data you can collect from a car, so we determined that we wanted to collect the roll, pitch, and yaw of the car, from an IMU; a hall-effect sensor, to determine velocity of the car; and a thermal sensor to determine component temperatures and the ambient temperature of the surrounding area. There are many resources on the internet on how to implement sensors, so utilizing our own knowledge, and internet resources, we got the three sensors we selected working. To verify the sensors are working accurately, we developed tests comparing our sensors to commercially available products which mesease the same data. There are far more details on this process below in section 3.2. After the sensors were verified to be working, we installed the package on a RC car, and did a live test drive. This worked flawlessly, and by installing the module on a car, we were able to more successfully create the user guide for ME4320 students to utilize in their course.

5.1 Sensor Module

The sensor package developed by the team consists of a variety of sensors; there is a hall effect sensor to determine the velocity of the car, thermal sensors to monitor component temperatures, and an IMU to record the roll, pitch and yaw of the car. Additionally, there is an Arduino Mega and a Raspberry Pi to record the sensor data and save it to a CSV file so it can be analyzed later. An exterior casing was designed and printed as an enclosure to house the electronics, equipped with mounting holes and an external portable battery so that the sensors can be powered independently from the car, increasing the modularity. See, the package functions as an independent unit that can be mounted on any car that has the minimal space for it. The sensors are also designed to be easily mounted and changeable, such that users can relocate their sensor locations between testing. As seen below in Figure 5.2, this is the complete wiring diagram for the Arduino and sensors.

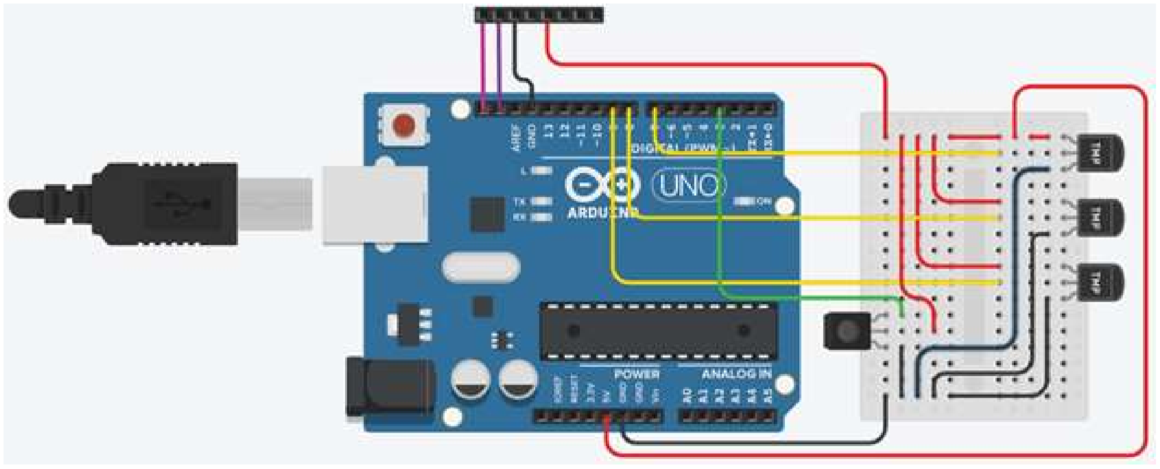


Figure 5.2: Arduino and Sensor Wiring Diagram

The package was designed to be of simple construction for application in the ME 4320 class. Sensors were chosen with priority being compatibility with Arduino and ease of installation, as the idea was not to over-complicate the sensor system.

5.1.1 Sensor Testing

With each individual sensor that was added to the package, there were multiple tests performed on each sensor before it was incorporated fully. Each sensor was subjected to both a verification test and an accuracy test in order to demonstrate the relative functionality of each sensor as it was being added into the design. The verification test was a much more binary test, being conducted immediately following setup of the sensor in order to ensure that the device was functioning within a reasonable range before moving forward with accuracy verification. Once the selected sensor was verified to be worth testing, the test evolved from a verification test to an accuracy test, in which the electronic sensors were tested against various other instruments to identify how accurately the data was being detected. Of course, with each sensor, a different test needed to be designed.

5.1.1.1 Thermal Sensor Testing

For the thermal sensors, each device was individually verified using simply the thermostat in the room in order to verify the temperature and the currently unused feature of humidity. With the device connected into the Arduino script, the data collection script was run in order to store the temperature as perceived by the sensor. This number, averaged over the five minute it was held operational, was then compared against the temperature as displayed by the thermostat of the room. When comparing the five-minute average to the thermostat setting, a sensor was considered verified if the average were within the range of tolerance for the thermostat itself, being about plus or minus a degree. So, with the temperature of the room set to be 20°C, the target of verification was in the range of 19°C-21°C. Once a thermal sensor was observed to report data in the appropriate range, it would then move on to accuracy testing.

The accuracy test for the thermal sensors involved the use of a known, constant and controlled heating source as the baseline, with and having the thermal sensors simultaneously run for an extended period of time and compare the recorded values as such. Below, in Tables 5.1, 5.2, and 5.3, are the average temperatures as recorded by the sensors over the testing period of five minutes, the temperature that the constant heating source was set to, and the average accuracy of the sensors during the test in registering the correct temperature. The full data from this test can be found in Appendix 10.7.

Table 5.1: Thermal Test 1: Temperature at 20°C (Room Temperature)

Sensor Number	Sensor Average Temperature (°C)	Source Temperature (°C)	Accuracy (%)
1	20.5	20	97.4
2	20.5	20	97.3
3	20.5	20	97.4

Table 5.2: Thermal Test 2: Temperature at 35°C (Warm Operating Temperature)

Sensor Number	Sensor Average Temperature (°C)	Source Temperature (°C)	Accuracy (%)
1	35.4	35	98.7
2	35.4	35	98.7
3	35.5	35	98.4

Table 5.3: Thermal Test 3: Temperature at 85°C (Critical Temperature)

Sensor Number	Sensor Average Temperature (°C)	Source Temperature (°C)	Accuracy (%)
1	85.6	85	99.2
2	85.7	85	99.2
3	85.8	85	99.1

The baseline temperatures of 20°C, 35°C and 85°C were chosen as specific targets: 20°C being average room temperature, 35°C being an estimated operating temperature of the car in extreme cases, and 85°C is the reported temperature at which the Raspberry Pi is said to shut-down or suffer operational damages as a result of heat. With these three sensors tested at these temperatures, we were confident that the sensors would be more then well equipped at their target of monitoring temperatures. The goal of these sensors is not to be constantly measuring high temperatures, but rather, to be able to make note of potential high-temperature hazards. The heating source temperature is verified by its own internal thermocouple used to monitor the heating source's temperature. Further, temperature data in the future would be truncated to the nearest whole degree, as the decimals are more a product of electrical noise than they are a representation of accuracy for temperature..

5.1.1.2 IMU Testing

For the testing of the IMU, again a preliminary verification test was performed prior to moving forward with more formal testing. The verification test involved establishing x-, y-, and z-axis using these to observe pitches, rolls, and yaws in varying directions. After establishing our axes, we fitted the IMU into a fixed location inside of the sensor box module. Once fixed, the data was recorded as the box was rotated 90° in each direction, standing on each exterior face of the box. This test was performed that the IMU would respond in the desired directions and would have corresponding changes in angle measurement in the data provided. The IMU tested to have an accuracy of 98% to the expected rotations. However, something to note is that the IMU has been coded specifically to only report angle measures of a whole degree, in the range of [0, 359]. For the purposes of calculating the average IMU reading, the rotation period recording was set to be 10 times, once every three seconds, for thirty seconds after each rotation to ensure IMU had time to settle before being recording data. Below in Table 5.4 are the results of this axis verification, demonstrating how well the IMU performed. The full data from this test can be found in Appendix 10.8.

Table 5.4: IMU Individual Axis Rotation Test Data

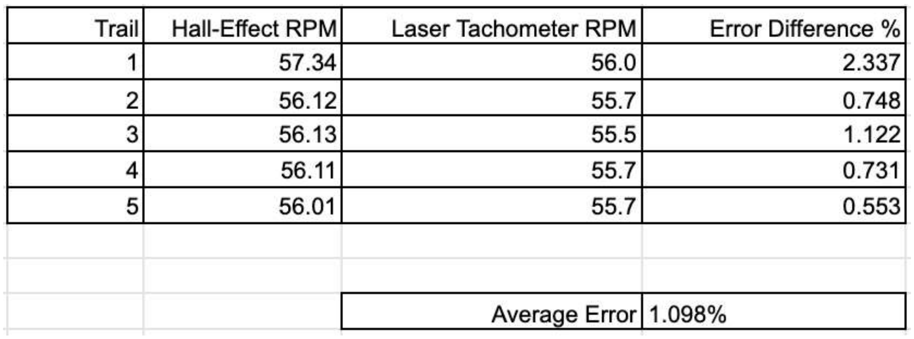
X-Axis Rotation		Y-Axis Rotation		Z-Axis Rotation	
IMU Recording	Actual Rotation	IMU Recording	Actual Rotation	IMU Recording	Actual Rotation
0	0	0.1	0	-0.2	0
89.8	90	89.9	90	89.5	90
180.1	180	179.6	180	180.4	180
269.2	270	269.9	270	269.5	270
0.2	0	0.1	0	0.7	0

5.1.1.4 Hall-Effect Sensor Testing

The hall-effect sensor is the last sensor tested and implemented into the sensor package. However, with the structure provided by the user manual and accompanying testing procedures, there is room for students in the ME 4320 to branch out and include more sensors.

The hall-effect sensor was originally verified using a basic electric motor, a magnet, and the sensor. Having the electric motor set to run at 60 RPM, the magnet was glued to a wheel attached to the motor, and the hall-effect was held firmly in place such that the magnet was on path to pass by the sensor, as it would be in practice on the car. Once it was verified that the hall-effect sensor was able to keep passing with the motor at roughly 60 RPM, the test moved from verification to accuracy. By refixing the setup in front of a laser tachometer, the hall-effect could be compared in real time to what was being detected by a laser tachometer. The results of the test below, shown in Table 5.5, calculate an accuracy of 98.902%.

Table 5.5: Hall-Effect Sensor Test Results



Trail	Hall-Effect RPM	Laser Tachometer RPM	Error Difference %
1	57.34	56.0	2.337
2	56.12	55.7	0.748
3	56.13	55.5	1.122
4	56.11	55.7	0.731
5	56.01	55.7	0.553
		Average Error	1.098%

Each of the sensors was originally tested individually before being connected on the car to be run for a live data run test. During the live data test, the car would be driven in a straight line with all sensors recording, stopping to have the exterior temperature sensor disconnected intentionally during operation, before continuing driving over a small-scale speed bump to purposefully disrupt the car and tilt it in its position while driving. The bump was designed to bump the car roughly 22° in the x-direction based on the collision path of the car. The temperature sensor was intentionally disconnected to demonstrate that the script will not quit if something disconnects or goes wrong during operation. The data from this test can be viewed in Appendix 10.3, and a picture of the car driving over bumps can be seen below in Figure 5.3.

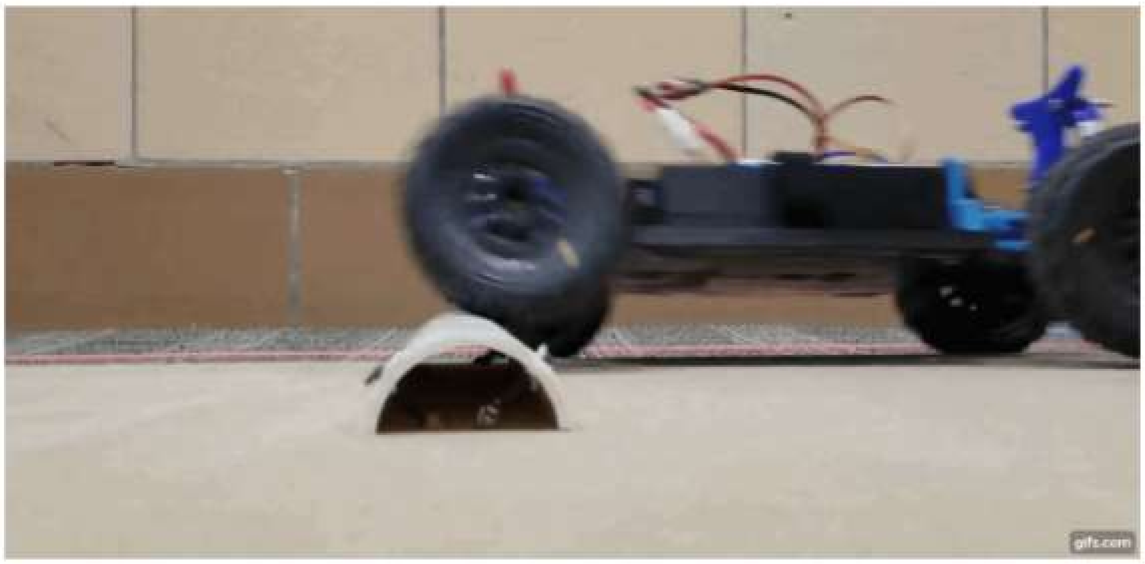


Figure 5.3: 3D Printed Car Driving Over a Bump

The live drive test was able to demonstrate that our car can operate with a variety of sensors installed and can record and save accurate data in a CSV file for future analysis. The car was able to detect the change in the x-direction caused by the bump in the road through the CSV, was able to monitor the interior temperatures and exterior temperatures of the car, was able to record the RPM of the car during operation and was even able to continue operation after one of the sensors was disconnected from the package.

5.2 Integration of Self-Driving and Sensors into a single kit

In order to have uniformity in the implementation of both the self-driving and sensor systems into the Advanced Engineering Design course, the team took upon themselves the task of designing and manufacturing a containment box to hold all the electronics necessary in a well laid out and organized way. The kit is fully 3D printed and does not require any support material to print successfully, or hardware to assemble. This aids to the systems modularity; anyone with a 3D printer and basic computer skills can fully implement our self-driving and sensor packages. There is a need for two M2.5 bolts if one wants to utilize the internal mounting holes for the IMU and thermal sensor, as a user would want these sensors mounted as rigidly as possible.

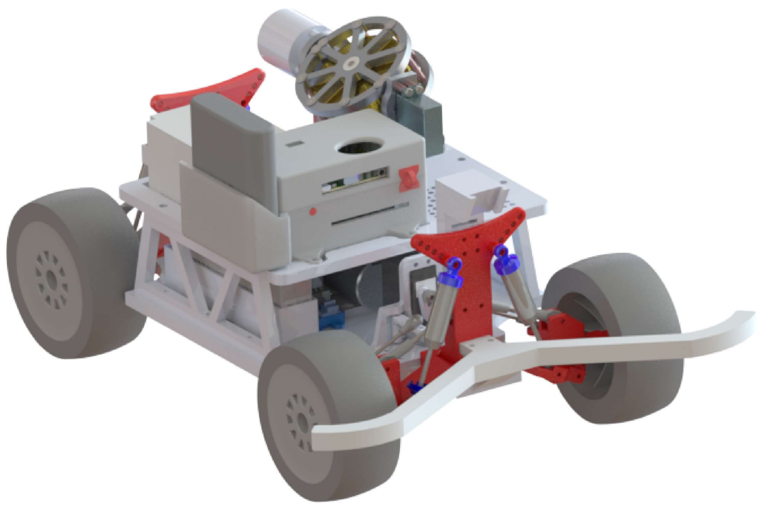


Figure 5.4: Self-driving and Sensor Kit Mounted on Rc Car

As seen above in Figure 5.4 the kit has four integrated mounting holes located on the bottom surface which allows the users to early mount the kit to their car. This example in Figure 5.4 is a fully 3D printed RC car provided to us by another project team at Worcester Polytechnic Institute.

5.3 User Manuals

When the team was creating the sensor package for the ME4320 course, it was imperative to create a user guide so that any student, regardless of experience levels, could begin the process of assembling this sensor package. We developed a user guide, full of detailed images and instructions so that the steps can be followed by most students with little effort. As previously mentioned, we gave an earlier version of the user-guide to two Mechanical Engineering graduate students, who were able to follow the given user guide and end up with their own self-driving RC car. Now, students in the Advanced Engineering Design course can use our self-driving package to implement their own self-driving systems on the scale cars that they build. The guide also includes a list of required hardware, wiring diagram, and short tutorials on how to install and operate the required Arduino and Raspberry Pi code for the sensor package. The sensor user guide can be found in Appendix 10.2, Arduino Code can be found in Appendix 10.5, and the RasPi code can be found in Appendix 10.6.

5.4 Summary

In this chapter, we discussed the implementation of the sensor package, the combination of the self-driving and sensor packages, and the further development of our user guides.

In the next chapter, we will discuss the work we did to aid our sister MQP team with the construction of Testing Rigs.

6 Creating Testing Mechanisms for Subcomponent Testing

In this chapter, we will discuss the work we did to aid our sister MQP team with the construction of Testing Rigs.

Our project team worked closely with our sister MQP team to assist them with the creation of testing rigs. This was primarily used for the teams steering linkages, and can be seen below in Figure 6.1. The test rig was set up so that a front suspension can be mounted to a base plate.

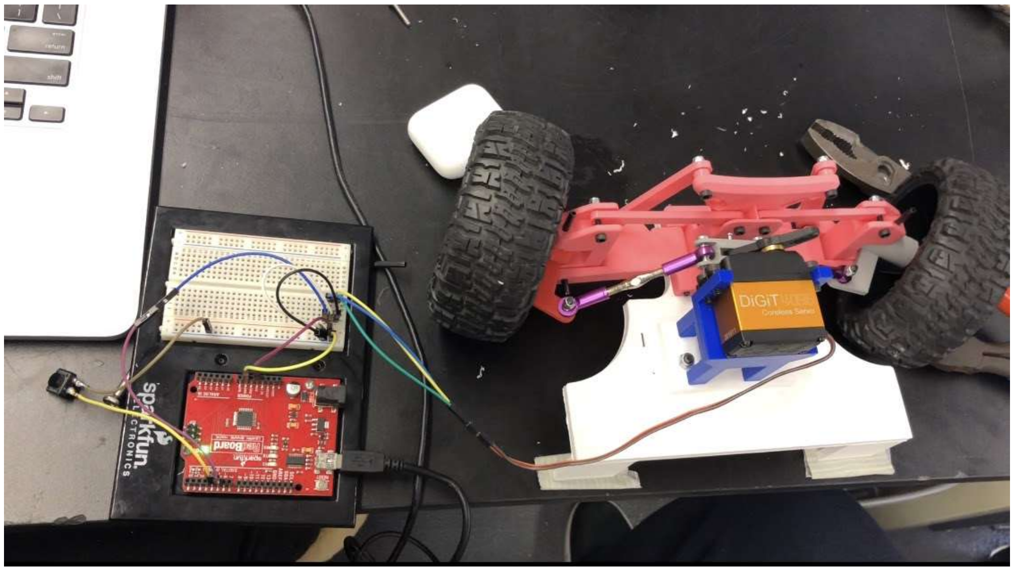


Figure 6.1: Steering Linkage Testing Platform

To run the steering test, a simple Arduino script (Appendix 10.4) which was created turned the wheels back and forth 250 times. Then a video was taken of the range of motion of the wheels, finally, the test was repeated 8 times in total, for a sum of 2,000 cycles of left and right. This Arduino code can be seen in Appendix 10.5. The code utilizes an Arduino to control the servo position, and an external power supply to provide the servo with 7.2 Volts. Our team helped this MQP team with the development of this testing platform, and after the data was collected, the MQP team utilized a point tracking application to determine the degradation of the steering linkage at each stage of usage, from new to 2,000 cycles.

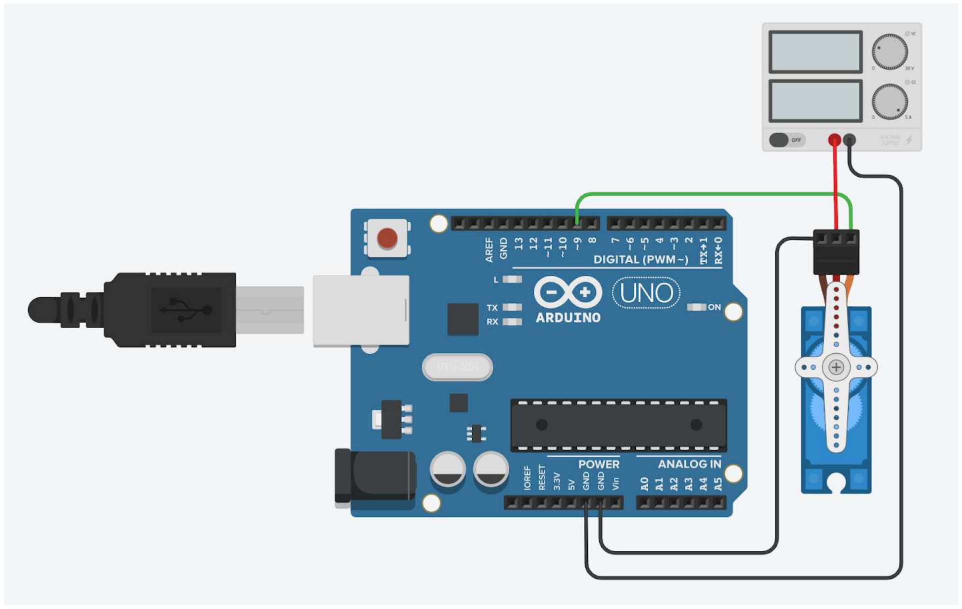


Figure 6.2: Wiring Diagram for Front Suspension Test

As seen above in Figure 6.2, this is the wiring diagram for the servo test rig. It was necessary to use a power supply to provide 7.2 Volts compared to the 5 Volts out from a USB port. The positive from the power supply goes to servo power, and the ground from the power supply goes to the Arduino so that we have a common ground. Then ground from the Arduino is sent to the servo along with a PWM signal from a digital pin.

In this chapter, we discussed the work we did to aid our sister MQP team with the construction of Testing Rigs. In the next chapter, we will talk about the results of our tests on our self-driving and sensor modules. We will also discuss issues with our implementations, and areas where our work could be expanded on by future MQP teams.

7 Results and Discussion

In this chapter, we talk about the results of our tests on our self-driving and sensor modules. We also discuss issues with our implementations, and areas where our work could be expanded on by future MQP teams.

We were successfully able to create our self-driving and sensor packages to our specifications and were able to create user guides and test them on Mechanical Engineering graduate students, who were able to implement the self-driving and sensor packages on a car that they built.

The total cost to buy all of the parts required for the self-driving and sensors kits is $401.13 as of May 2020 (Table 7.1). This cost also includes a full kilogram of Hatchbox PLA. The print settings utilized during our iterations were: 0.10 mm layer height, 20% fill density, and support on build plate only. With these settings, at 100% print speed all of the components would take 19.66 hours to fully print. If the layer height is increased, infill density decreased, or print speed is increased, the time to print all components would be reduced. Additionally, printing all of the components uses 157 Grams of PLA, or 15.7% of the spool. This comes out to $3.13 of PLA.

Table 7.1: Cost of Self-Driving and Sensor Kits

Item Description	Qty	Unit Price	Total Price
RasPi 3B+	2	$51.99	$103.98
Arduino Mega	1	$14.99	$14.99
Micro SD (2-Pack)	1	$24.03	$24.03
Servo Driver	1	$9.99	$9.99
Jumper Wires (Variety Pack)	1	$9.88	$9.88
Hall-Effect Sensor (5-Pack)	1	$6.59	$6.59
Thermal Sensor	3	$8.59	$25.77
IMU	1	$36.00	$36.00
Xbox Controller	1	$74.95	$74.95
Micro USB (2 -Pack)	1	$6.99	$6.99
Hatchbox PLA	1	$19.99	$19.99
Breadboard	1	$8.99	$8.99
Portable Battery	1	$31.99	$31.99
RasPi Carmera Module	1	$26.99	$26.99

Total	$401.13

It takes around 3 hours to set up the sensor kit on a scale car, three to four hours to set up the self-driving kit on a scale car, and then three hours to get the scale car to self-drive. The 3 hours for the sensor kit assumes that the user has already 3D printed the necessary components for the enclosure.

7.1 Self-Driving Module

We ran tests on our self-driving system and were able to get cars made in the Advanced Engineering Design course at WPI to self-drive as well as 3D printed cars made by a project similar to ours to self-drive. We tested our self-driving package on 7 different tracks. The latter tracks were more complex than earlier renditions, featuring added turns. Table 5.2 shows a summary of the various tests that we ran on our self-driving package. As the tests progressed, so did the amount of images required for the training data, however, by test number fifteen, we had devised an optimal method for training the NN, and drastically cut down the number of images required to train the NN, and it had the ability to drive for many more laps than the early iterations.

Table 7.2: Table of Self-driving Package Tests

Test	Track	Ability to Toggle Training Data	Car	Number of Training Data Records	Laps Self-Driven Around Track
1	1	No	Mission D	16k	0
2	2	No	Mission D	5k	2
3	3	No	Student-Made	3k	1
4	4	No	Student-Made	16k	<1
5	4	No	Student-Made	15k	<1
6	4	No	Student-Made	31k	2
7	4	No	3D Printed	18k	<1
8	5	No	3D Printed	35k	<1
9	**5**	**No**	**3D Printed**	**40k**	**>4**
10	**5**	**No**	**3D Printed**	**45k**	**>4**
11	**5**	**No**	**3D Printed**	**35k**	**>4**
12	6	No	3D Printed	60k	<1
13	7	No	3D Printed	50k	2
14	7	No	3D Printed	91k	<1
15	**7**	**Yes**	**3D Printed**	**18k**	**>4**
16	**7**	**Yes**	**3D Printed**	**26k**	**>4**

Our goal was to figure out how to consistently create neural network models that when implemented, self-drove an RC car around a track for at least four laps. Our first successful neural network models had 35-40 thousand records in their training data sets and took 2-3 hours to train. With each new track, we learned techniques to improve the quality of the training data. This improved the robustness of the car's self-driving ability and reduced the number of images required for our training data sets. Our final models only required 20,000 records of training data and only took one and a half hours to train. We were able to significantly reduce the number of training data records by enabling the ability to toggle when the car is collecting training data during the training data collection process. This makes sense, as it gave us more control over what driving behaviors are present in the behavior, causing the resulting neural network to better replicate the behavior that we want it to have.

7.1.1 Testing with Students

After meeting our goal of verifying that our self-driving package could be used to make an RC car self-drive four laps around a track without human intervention, we made a user guide to teach students how to use our self-driving package. We gave this user-guide to two Mechanical Engineering students, Brian King and Jesse Kablik, to test and see if they could follow the user guide and create their own self-driving RC car. It took them about three and a half hours to set-up the self-driving kit on their RC car, and then another three and a half hours to get their car to self-drive. We noticed that they had the hardest time with connecting the Raspberry Pi to the internet due to the security on WPI's Wi-Fi network, but in ME 4320 and ME 4322, this step would be taken care of by WPI's IT department and the class TA's. Aside from that step, Brian and Jesse were able to easily connect all of the necessary hardware for the self-driving kit, download all of the necessary software, collect training data, and create a neural network model. In the end, they got their own RC car to self-drive! We used feedback from Brian and Jesse and notes we took on their experience to improve the user-guide. Now, we are confident that students in the Advanced Engineering Design course can use our self-driving kit to implement their own self-driving systems on the scale cars that they build.

7.1.2 Issues Faced and Future Work

7.1.2.1 Issue: Non-Transferrable Neural Networks

There are some issues with the self-driving kit that also present themselves as areas for future work. First, neural network models are non-transferrable: they only work on the car that they were trained on, and on the track that they were trained on. If these NN models were transferable, that would significantly help the students in the ME 4320 and ME 4322 courses as they would only have to put the self-driving kit onto their car before they could start self-driving their car and would not be required to train their own car.

7.1.2.2 Future Work: Recommendations to Make Neural Networks Transferrable

One way to make the neural network models transferable between cars could be to ensure that all of the scale cars used in the ME 4320 and ME 4322 courses have exactly the same steering behavior and throttle behavior. Currently, the neural network models take in a “servo level” and a “throttle level”. The servo level is a value from negative one to one that is based on how close the steering servo is to being fully rotated in one direction or another. The throttle level is a value from negative one to one that is based on how close the motor is to full power in the forward or negative direction. These servo and throttle levels correspond to different steering angles and speeds on different cars, which is why the neural network models are non-transferable from one car to another. However, if all of the cars in the ME 4320 and ME 4322 courses had the same servo level for a given steering angle and the same throttle level for a given speed, then a neural network model trained on one car could likely be used on the other cars too.

Alternatively, additional sensors could be incorporated into the self-driving kit so the neural network would know the car’s current steering angle and speed to make the neural network models transferable between cars. An IMU or another sensor could be used to detect the current steering angle of the car, and a tachometer could be used to get the speed of a car. Then, the neural network model could learn to react to situations with a certain steering angle and speed instead of the car-dependent metrics “servo level” and “throttle level”. Then, if a working neural network was put on another car, it would likely still work as the neural network would know what speed and steering angle it wants the car to be at instead of using “servo level” and “throttle level”.

7.1.2.3 Issue: Degradation of Steering Performance

Another issue with the self-driving kit is that when we had a scale car self-drive for more than four or five laps, sometimes the scale car would appear to be less responsive to turns. Our sister MQP team, which made the 3D printed RC cars, ran tests on steering servos and saw that a servo’s torque would significantly decrease with continuous use, which could account for this observation.

7.1.2.4: Future Work: Mitigate Degradation of Steering Performance

Since the neural network model is currently set up to set the servo to a specific level with no controls feedback on the actual position of the steering servo (and the wheels of the car), it is possible that giving the neural network more accurate information on the current steering position would improve the self-driving performance. This could then allow scale cars with our self-driving kit to self-drive around tracks for longer before they drive off of the track.

7.1.2.5: Future Work: Improving Self-Driving Performance

One area of future work that would likely improve self-driving performance would be to improve the neural network models that are created by the self-driving kit. One way to do this would be to use PyTorch[29] to design a model more specific to our use case. The Donkey Car platform currently uses the Keras library[30] to create neural network models, but the PyTorch library is more customizable, giving us more control over how the neural network models are made and how they function. A behavioral model could also be implemented to improve neural network performance. Currently, the neural network models try to mimic the behaviors present in the training data, but a behavioral model rewards the neural network model based on whether it's behavior is deemed to be "good" (like driving in the center of the track) or "bad" (like driving off of the track). This could potentially train the cars to stay on the track for longer, improving self-driving performance.

7.1.2.6: Future Work: Improving Training Data

The quality and quantity of training data used to train the neural network models also impacts the self-driving performance of the cars and could be improved upon. A simulation could be created to collect a lot more training data. This would make it easier to collect large amounts of training data and could be used to prepare the cars for a much wider variety of situations, and possibly even make it such that the cars could self-drive on many different tracks. The quality of training data could be improved by writing a program to better edit the set of training data. This could be used to select which frames should be used in the training data set to make sure the neural network models do not learn small mistakes that were left in the training data set. Another use for a training data editing program could be to mark frames as "good" or "bad", which could be useful in making a behavioral neural network model where the cars are explicitly taught what “good” behavior they should do and what “bad” behavior they shouldn’t do.

7.1.2.7: Future Work: Testing with Multiple Cars Self-Driving

Our self-driving kit could also be improved to make it such that multiple cars could self-drive on a track at the same time. We had done some testing with having multiple cars on the same track and we were able to have a self-driving car stop behind a manually driven car or to pass a manually driven car. More extensive testing could be done to determine how well our current kit could be used to have multiple cars driving on the same track. However, one flaw that exists with our kit is that each car would have to be trained individually with multiple cars on the same track. This would take a lot of time as multiple people would have to drive their cars on the track at the same time. This problem could be solved by using a simulation to create training data as it could be easy to simulate multiple cars at the same time to create training data with multiple cars. This could also be solved by making the neural network models more transferable between cars using the methods previously mentioned. But, before either of these solutions are implemented, thought needs to be put in to determine what the ideal behavior is for a self-driving scale car if there is another car in its path. Should the car be trained to drive aggressively, and exhibit behaviors like cutting off other cars and passing other cars, or should the car be trained to drive defensively exhibit behaviors like stopping whenever it gets within three feet of another car? Additionally, if a car is trained to sometimes drive around other cars and sometimes to stop behind other cars, how does one differentiate between the two situations? If there is no distinction between when a self-driving scale car should stop behind another car or drive around another car, then it would be up to chance as to which action the car will make, making the self-driving car’s behavior less predictable, which could make it harder to train the other self-driving cars on the track. These questions and this discussion could be used to teach students in the ME 4320 and ME 4322 courses about what decisions go into making a neural network and a self-driving car, further improving their experience with this type of automated vehicles. One solution to having multiple cars on the same track could be to have a track with several lanes.

Slower cars could be trained to stay in the right-most lanes, and faster cars could be trained to pass these slower cars.

7.1.2.7: Future Work: Obstacles and Level 5 Autonomy

More testing could also be done on the self-driving kit to observe how the self-driving cars react to obstacles and to understand what kind of training data is necessary to train a neural network model that avoids obstacles. This could also be used to implement stop signs and go signs to tell the self-driving cars when to start and stop.

Additional future work could be done to make the self-driving cars created using our self-driving kit to qualify as Level Five Autonomous. This would mean that the self-driving cars could operate on any track with any conditions, which for our case would mean that the car would have to be able to drive around any track without first being trained on that specific track. This would require a lot of training data, which, as previously mentioned, could be created using simulations. More tests would need to be done to improve the self-driving performance on tracks that the cars were not already trained on.

7.1.2.8: Future Work: Improving User Guide

Improving the user guide could be another area of future work. The user guide is able to teach students how to install and how to use our self-driving kit, but additional user-experience testing could be done to make the user guide easier to understand and reduce the amount of time it takes to set up and use the self-driving kit. This could also be used to make the self-driving kit more accessible to a much wider audience beyond engineering students in college.

7.2 Sensor Module

The objective involving the sensor package was to design and implement a data recording package that could store all of the data as perceived by the sensors in a readily available file for users to then be able to access and analyze. With the conclusion of C-Term, we were able to come to a fully functioning sensor package that can sense and record the three different types of data that we originally set out to record: temperature, 3D orientation, and RPM. The package, through serial communication, then stores the data as recorded by the sensor package in the on-board Raspberry Pi. The package, in combination with the developed user guide, can allow for completed assembly in about two hours from start to finish.

7.2.1 Potential for Expansion and Future Work

This project serves as a strong jumping off point for both future project work as well as future classroom application. With the completion of a step-by-step setup guide and multi-car testing, the sensor package can be expanded upon in both depth and breadth. Because the sensor package is designed with expansion in mind, future iterations could include many more sensors, including a current sensor to be used to measure battery usage during operation, strain gauges to measure the strain on the chassis of the car that the sensor package is connected to, and many more. The idea of having the students work alongside mechatronic systems is to offer them the tools and skills necessary such that they can make the improvements they want to see on their mechatronic devices, and the interwoven relationship between the electronic and mechanical components of a device or system.

In combination with the work performed by our sister MQP when working on the controlled-moment gyroscope, one of the next potential project-based improvements could involve the collaboration of the IMU in the sensor package with the CMG to create a vehicle capable of balancing and self-righting itself independent of user interaction. By integrating both of these devices together, the self-driving package portion of the project could also be expanded to create a more versatile, all-terrain self-navigating vehicle.

7.2.2 Potential for Live Webpage

With the preliminary work done towards developing a real-time tracking website, the next step in this department would be to finalize the data storage and begin working on implementing a real time update system to the website. In its current stages, the website is nothing more than a locally hosted webpage that can be used to display data. However, with further work, the end goals of having all the data that is stored in the CSV file be on display in real-time for all students to use during driving can be accomplished.

7.2.3 Issues and Areas for Improvement

In terms of area for improvement, more precise and complicated sensors can be installed and used to replace the existing sensors as needs arise. The current sensors are implemented with the premise of being simple to install and monitor, with only the basic data being returned and stored. By using newer and more precise sensors, the project can take in and provide more data for analysis. The code package could also potentially be compressed down onto significantly smaller printed circuit boards, which would allow the package to decrease in size and increase its modular application to even smaller vehicles and more universal application. More calibration tests could be performed alongside these new sensors to ensure even stronger precision and accuracy from data, should the current package serve insufficiently. The user manual could always see improvements as well, where they could provide more detailed analysis, and even a section on how to the process of adding in new sensors alongside the current instructions. This way, it can offer insight on how to move forward should a team or user decide they want to take what they have learned and expand on it for the purposes of their project.

With our own objectives met, and the framework for further development firmly laid in place, the sensor package portion of the project was able to successfully demonstrate the connection between electrical and mechanical components. In combination with the self-driving package, the combination of the two developed code and hardware packages met nearly all of the target objectives of the term, and even with the complications caused by the COVID-19 pandemic, left a strong foundation and framework for future progress to continue in this field.

7.3 Combined Sensor and Self-Driving Box

In order to better serve the Advanced Engineering Design course, a containment box was designed and built to house both the self-driving and the sensor packages that the students will use. Early iterations of the box did not include the bottom mounting holes nor the battery holder. This was quickly revised to include these critical features. The battery holder on the back side is specifically designed with the Anker Portable Battery listed in the parts list, but can easily be modified in SolidWorks to accommodate most batteries. The mounting holes on the bottom of the box are 0.17 inches in diameter, meaning they can accommodate M4 and smaller bolts, and the holes width wise are 3.5” apart on center, and length wise, the holes are 3” apart on center. These relatively simple numbers will help the students in ME4320 to create a mounting location on their cars for our sensor and self-driving box.

7.4 Summary

In this chapter, we talked about the results of our tests on our self-driving and sensor modules. We also discussed issues with our implementations, and areas where our work could be expanded on by future MQP teams.

In the next chapter, we will summarize our results, and discuss the social, economic, ethical, and environmental aspects of our project. We will also discuss how various courses at WPI have helped us with this project.

8 Conclusion

In this chapter, we will summarize our results, and discuss the social, economic, ethical, and environmental aspects of our project. We will also discuss how various courses at WPI have helped us with this project.

In this project, our goal was to enhance the Mechatronic System Design Education at WPI by creating a modular sensor package, a modular self-driving package, and an installation procedure for the system for the scale cars that students build in the course. We created, designed, implemented, and analyzed this system to ensure a successful implementation into the Advanced Engineering Design course in the 20/21 academic year. Our package is easy to assemble and implement making it ideal for a classroom environment. This package will allow students in the course to make their scale cars autonomous and will familiarize students with sensor capabilities, improving their understanding of mechatronic systems.

8.1 Social, Economic, Environmental, and Ethical Aspects of Our Project

Our project has social, economic, environmental, and ethical aspects. Our project is ethical because we are trying to help students learn more. Additionally, we are crediting all authors whose work we are using in this project. As for the artificial intelligence present in our project, we are creating something that is not destructive. A lot more has to be done with our self-driving kit in order to make it into a danger. The course this kit will be used will not teach or encourage students to use artificial intelligence for destructive purposes. Our self driving kit will not harm students nor pose a risk to their safety while using it: the worst case scenario is that a self-driving scale car runs into a student, but these scale cars have very little mass, so an impact with a self-driving car does not cause bruises.

From an environmental standpoint, our self-driving and sensor kits are reusable, reducing environmental impact as these kits can be reused many times to teach students about mechatronic systems. Also, our project will help students learn about mechatronic systems and how manufacturing can be improved if there are improved sensing capabilities. This would reduce energy waste and loss of productivity as sensors can alert people about possible issues in machines. Increased energy efficiency and productivity efficiency benefit the environment by reducing fossil fuel usage.

Lesser loss of productivity also means lesser job losses since downtime can be reduced, helping society from an economic standpoint. Additionally, these self-driving and sensor kits require parts that need to be bought, slightly benefiting the economy by slightly increasing GDP. Because our kits are reusable, our project helps WPI economically as the kits only have to be bought once before they can be reused in many courses.

From a social standpoint, students will be working in teams to implement these self-driving packages, improving and creating social bonds between the students. Additionally, our project exposes new engineers to self-driving and sensors concepts. Eventually, these engineers will go into the working world where they will benefit from their experience with self-driving cars and mechatronic systems as these systems are a growing field of engineering. In other words, we are preparing them for the eventuality that they will have to work with autonomous and mechatronic systems.

8.2 Reflection on Our Project

This project required us to use knowledge from many different courses that we have taken throughout our time at WPI. We used knowledge of how to create and model parts from ES 1310 (Introduction to Computer Aided Design), to create a camera mount for the self driving system and to create the housing for the sensor and self driving system. We used knowledge learned in ME 4322 (Modeling and Analysis of Mechatronics Systems), ES 2503 (Intro to Dynamic Systems), and ES 2501 (Intro to Static Systems) to do force analysis and analyze the motion of the car to design tests for the IMU as part of the sensor kit performance verification. When wiring the sensors for the sensor kit, we used knowledge from ME 3901 (Engineering Experimentation) and ECE 2010 (Introduction to Electrical and Computer Engineering). When we used python in modifying files used as part of the software of the self-driving kit, knowledge from CS 2301 (Systems Programming for Non-Majors) and CS 1004 (Introduction to Programming for Non-Majors) was useful as we learned Python and programming in general in those courses. Lastly, in ECE 2049 (Embedded Computing in Engineering Design) and RBE

1001 (Introduction to Robotics), we gained experience programming in C and programming embedded systems, which was useful for programming the Arduino used in the sensor kit.

We learned many new skills while working on this project. For the self-driving kit, we had to learn about neural networks and how they worked despite the fact that none of us have ever worked with machine learning and neural network systems. We learned that neural network models mimic the behavior present in the training data that the neural network was trained on, and so we also learned techniques on how to collect training data such that the neural network would self-drive the car around the track instead of self-driving the car off of the track. Some of these techniques, as mentioned in the Self-Driving Implementation chapter, were to only record the car driving back onto the track such that the car would learn to drive towards the center of the track if it ever strayed off course.

We learned new software while working on this project. The development of the self-driving kit required us to learn about how the Donkey Car platform worked so that we could incorporate it as part of our self-driving kit. We also had to learn how to use a computer running the Linux operating system as the Raspberry Pi runs a modified version of linux and we had to use terminal commands to pair the xbox one controller we used to control the car to the Raspberry Pi.

9 References

[1] Center for Automotive Research. (2015). Contribution of the automotive industry to the economies of all fifty states and the United States.

[2] Singh, S. (2015). Critical reasons for crashes investigated in the national motor vehicle crash causation survey (No. DOT HS 812 115).

[3] Heron, M. P. (2017). Deaths: leading causes for 2015.

[4] Coalition for Future Mobility. (n.d.). Highly automated technologies, often called self-driving cars, promise a range of potential benefits.

[5] Radhakrishnan, P. (2020). *Mechatronic Systems* [PowerPoint slides]. Retrieved from

https://onedrive.live.com/view.aspx?resid=166BB6671C27C8D1!8468&ithint=file%2cpptx&authkey=!AFOIBS07RttUoAc

[6] Kim, P. (2017). *Matlab deep learning: with machine learning, neural networks and artificial intelligence*. Apress.

[7] Donkey Car (2020). Train an autopilot. Retrieved May 11, 2020, from https://docs.donkeycar.com/guide/train_autopilot/#collect-data.

[8] Donkey Car (2020). How to Build a Donkey. Retrieved May 11, 2020, from https://docs.donkeycar.com/guide/build_hardware/#choosing-a-car

[9] OSOYOO. (n.d.). OSOYOO Robot car kit Lesson 1: Basic Robot car. Retrieved May 11, 2020, from https://osoyoo.com/2018/12/07/new-arduino-smart-car-lesson1/

[10] Amazon. (n.d.). Developers, start your engines. Retrieved May 11, 2020, from https://aws.amazon.com/deepracer/

[11] Donkey Car. (n.d.). Donkey® Car. Retrieved May 11, 2020, from https://www.donkeycar.com/

[12] Tian, D. (2019). DeepPiCar‑Part 1: How to Build a Deep Learning, Self Driving Robotic Car on a Shoestring Budget. Retrieved May 11, 2020, from https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c

[13] *Ryan Zotti: How to Build Your Own Self Driving Toy Car | PyData Dc 2016*. (2016). Retrieved May 11, 2020, from https://www.youtube.com/watch?v=QbbOxrR0zdA

[14]Cava, R. (2019). I built my own Self Driving (RC) Car! Retrieved May 11, 2020, from https://medium.com/@rodrigocava/i-built-my-own-self-driving-rc-car-1b269fc02e6c

[15]The Github Blog(2019). Retrieved from May 11, 2020, https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/

[16] Elegoo. (n.d.). ELEGOO Upgraded 37 in 1 Sensor Modules Kit V2.0. Retrieved May 11, 2020, from https://www.elegoo.com/product/elegoo-upgraded-37-in-1-sensor-modules-kit-v2-0/

[17] Home Science Tools. (n.d.). ElecFreaks micro:bit Tinker Kit. Retrieved May 11, 2020, from https://www.homesciencetools.com/product/elecfreaks-microbit-tinker-kit/

[18] Hiletgo. (n.d.). HiLetgo. Retrieved May 11, 2020, from http://www.hiletgo.com/ProductDetail/2169787.html

[19] Aggarwal, C. (2018). Neural Networks and Deep Learning A Textbook (1st ed. 2018.). Retrieved May ll, 2020, from https://doi.org/10.1007/978-3-319-94463-0

[20] Pietschmann, C. (2020, April 28). Raspberry Pi 4 Vs NVIDIA Jetson Nano Developer Kit. Retrieved May 11, 2020, from https://build5nines.com/raspberry-pi-4-vs-nvidia-jetson-nano-developer-kit/

[21]Raspberry Pi (2020) Buy a Raspberry Pi 3 Model B – Raspberry Pi. Accessed May 17, 2020. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[22] Adafruit Industries (2020) Adafruit 16-Channel 12-Bit PWM/Servo Driver - I2C Interface. Accessed May 17, 2020. https://www.adafruit.com/product/815

[23] Anker (2020) PowerCore II 6700. Accessed May 17, 2020. https://www.anker.com/products/variant/powercore-ii-6700/A1220011

[24] Raspberry Pi (2020) Buy a Camera Module V2 - Raspberry Pi. Accessed May 17, 2020. https://www.raspberrypi.org/products/camera-module-v2/

[25] Xbox (2020) Xbox Wireless Controller - Black. Accessed May 17, 2020. https://www.xbox.com/en-US/accessories/controllers/xbox-black-wireless-controller

[26] Raspberry Pi (2020) Download Raspbian for Raspberry Pi. Accessed May 17, 2020. https://www.raspberrypi.org/downloads/raspbian/

[27] AnyDesk (2020) The Fast Remote Desktop Application - AnyDesk. Accessed May 17, 2020. https://anydesk.com/en

[28] Python (2020) Python 3.0 Release | Python.org. Accessed May 17, 2020. https://www.python.org/download/releases/3.0/

[29] PyTorch (2020) Pytorch. Accessed May 18, 2020. https://pytorch.org/

[30] Keras (2020 Keras: the Python deep learning API. Accessed May 18, 2020. https://keras.io/

[31]GFlops, G-hours, and CPU hours. (n.d.). Retrieved May 18, 2020, from http://www.gridrepublic.org/joomla/components/com_mambowiki/index.php/GFlops,_G-hours,_and_CPU_hours

[32] Entov, G., Mao, L., Pepicelli, C., Tai, J., White, S., Wolanin, C.: Major Qualifying Projects, https://www.wpi.edu/academics/undergraduate/major-qualifying-project/project-search.

[33]Miller, E. A. (2015, March). *Robocart: System Design for the First Generation Autonomous Golf Cart*. Major Qualifying Project. https://digitalcommons.wpi.edu/mqp-all/1140/.

[34] Kimari, K. (2019, June 24). Nvidia Jetson Nano vs. Raspberry Pi. Retrieved May 18, 2020, from https://www.maketecheasier.com/nvidia-jetson-nano-vs-raspberry-pi

[35]Curbelo, M., Darnley, R., Karet, A., Madhurkar, K., McKillip, D., Schutzman, E., & Scillitoe, C. (2019, June 15). Major Qualifying Project. Retrieved May 17, 2020, from https://digitalcommons.wpi.edu/mqp-all/7103/

[36]LEGO Education. (n.d.). Make a Driverless Car - EV3 Real-World Vehicles - Lesson Plans - LEGO Education. Retrieved May 18, 2020, from https://education.lego.com/en-us/lessons/ev3-real-world-vehicles/make-a-driverless-car

10 Appendices

10.1 Self-Driving User Guide

Self-Driving Package User Guide

Project Team Members

Anthony Marge(ECE & ME)

Joshua Rondon(ME)

Advisor

Prof. Pradeep Radhakrishan(ME/RBE)

Introduction

This is a guide for how to set up and use the self-driving package, which is based on Donkey Car. Donkey Car is an open-source DIY self driving platform for scale cars, and was made by and belongs to Donkey®. We made our own guide by following the steps from Donkey®'s website: http://docs.donkeycar.com/.

Table of Contents

List of Figures

Parts List

Part Specifications	**Possible Purchase Location**	**Cost Estimate**
User Designed RC Car	-	Cost Varies depending on User Design
Anker External Battery Power Bank w/ Micro USB Cable	Amazon	$20
Raspberry Pi 3b+	Amazon	$40
MicroSD Card **Note - Most SD cards will work but the one link is recommended by Donkey®*	Amazon	$20
Raspberry Pi Camera Module	Amazon	$27.00
Raspi Camera Extension Cables **Note - This is optional, however, we suggest getting it because it allows the User to not be limited by the length of the camera cable that the RasPi comes with*	Amazon	$9.99
Female to Female Jumper Cables	Amazon	$7.00
Servo Driver PCA 9685	Amazon	$9.99

Fasteners	This depends on the User design when assembling the components we used duct tape as an adhesive to quickly assemble	-
Xbox Wireless Bluetooth Controller for Training	Amazon	$74.95
Logitech Gamepad F710 **Note - This is just another option for a controller that can be used to train the car on the Self-Driving.*	Amazon	$39.00
USB Keyboard	Amazon Any Keyboard will work	$18.99
HDMI to VGA Connector **Note - The Raspberry Pi has an HDMI port. The VGA connector is for the monitor we used. If the User has a monitor w/ a different port that they should get a cable that is HDMI to the port on their own monitor.*	Amazon	$11.59
Monitor **Note - Any monitor that allows connectivity through a cable will work*	Amazon	$172.86
Mouse	Amazon	$6.99

**Note - Any mouse will do*		
SD Card Reader **Note - This is not need if your computer has an SD card reader*	Amazon	$6.99
USB wall power supply	Amazon	$6.99

1: Hardware Setup

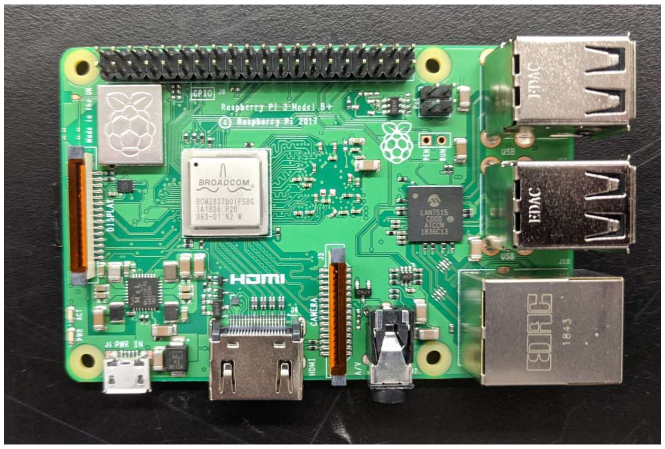


Figure 1. Raspberry Pi

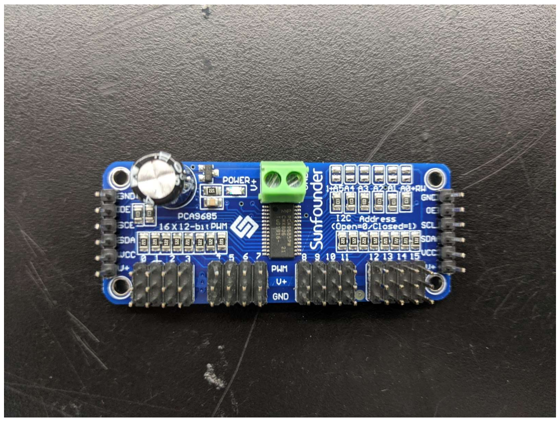


Figure 2. Serve Driver PCA 9685

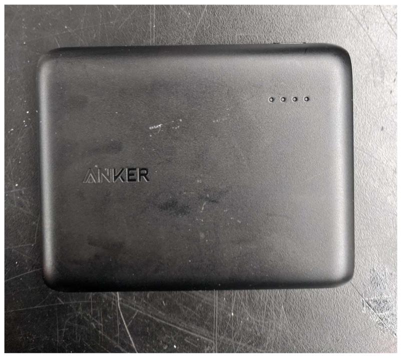


Figure 3. Anker External Battery Power Bank

Figure 4. Micro USB Cable

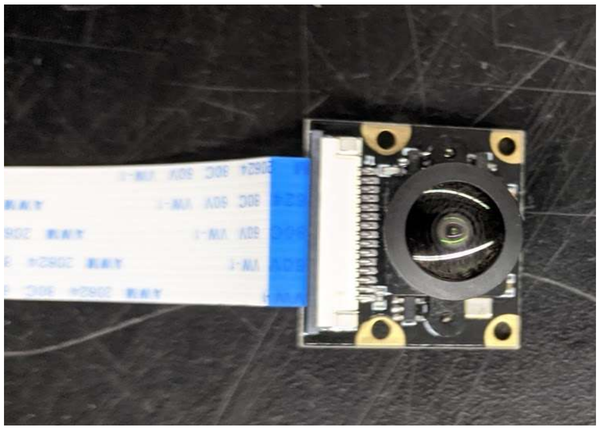


Figure 5. Raspberry Pi Camera Module

Figure 6. Raspi Camera Extension Cables

Instructions:

1. Build your car
    a. It should have space for a camera mount between the front wheels angled downwards from the front plane at a 30°. The camera mount criteria varias per User Design; users should consider that the camera has enough height from the ground so that it can clearly see the desired track. It shou And a camera mount is shown below.

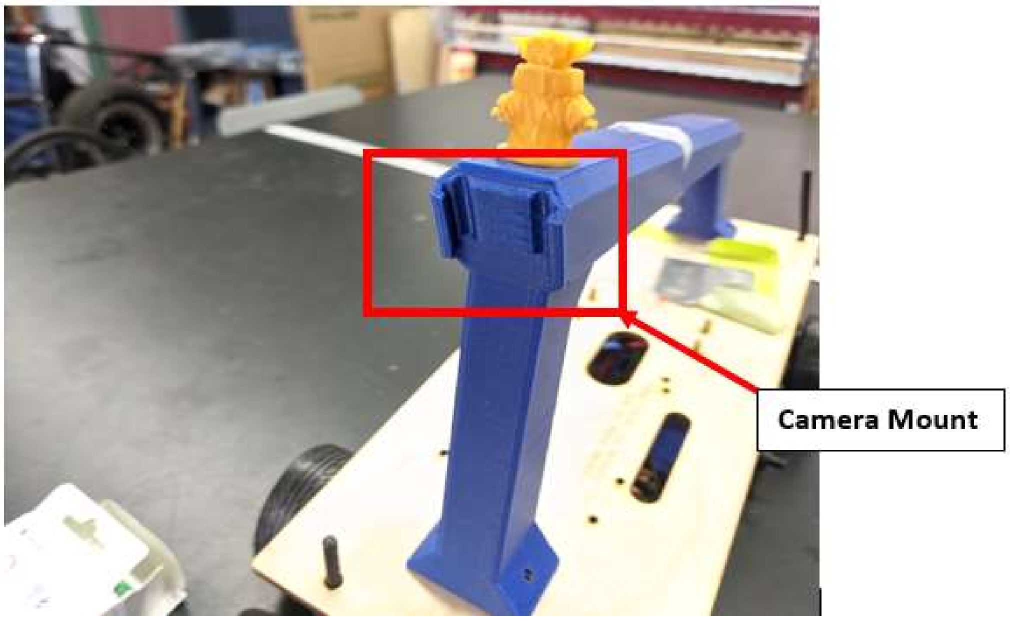


Figure 7. Example Camera Mount that can be purchased at ***[DonkeyCar.com]()***

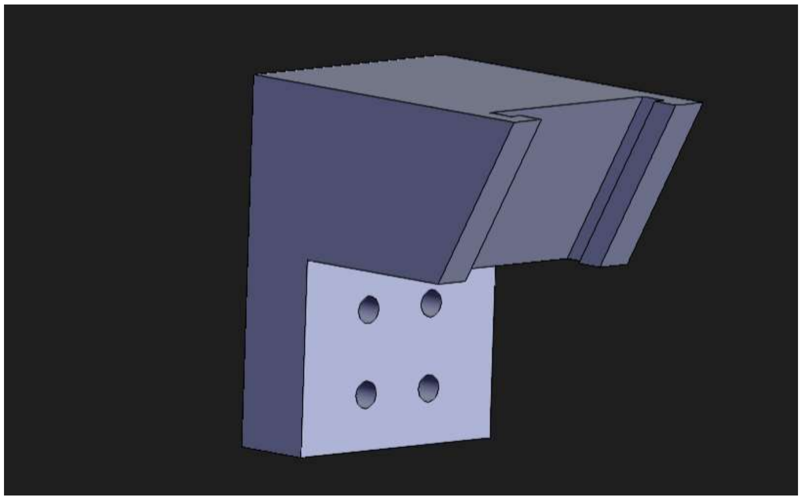


Figure 8. CAD Model of Predesigned Camera Mount. It can be downloaded at this [GrabCAD](). Feel free to make alterations to match your needs.

b. It should also have a place to attach the ***Raspberry Pi, Servo Driver, and Power bank.***

2. Attach raspberry pi to car

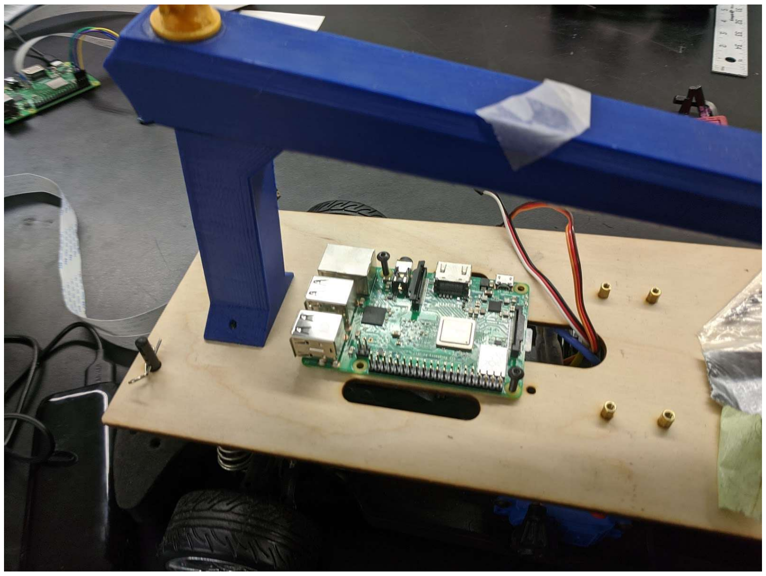


Figure 9. Example of Raspberry Pi on Car

3. Attaching Raspi to Servo-Shield
    a. Don't be overwhelmed by ***Figure 10.*** just use it as a reference for attaching the Servo shield to the Raspberry Pi. The most important aspect of Figure 10 is that it shows you the orientation of the Raspberry Pi with the GPIO(General Purpose Input/ Output) Diagram next to it. The GPIO diagram shows all the Raspberry Pi pins numbered. Use ***Figure 11.*** For the next step of attaching the pins to one another.

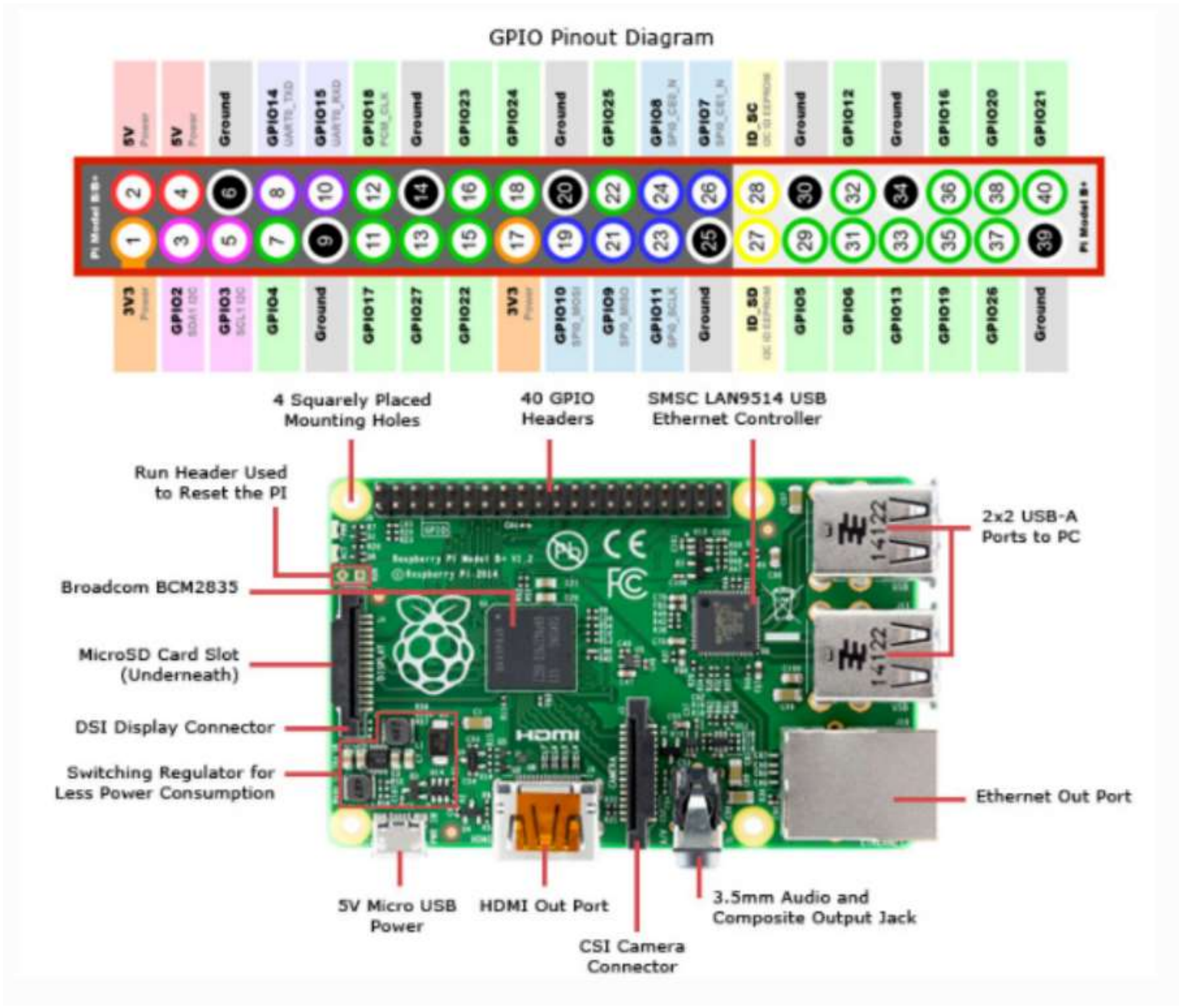


Figure 10. Pinout of Raspberry Pi 3 B+

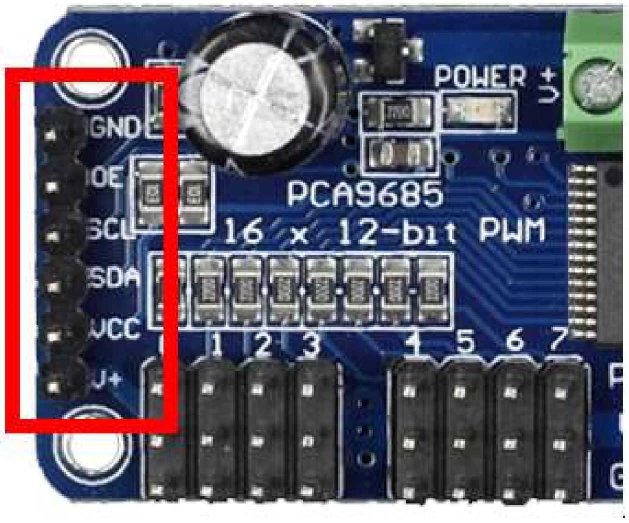


Figure 11. Servo Driver PCA 9685 Pin Highlighted

b. This is connecting the GPIO Pins to the Servo Driver. The VCC, SDA, SDL, and GND pins are highlighted in the ***Figure 11*** diagram. Example end products can be seen in ***Figures 12*** and ***Figure 13***. Use **Female Connectors** for the followings steps:
   i. Pin 1 to Pin Vcc on Servo Driver
   ii. Pin 3 to Pin SDA on Servo Shield
   iii. Pin 5 to Pin SCL on Servo Shield
   iv. Pin 9 to Pin GND on Servo Shield

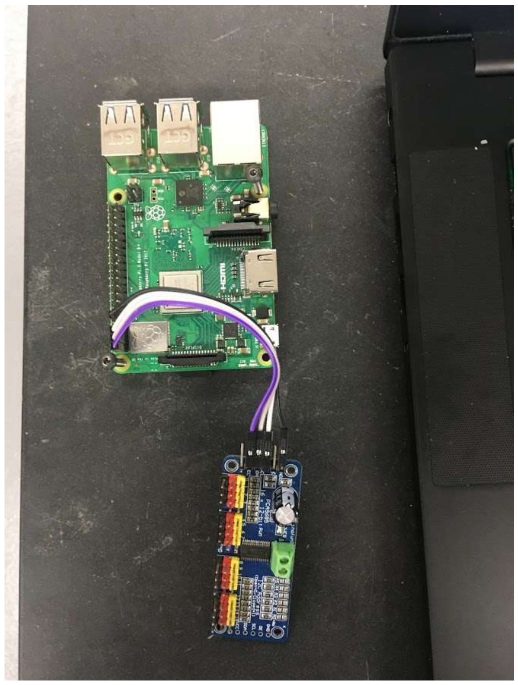


Figure 12. Example of finished Connection between the Raspberry Pi and the Servo Driver

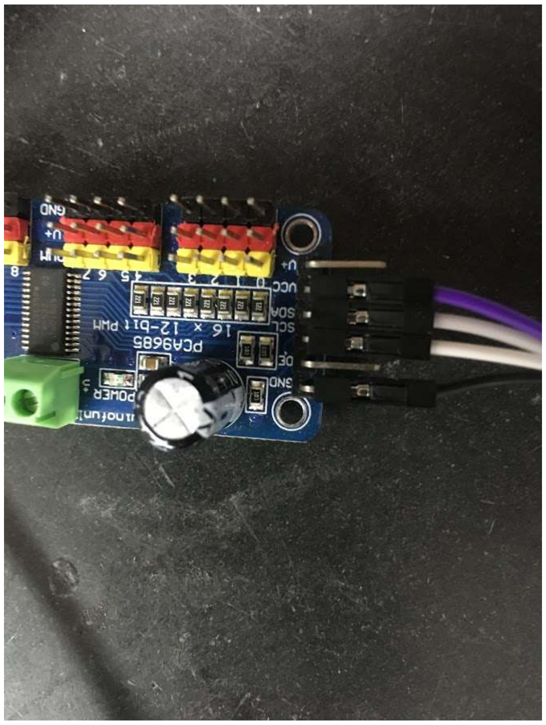


Figure 13. ***2nd Example of the connection between the Raspberry Pi and the Servo Driver***

c. Connecting Servo and ESC to Servo Shield as shown in ***Figure 14*** an example of the Raspberry Pi and Servo Driver on the Car is shown below.

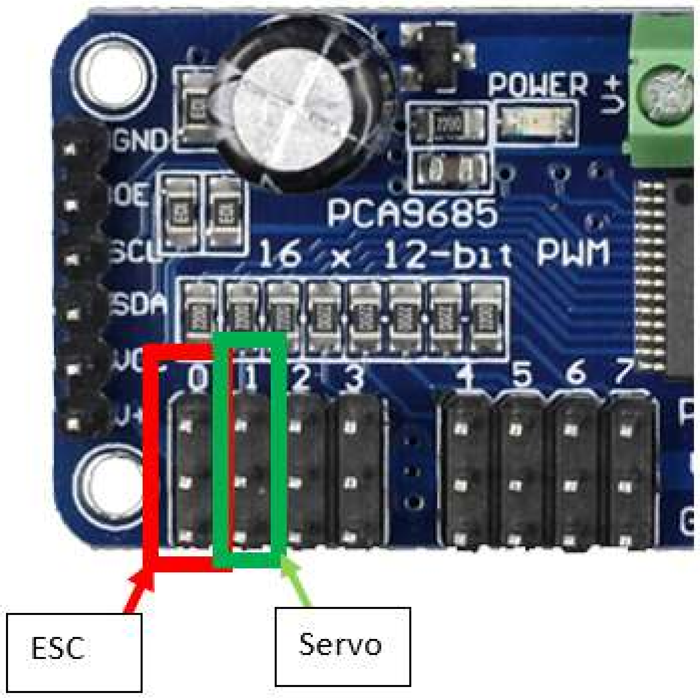


Figure 14. The User Design's ESC will go in Pin 0 and the Servo will go in Pin 2.

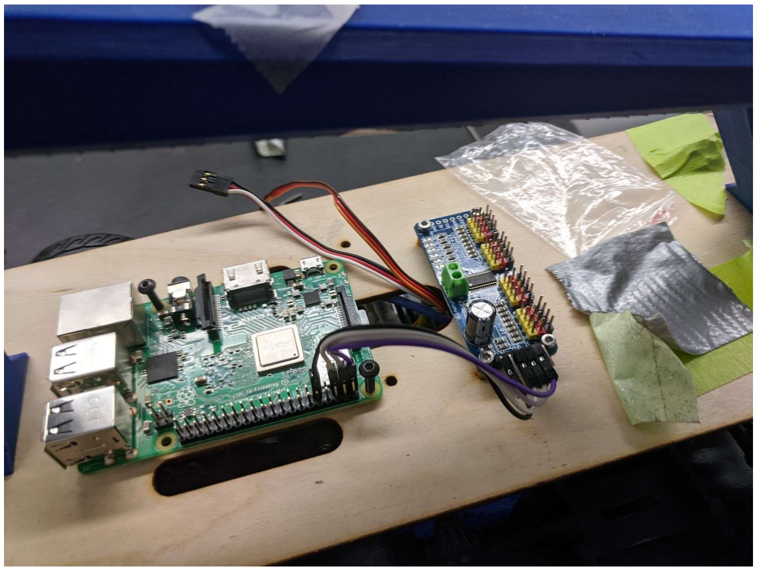


Figure 15. Example of Raspberry Pi and Servo Driver on Car

4. Attach Camera to car

a. First, put the camera in the camera mount as seen in ***Figure 16***. The tolerancing may be tight; users should feel free to make the needed alterations.

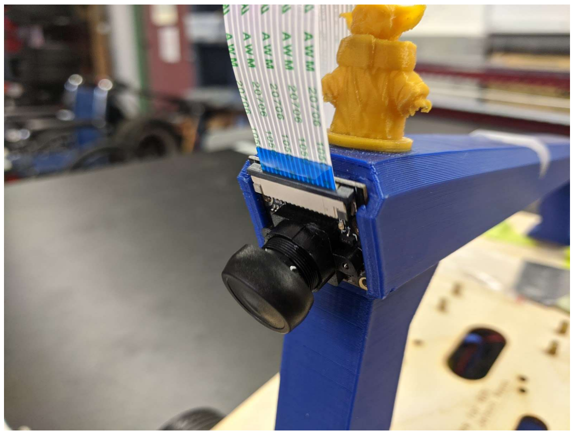
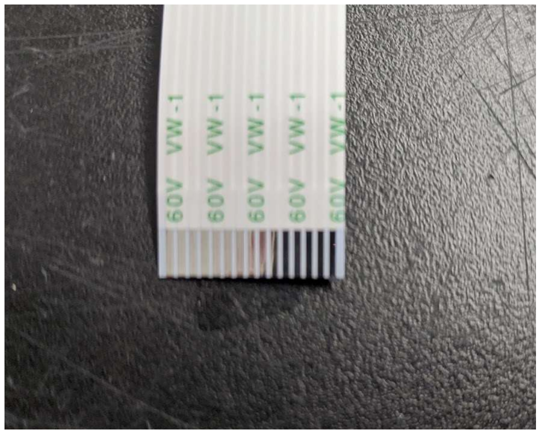
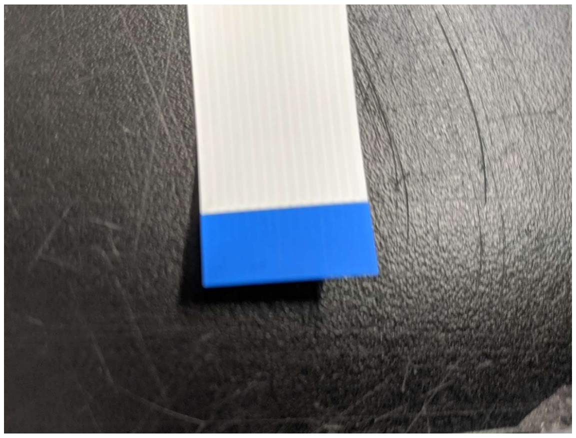


Figure 16. Example camera in camera mount

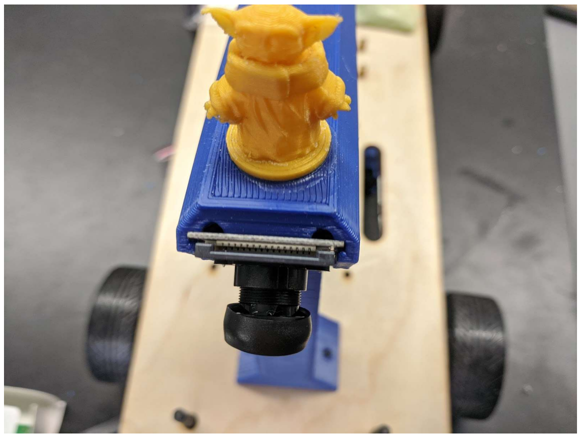
b. Then, connect the camera wire to the camera. The following figures show how to attach the wire to the camera. The leads on the wire should line up with the leads on the connector.



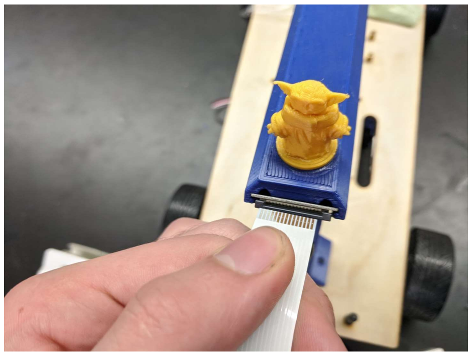
1) *Cable wire lead*



2) *Back of Cable Wire*



3)Open Camera Module without Cable



4) Go to CSI Camera Connector, if you have trouble finding it go to Figure 10 it will show you the location. Same as the prior photos match the cable lead with the connector lead. A good rule of thumb is that the shine side of the cable and connector should match up.

Figure 17. How to connect camera wire to camera. For further assistance you can use this video as a reference.

c. Then, connect the camera wire to the Raspberry Pi. ***Figure 17***, gives information about connection cable leads to Raspberry Pi. A finished attachment is shown in ***Figure 18***.

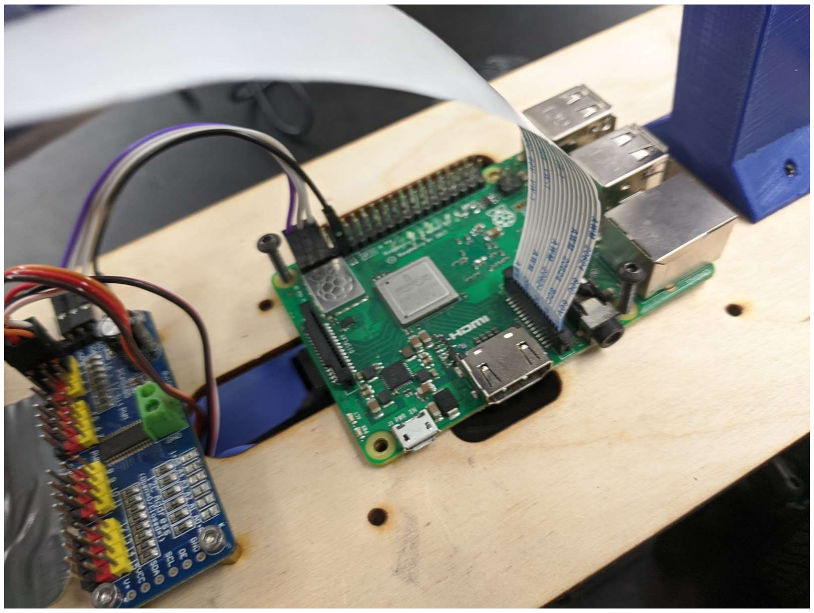


Figure 18: Camera wire connected to Raspberry Pi

5. Then, put the battery on the car.

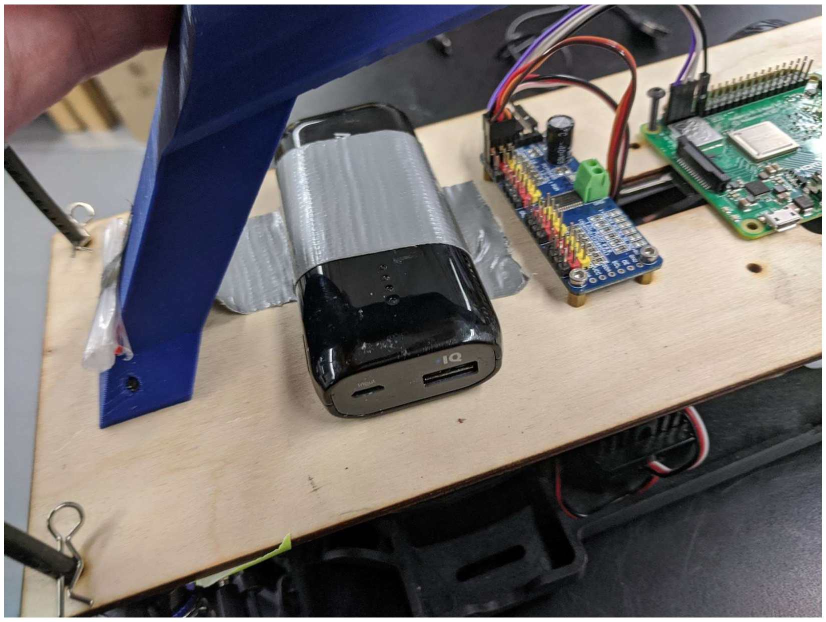


Figure 19: Example of Battery on Car

a. ***Figure 20.*** shows how to connect the raspberry pi to the battery for when you want to turn on the raspberry

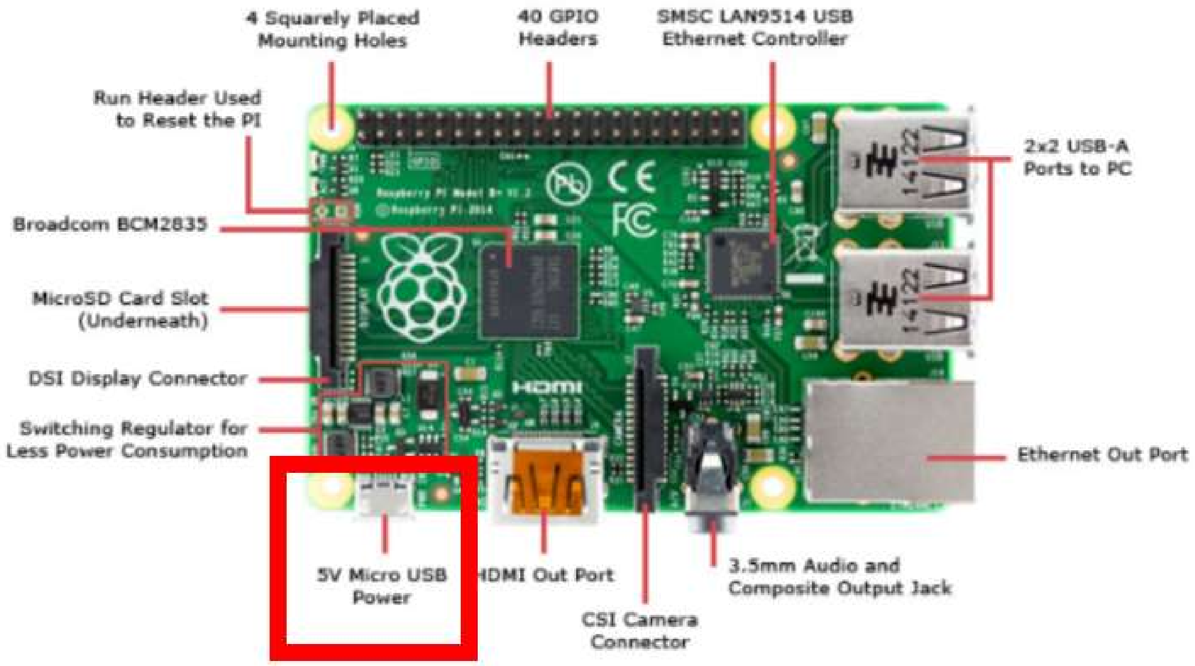

4 Squarely Placed Mounting Holes
40 GPIO Headers
SMSC LAN9514 USB Ethernet Controller
Run Header Used to Reset the PI
2x2 USB-A Ports to PC
Broadcom BCM2835
MicroSD Card Slot (Underneath)
DSI Display Connector
Switching Regulator for Less Power Consumption
Ethernet Out Port
5V Micro USB Power
HDMI Out Port
3.5mm Audio and Composite Output Jack
CSI Camera Connector

Figure 20. The object in the red box is where the User connects their Micro USB cable.

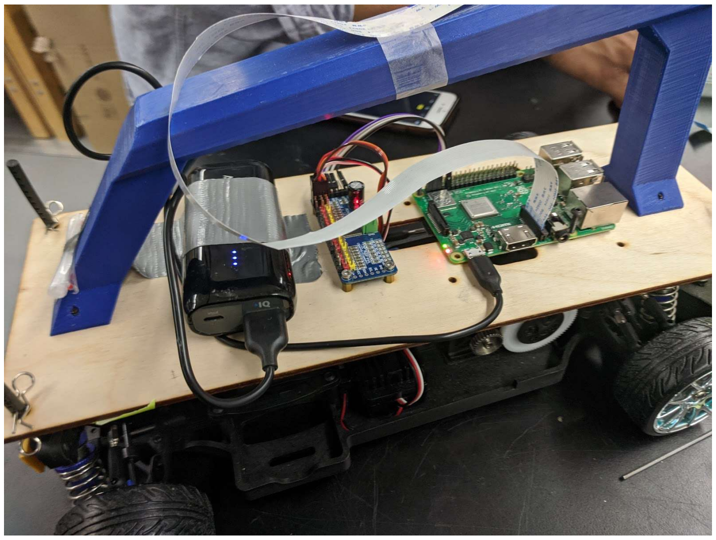


Figure 21. Example of Battery connected to Raspberry Pi

6. The below images are examples of the Self-Driving Package completely set up on a User Designed RC car

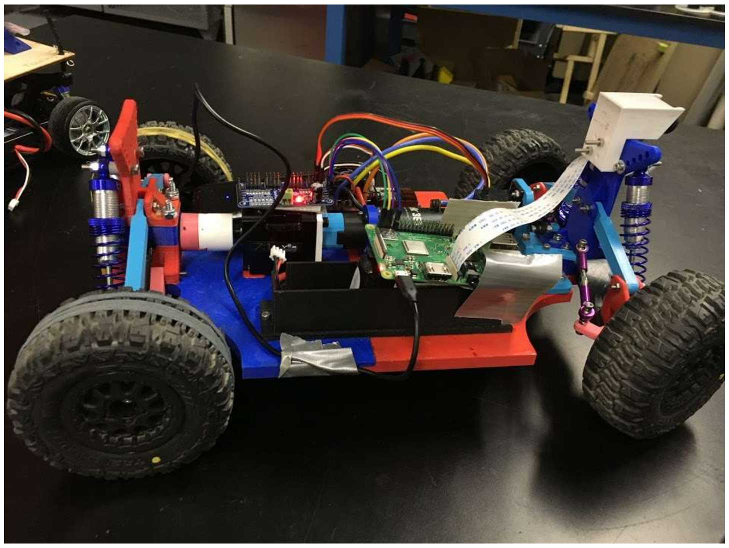


Figure 22. Completed Self-Driving Package

2: Software Setup

The software set up instructions are done on a Windows operating for the User's personal computer and Linux instructions for the Raspberry Pi. The Linux instructions on the Raspberry Pi is because Raspbian, the operating system that is run on the Raspberry, uses a Linux operating system. Windows is for the User's personal computer; because that was the operating we were using when we were developing the User Guide. If the Users wants to explore using a different operating system they can look into it at http://docs.donkeycar.com/.

Some notes for this section is that when Host Machine is mentioned it is referencing your personal computer.

Below are Quick Linux commands for novices that want to have a better understanding of the commands they are entering into both their personal computers and Raspberry Pi.

Quick Linux Commands

ls	Allows User to see files that are in directory
ifconfig	Allows User to see ip-address
cd(FileName or FileName\FileName2)	Allows you to change directories and cd typed by itself brings the User back to the root directory
cd..	Allows the User to go up one directory
mkdir(FileName)	Allows the User to make a file in the directory that the User is currently in.
sudo poweroff	This command shuts off the pi

sudo reboot	This commands reboots the pi
Nano (.txt, .py, etc.)	Allows the User to open a file in the terminal and see its contents

Raspberry Pi SetUp

1. First, we want to set up the micro SD card that will be put in the Raspberry Pi. To do this, first insert the microSD card into the SD card reader on your computer. If your computer only takes full size SD cards, then first put the microSD card into the microSD card to SD card adapter and then insert it into the SD card reader. If you have an external SD card reader, connect it to your computer using a USB port. Look to ***Figure 23.***

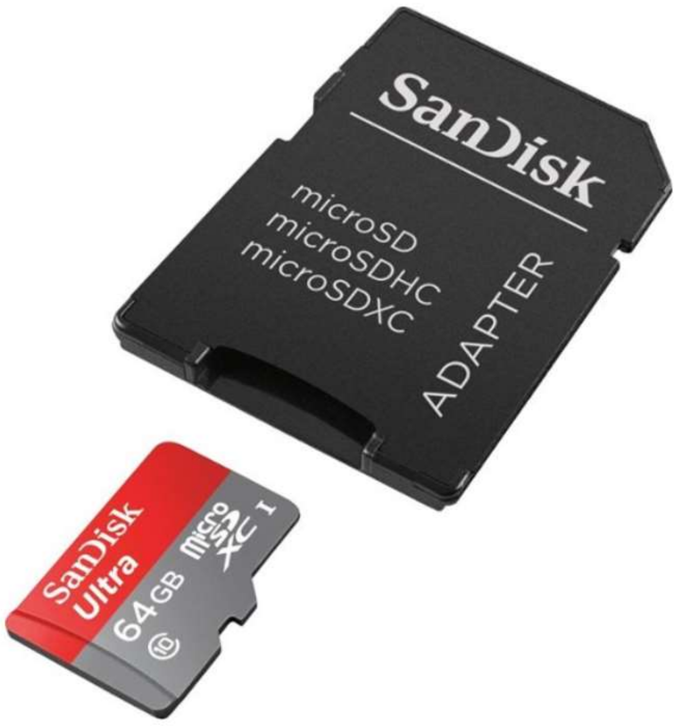


Figure 23. Above image of an microSD card to SD card adapter. This is all that is necessary if your computer has an SD card slot

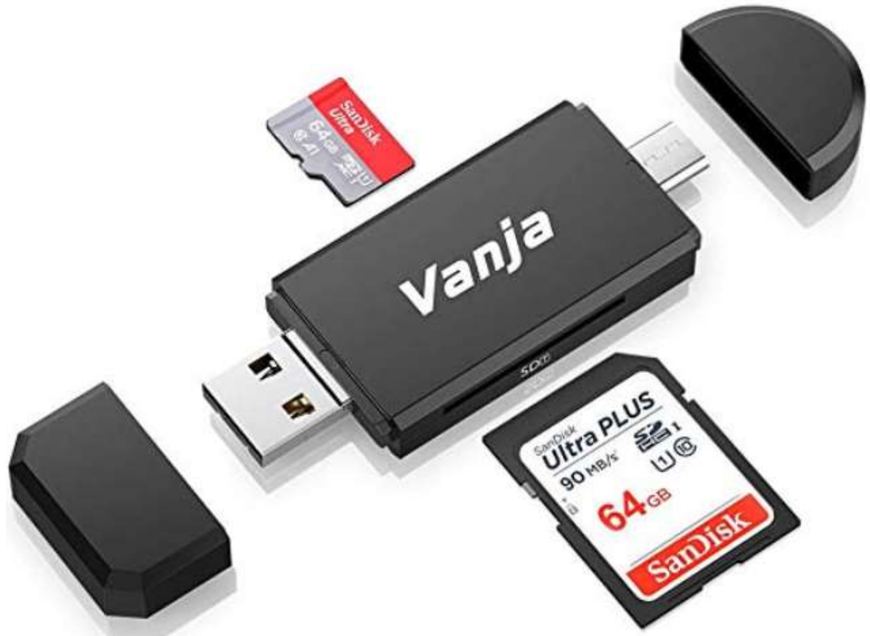


Figure 24. Example of an SD to USB adapter which is needed if your computer does not have an SD card slot. For this specific adapter you can insert the microSD(or the microSD in the

adapter from Figure 23) into the side of this adapter. And then plug the usb directly into the USB port into your computer.

Format SD card

2. Click the link below and continue to follow the instructions in this manual.
   https://www.sdcard.org/downloads/formatter/
3. Scroll down to the very bottom of the website page and click for Windows, as shown in ***Figure 25.***

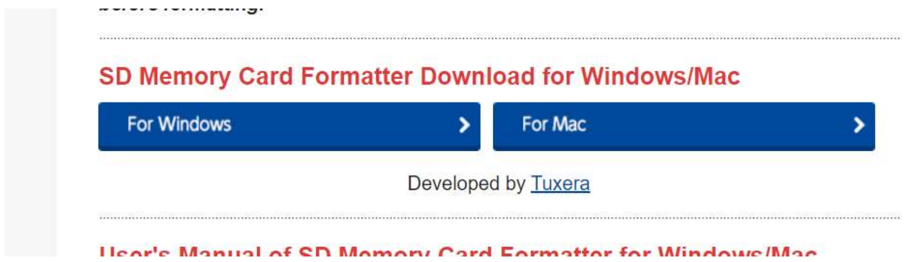
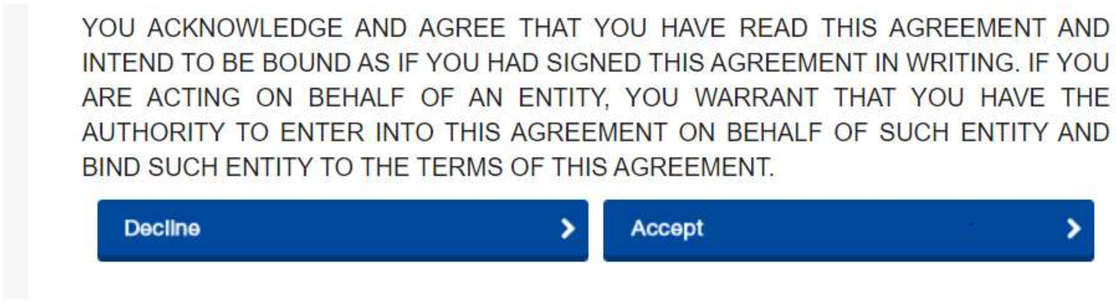


Figure 25. Example of finding Windows selection on SDCard.org

4. Then scroll down to the bottom of the page and select accept, as shown in ***Figure 26***.



YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE READ THIS AGREEMENT AND INTEND TO BE BOUND AS IF YOU HAD SIGNED THIS AGREEMENT IN WRITING. IF YOU ARE ACTING ON BEHALF OF AN ENTITY, YOU WARRANT THAT YOU HAVE THE AUTHORITY TO ENTER INTO THIS AGREEMENT ON BEHALF OF SUCH ENTITY AND BIND SUCH ENTITY TO THE TERMS OF THIS AGREEMENT.

Figure 26. Example of selecting Accept after Window's selection

5. You should have a zip file like the one in ***Figure 27***.

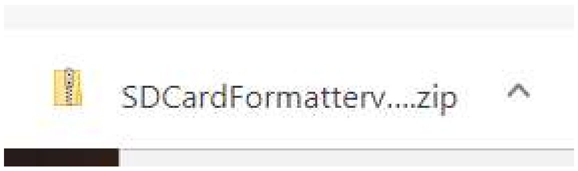
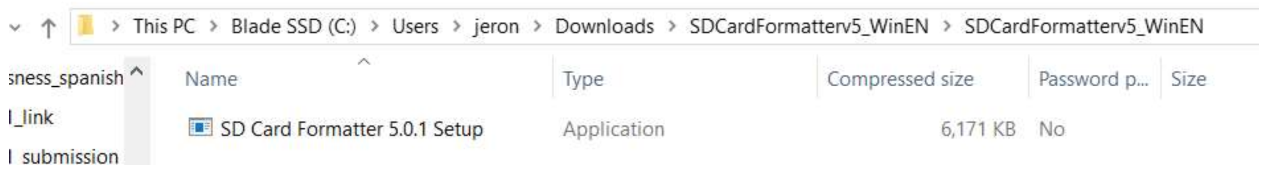


Figure 27. Example of Zip Folder

6. Extract the files that were downloaded from the **SD Memory Card Formatter** to a location on your computer and run the set up.


> This PC > Blade SSD (C:) > Users > jeron > Downloads > SDCardFormatterv5_WinEN > SDCardFormatterv5_WinEN

Name	Type	Compressed size	Password p...	Size
SD Card Formatter 5.0.1 Setup	Application	6,171 KB	No	

Figure 28. Example of Files downloaded onto Personal Computer

7. Now run the SD card formatter app on your computer
8. Select the drop down arrow and choose the drive that you would like for formatting

******Note that many times the correct drive will be chosen automatically. However to be sure that you are selecting the correct drive click File Explorer< This PC, and it'll you'll be able to figure out what drive it is.******

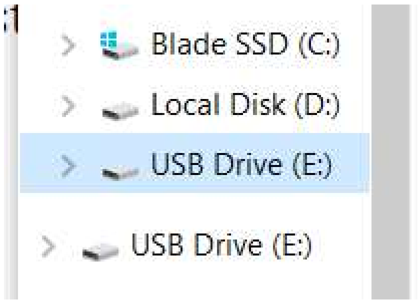
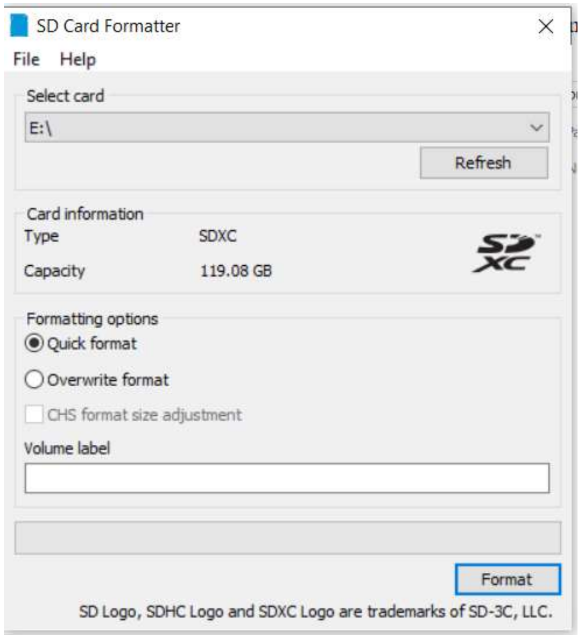


Figure 29. This is an example of the SD adapter that is present on the computer used for creating this manual

9. Knowing the drive name and with an empty SD card inserted into your computer, and Quick format selected you should have an image like **Figure 30**. below.



Select card: E:\

Card information
Type: SDXC
Capacity: 119.08 GB

Formatting options
[✓ Quick format]
Overwrite format
CHS format size adjustment
Volume label

SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

Figure 30. *SD Card Formatter*

10. Now press format; and you have a SD card that is ready for the Raspberry Pi operating system, **Raspbian**.

11. Keep SD card plugged into the computer

12. Installing **Raspbian** operating system images onto SD card
13. Click the the following link, https://www.raspberrypi.org/downloads/raspbian/
14. Download the zip file of the latest version of **Raspbian Buster with desktop and Recommended software** onto your host computer. This file is the Raspbian Operating with a Graphic user interface with some pre-installed software packages that you may find useful later; depending on what else you want to do with the Raspberry Pi.
15. Run balenaEtcher SetUp - click the following link,https://www.balena.io/etcher/.You may get a warning just press continue. This software is what we will use to burn the Raspbian operating system.
16. When you arrive at the site. Click download for Windows, since you should have been on your personal computer for all the steps leading up to this one.

Run the Set-Up for balenaEtcher

17. Run balenaEtcher and put the zip file into the Select Image
18. Select the SD drive that you are going to burn the Raspbian operating system onto.
19. Once the previous 17 and 18 have been completed you should have an image similar to ***Figure 31.***

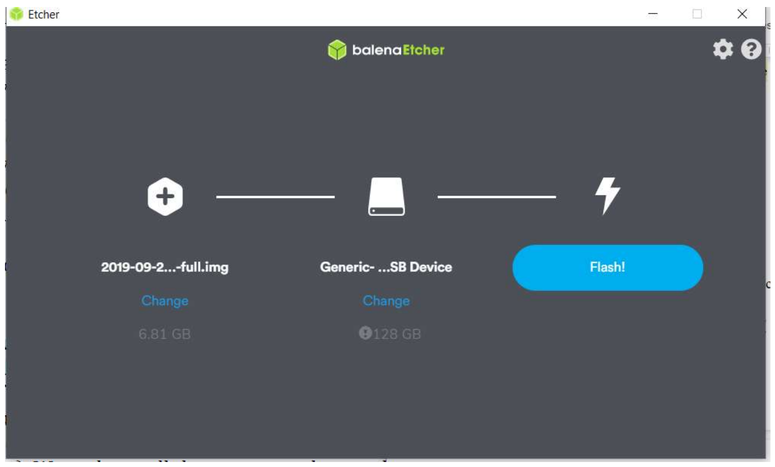


Figure 31. Example of balenaEtcher Operating System

20. Select ***Flash***; you may get a warning. Make sure you select the correct drive letter when formatting. Many times, it will automatically select; but just make sure to check. It should take no more than 5 minutes to burn the operating system on the SD.
21. Once you get a completed message, your SD card is ready for going onto the raspi. If it has not already, go to your PC and eject the drive that you have burned the operating system onto. An example of balenaEtcher burning the Raspbian operating system onto your SD card is shown in ***Figure 32***.

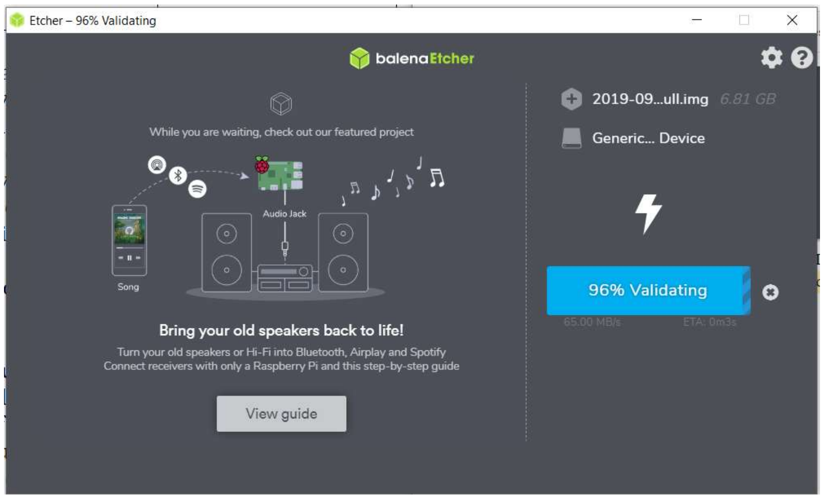

Etcher – 96% Validating
balenaEtcher
While you are waiting, check out our featured project
Audio Jack
Song
Bring your old speakers back to life!
Turn your old speakers or Hi-Fi into Bluetooth, Airplay and Spotify
Connect receivers with only a Raspberry Pi and this step-by-step guide
View guide
2019-09...ull.img 6.81 GB
Generic... Device
96% Validating
65.00 MB/s
ETA: 0m3s

Figure 32. Example of balenaEtcher buring the Raspbian operating system onto SD card

22. Setting Up SD card onto Raspberry Pi, please read the note that is emphasized below
23. Once the SD card has been set up do not worry about ejecting the SD card

******Note: Do not remove the SD card from the Raspberry Pi while it is turned on. Only remove the SD card once the Raspberry Pi has been properly shutdown******

24. For setting up the Raspberry Pi the first time you need a set up as shown in ***Figure 33***.

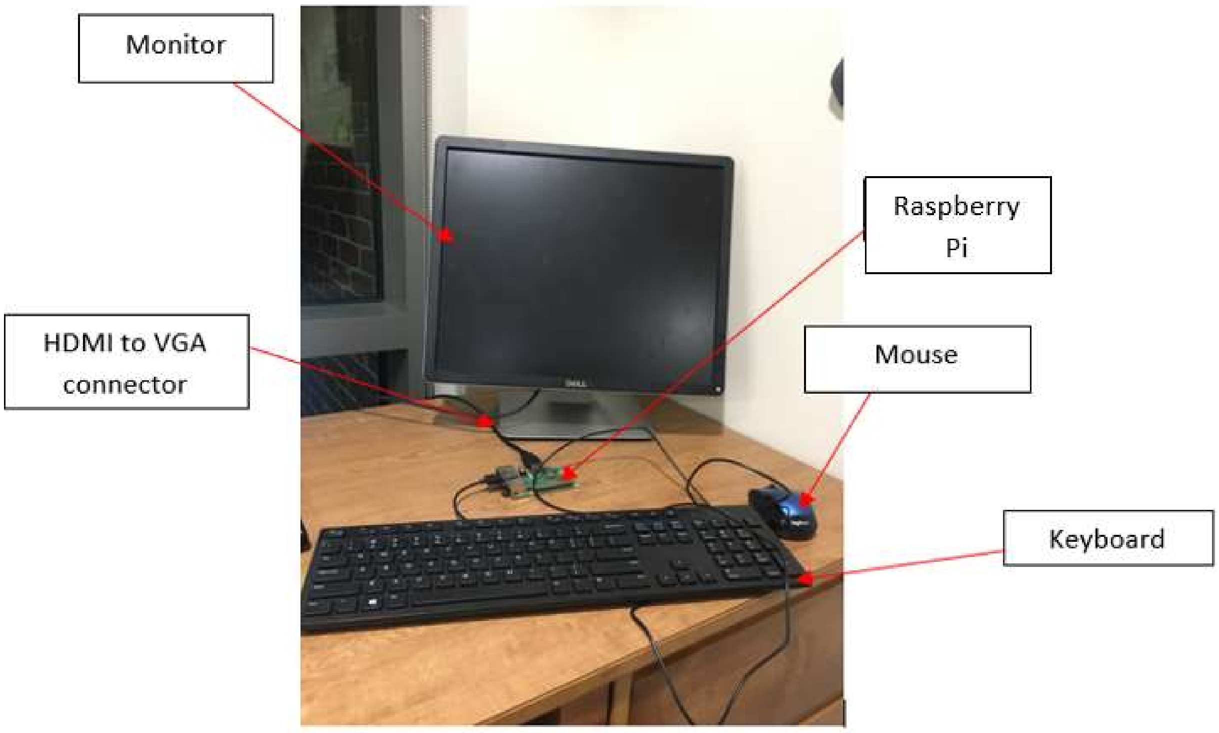


Figure 33. Raspberry Pi initial set-up using HDMI to VGA connector, monitor, mouse, keyboard, and Raspberry Pi.

25. Once you have the set up in ***Figure 33,*** power the Raspberry Pi with the microUSB power supply as shown in ***Figure 34*****.**

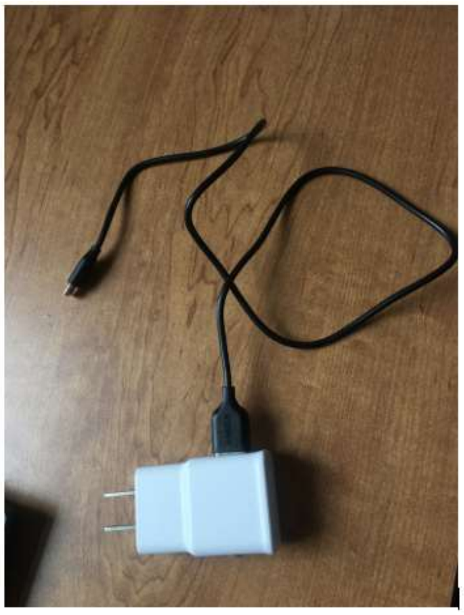
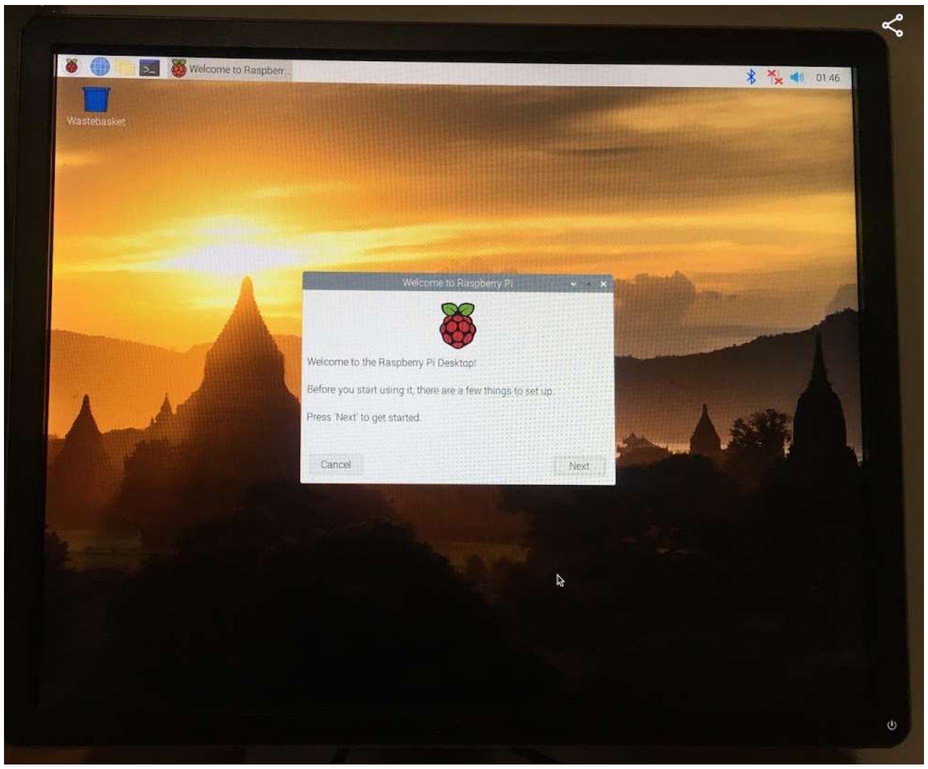


Figure 34. microUSB with wall power supply

26. While the Raspberry Pi is booting you will see berrys show up on the screen and then you will be brought to a welcome screen that looks like the image in ***Figure 34,*** once you see it press next.



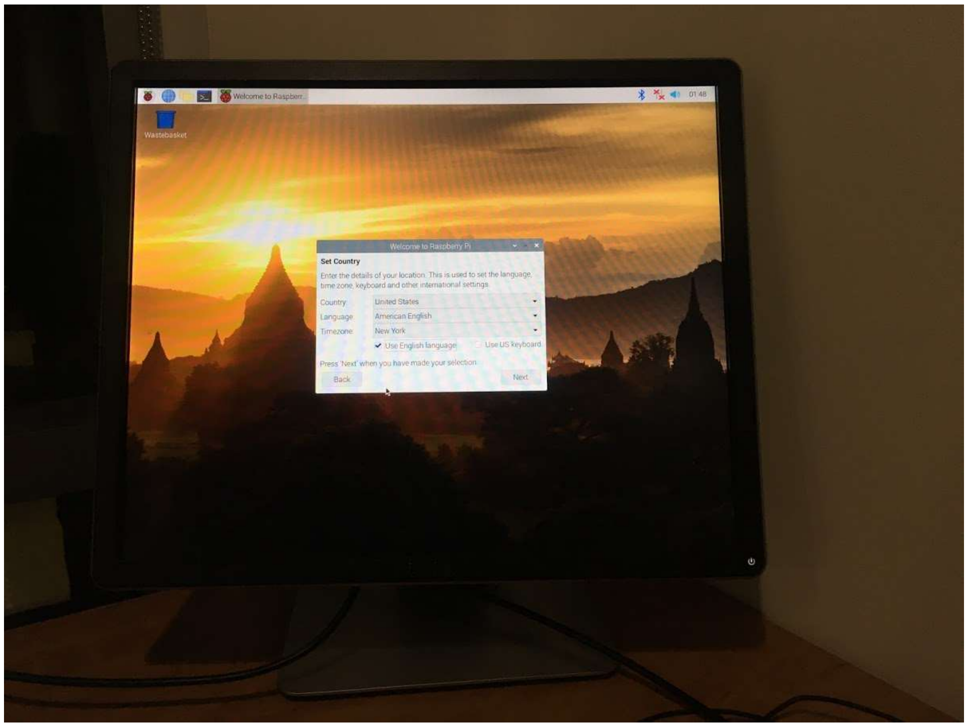
Welcome to the Raspberry Pi Desktop!

Before you start using it, there are a few things to set up.

Press 'Next' to get started.

Figure 34. Raspberry Pi welcome screen

27. Enter in your settings it should match the screen below in ***Figure 35***, once you finish the set up press next



Set Country

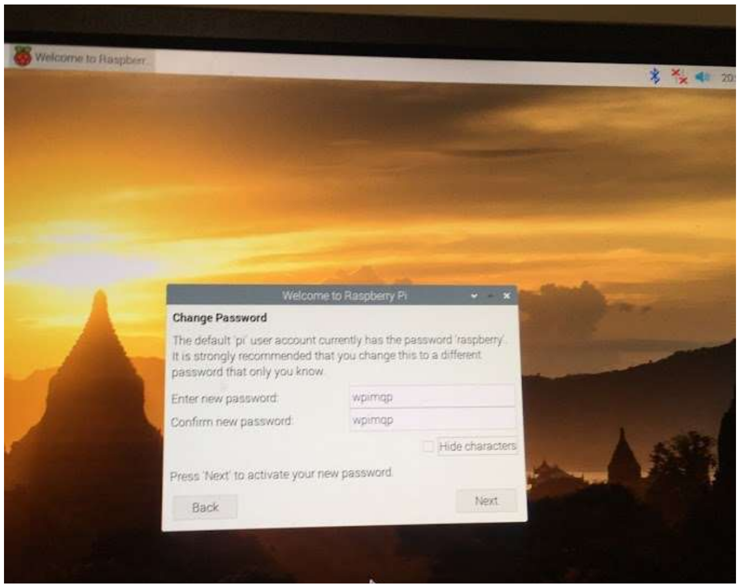
Enter the details of your location. This is used to set the language, time zone, keyboard and other international settings.

Country: United States
Language: American English
Timezone: New York
✓ Use English language ☐ Use US keyboard

Press 'Next' when you have made your selection.

Figure 35. Example of Raspberry Pi settings

28. Now you can change the password to whatever you want but remember to not forget it! Once you have shown click next. An example is shown in ***Figure 36*** below.



Change Password

The default 'pi' user account currently has the password 'raspberry'. It is strongly recommended that you change this to a different password that only you know.

Enter new password: wpimqp

Confirm new password: wpimqp

Hide characters

Press 'Next' to activate your new password.

Figure 36. Raspberry Pi Password Set-Up

29. If there are no black borders click next, if there are black borders check the box in the monitor
30. Select the WPI-Open wifi network
31. Skip Update software
32. At Setup complete restart the Raspberry Pi.
33. Wifi Set-Up. WPI changes the set-up procedures periodically google "WPI its raspi wifi" and follow the instructions that are shown on that webpage.

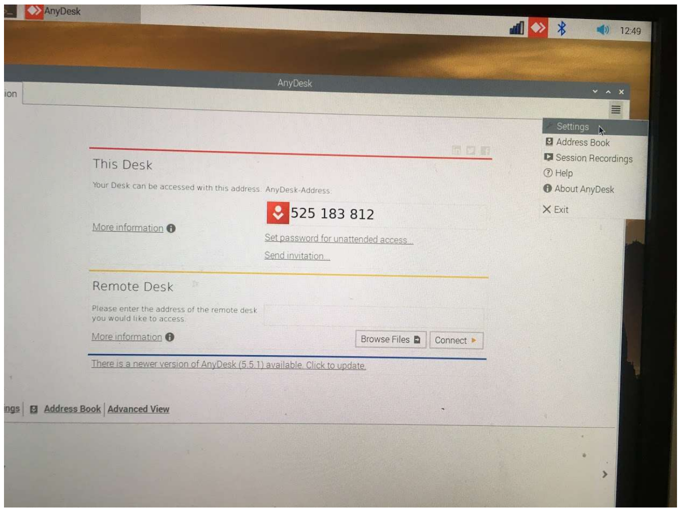
AnyDesk SetUp

34. Downloading AnyDesk to remote desktop to the pi
35. Download AnyDesk to your personal computer in the following link, https://anydesk.com/en?gclid=Cj0KCQiApt_xBRDxARIsAAMUMu-WTENQjUIU76rBV0wJ8H4Gonzfc7vhfzN_xL1XvD3JN8v_Y3aM29QaAsGQEALw_wcB&path=en%2F
36. Download Anydesk to your Raspberry Pi at the following link https://anydesk.com/en/downloads/raspberry-pi
37. Once any desk is downloaded onto the Raspberry Pi make sure to reboot the Raspberry Pi.

38. Click the Anydesk symbol that should be in the upper right hand corner of your monitor for the Raspberry Pi, the AnyDesk symbol is shown in ***Figure 37***.

Figure 37. AnyDesk Logo

39. You should be brought to an image as shown in ***Figure 38***. Click the upper right hand corner of the window that has three lines and select settings.



AnyDesk

12:49

AnyDesk

This Desk

Your Desk can be accessed with this address. AnyDesk-Address:

525 183 812

More information

Set password for unattended access...

Send invitation...

Remote Desk

Please enter the address of the remote desk you would like to access.

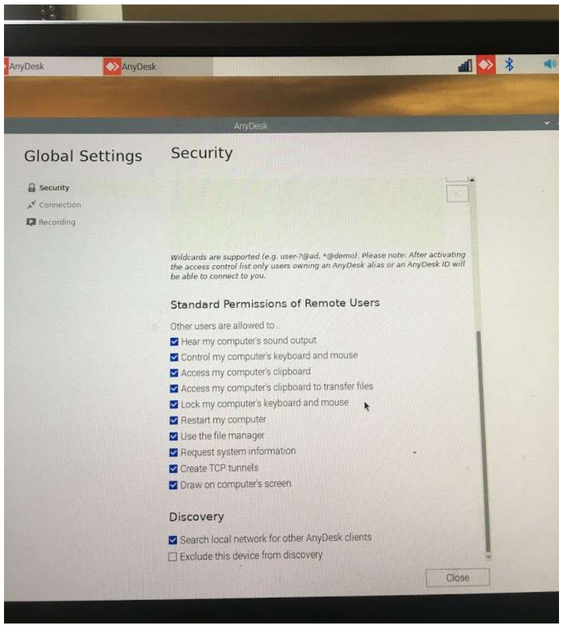
More information

There is a newer version of AnyDesk (5.5.1) available. Click to update.

Figure 38. Anydesk navigating to Settings. Also note the 525 183 812 number in the window. That number will be different in your window. Make sure to note that number because it is part of the process that allows the User to have remote access to the Raspberry Pi.

40. Once in settings select Security

41. You should be in AnyDesk settings in the Security section . There under *Interactive Access* make sure *Allow always* is selected.
42. Next make sure both *Enable unattended access* and *Allow other computers to save login information for this computer* are checked under *Unattended Access*. You may be asked to re-enter your Raspberry Password during this step. When you select **Unattended Access** you will be told to make a password; make sure to remember this password because it will allow computers to remote access your computer.
43. Scroll to the bottom of the page and make sure the *Standard Permissions of Remote Users* is selected as in ***Figure 39***. Once this is complete, AnyDesk on Raspberry Pi should be set up on your computer.



*Wildcards are supported (e.g. user-?@ad, *@demo). Please note: After activating the access control list only users owning an AnyDesk alias or an AnyDesk ID will be able to connect to you.*

Standard Permissions of Remote Users

Other users are allowed to...
- ☑ Hear my computer's sound output
- ☑ Control my computer's keyboard and mouse
- ☑ Access my computer's clipboard
- ☑ Access my computer's clipboard to transfer files
- ☑ Lock my computer's keyboard and mouse
- ☑ Restart my computer
- ☑ Use the file manager
- ☑ Request system information
- ☑ Create TCP tunnels
- ☑ Draw on computer's screen

Discovery
- ☑ Search local network for other AnyDesk clients
- ☐ Exclude this device from discovery

Figure 39. Standard Permissions of Remote User

44. Now that AnyDesk is set up on your Raspberry Pi you can now access it from your personal computer.
45. On your personal computer open AnyDesk wherever you have it stored. Then enter in the number ,that is referenced in ***Figure 38*** to take a note of, into the *Remote Desk* box as

shown in ***Figure 39***. Then press connect. It will ask you for a password in order to access the Raspberry. This will be the one you created in step 42.

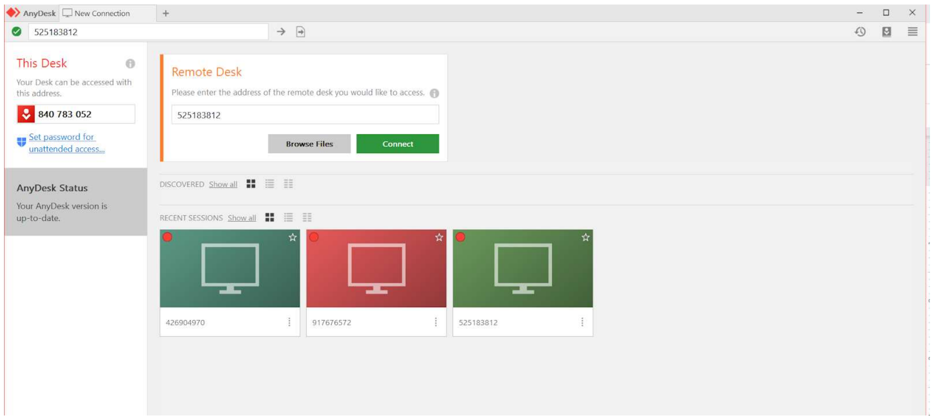

AnyDesk
New Connection
525183812
This Desk
Your Desk can be accessed with this address.
840 783 052
Set password for unattended access...
AnyDesk Status
Your AnyDesk version is up-to-date.
Remote Desk
Please enter the address of the remote desk you would like to access.
525183812
Browse Files
Connect
DISCOVERED Show all
RECENT SESSIONS Show all
426904970
917676572
525183812

Figure 40. AnyDesk on Windows

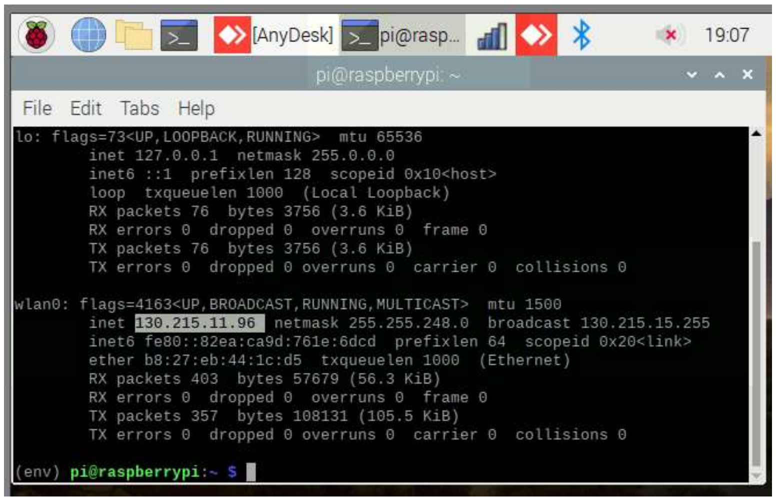
Putty SetUp on Raspberry Pi

The following steps you can do directly plugged into the Pi or on your personal computer through remote access by AnyDesk. Putty is good if you only use the terminal. Anydesk is good if you want to use the graphic User Interface.

1. Select the terminal icon, which is shown in ***Figure 40.***

Figure 41. Terminal Icon

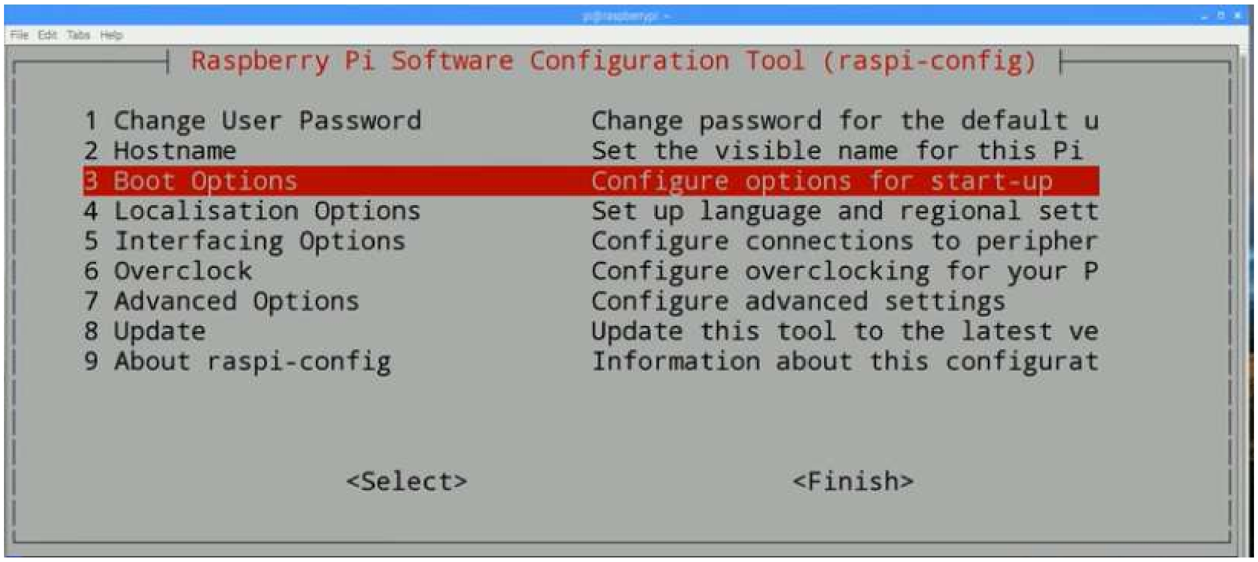
2. Type *ifconfig* into the terminal
3. Check your wireless connection address which can be found in wlan0 next to inet address and seen in ***Figure 41***.



```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 76  bytes 3756 (3.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 76  bytes 3756 (3.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 130.215.11.96  netmask 255.255.248.0  broadcast 130.215.15.255
        inet6 fe80::82ea:ca9d:761e:6dcd  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:44:1c:d5  txqueuelen 1000  (Ethernet)
        RX packets 403  bytes 57679 (56.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 357  bytes 108131 (105.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

(env) pi@raspberrypi:~ $
```

Figure 42. Example of finding wireless ip address

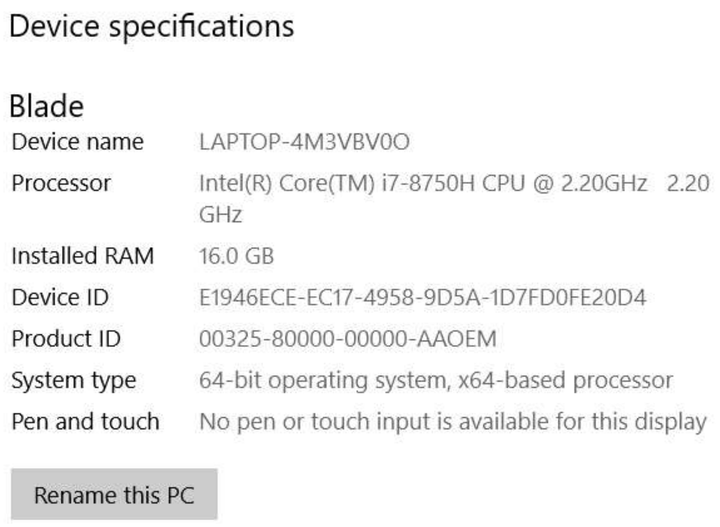
4. Take note of that wireless ip address
5. Now type ***sudo raspi-config*** into the terminal, and you should get a pop up like the one shown in ***Figure 42***.



```
Raspberry Pi Software Configuration Tool (raspi-config)

 1 Change User Password        Change password for the default u
 2 Hostname                    Set the visible name for this Pi
 3 Boot Options                Configure options for start-up
 4 Localisation Options        Set up language and regional sett
 5 Interfacing Options         Configure connections to peripher
 6 Overclock                   Configure overclocking for your P
 7 Advanced Options            Configure advanced settings
 8 Update                      Update this tool to the latest ve
 9 About raspi-config          Information about this configurat

            <Select>                      <Finish>
```

Figure 43: Raspberry Pi Config GUI

6. Select Interfacing options and enable SSH
7. Now once it is enabled hit finish
8. Go to the following website website below, https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
9. And download the latest version of putty that fits your personal computer. You can find this by right clicking the windows icon and then selecting system. An example of this a pop-up you could see is shown in



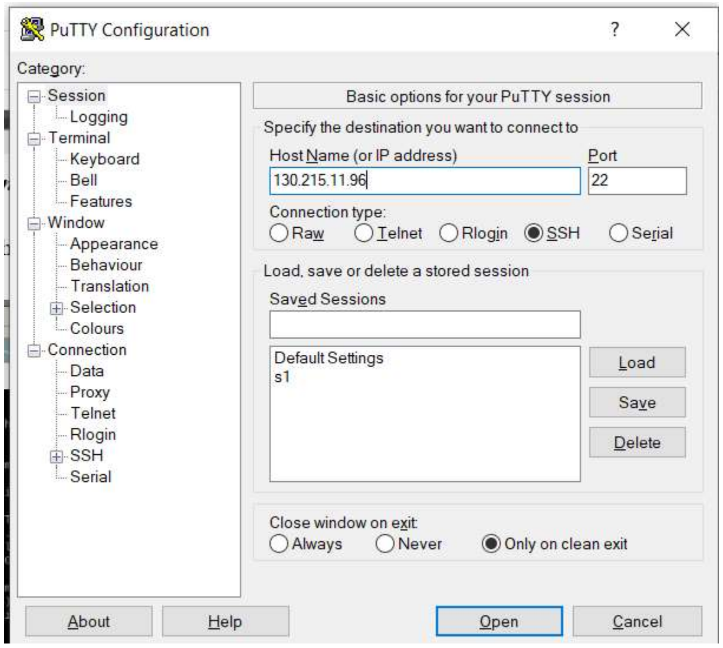
Device specifications

Blade

Device name	LAPTOP-4M3VBV0O
Processor	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz
Installed RAM	16.0 GB
Device ID	E1946ECE-EC17-4958-9D5A-1D7FD0FE20D4
Product ID	00325-80000-00000-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 44: Example of a device's System specifications

10. Once putty is downloaded you just enter in the Raspberry's IP address as shown in ***Figure 44*** , and then press open.



Specify the destination you want to connect to

Host Name (or IP address): 130.215.11.96

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions

Default Settings
s1

Close window on exit: Always Never [✓ Only on clean exit]

Figure 45. Accessing Raspberry Pi terminal through PuTTY

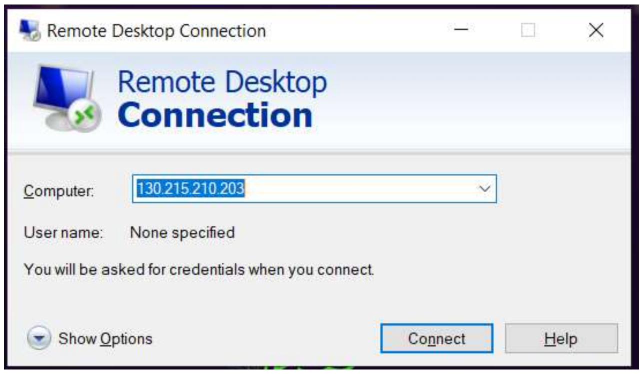
11. When you press Open, and the connection goes through, you will be asked for the Username and Password you created for the Raspberry Pi. When you start typing and you do not see anything showing do not be alarmed. Keep typing, it is just a privacy feature.

Windows remote Desktop

1. Go to the terminal of your Raspberry Pi and type in the following commands:

```
sudo apt-get install tightvncserver

sudo apt-get install xrdp
```

2. Once those commands are typed you can now remote Access to the Raspberry Pi through the Windows remote desktop. This is shown in ***Figure 46***.



Computer: 130.215.210.203

User name: None specified

You will be asked for credentials when you connect.

Figure 46. Windows Remote Desktop

Machine Preparation

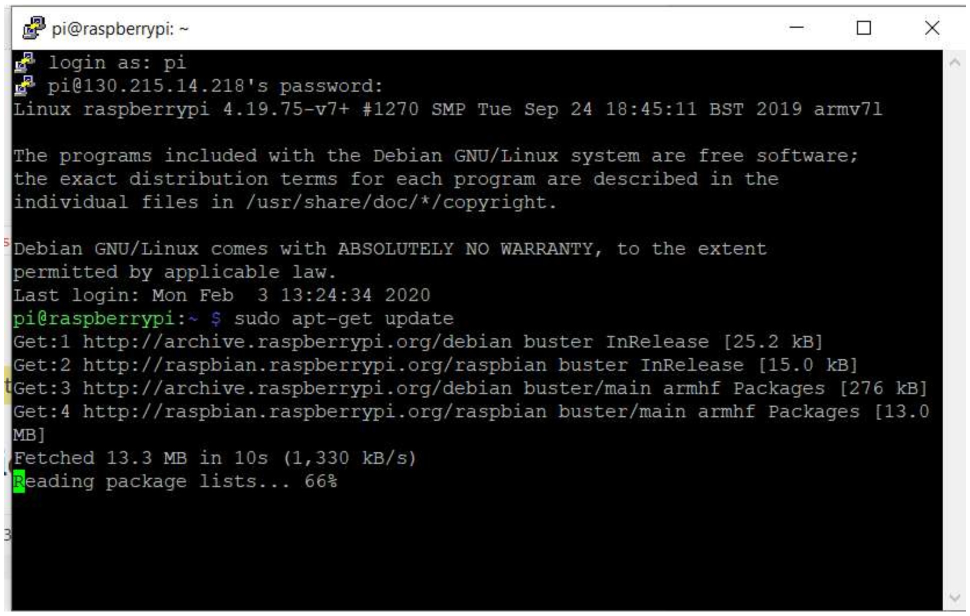
The purpose for setting up all that remote desktop software in the previous sections is so that you have options as to how you interface with the Raspberry Pi. In our experience with dealing with issues connecting to the WPI wifi we found that Anydesk worked best in initially connecting to the Pi to get the ip address, and for transferring files from our personal computer to the Pi and vice versa. Putty is good if you are only interested in inputting terminal commands. Windows Remote Desktop allows you to have a bigger GUI than is offered by Anydesk.

The following instructions you can follow using any of the remote desktop software that we have set up in the manual.

1. Update and upgrade the Pi by entering the following commands into the terminal. If prompted, reboot the RasPi. An example of entering these commands is shown in ***Figure 47.***

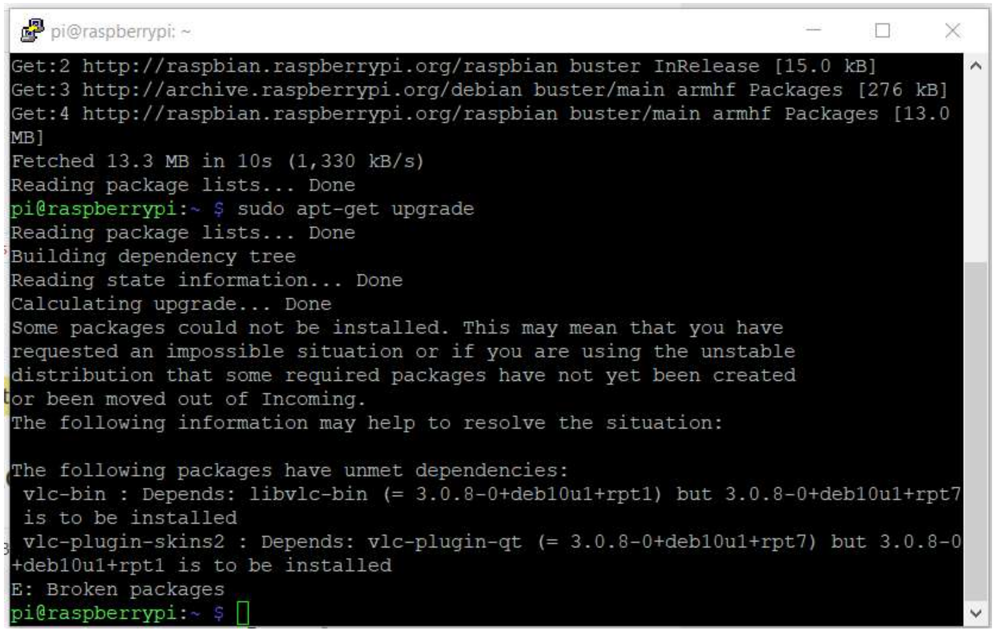
sudo apt-get update

sudo apt-get upgrade



pi@raspberrypi: ~

```
login as: pi
pi@130.215.14.218's password:
Linux raspberrypi 4.19.75-v7+ #1270 SMP Tue Sep 24 18:45:11 BST 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb  3 13:24:34 2020
pi@raspberrypi:~ $ sudo apt-get update
Get:1 http://archive.raspberrypi.org/debian buster InRelease [25.2 kB]
Get:2 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]
Get:3 http://archive.raspberrypi.org/debian buster/main armhf Packages [276 kB]
Get:4 http://raspbian.raspberrypi.org/raspbian buster/main armhf Packages [13.0
MB]
Fetched 13.3 MB in 10s (1,330 kB/s)
Reading package lists... 66%
```

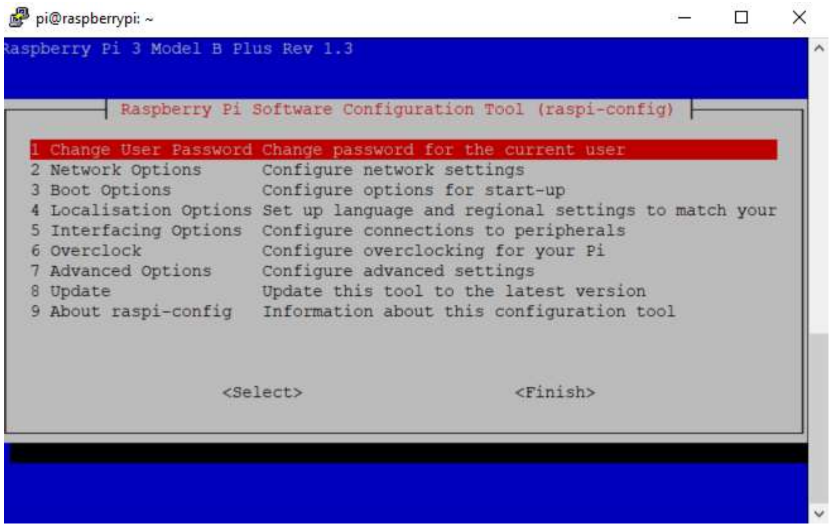


pi@raspberrypi: ~

```
Get:2 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]
Get:3 http://archive.raspberrypi.org/debian buster/main armhf Packages [276 kB]
Get:4 http://raspbian.raspberrypi.org/raspbian buster/main armhf Packages [13.0
MB]
Fetched 13.3 MB in 10s (1,330 kB/s)
Reading package lists... Done
pi@raspberrypi:~ $ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
 vlc-bin : Depends: libvlc-bin (= 3.0.8-0+deb10u1+rpt1) but 3.0.8-0+deb10u1+rpt7
 is to be installed
 vlc-plugin-skins2 : Depends: vlc-plugin-qt (= 3.0.8-0+deb10u1+rpt7) but 3.0.8-0
+deb10u1+rpt1 is to be installed
E: Broken packages
pi@raspberrypi:~ $ 
```

Figure 47. Sudo apt-get upgrade and update

2. Type **sudo raspi-config** in the terminal and you will be brought to the screen in ***Figure 48.***



```
pi@raspberrypi: ~
Raspberry Pi 3 Model B Plus Rev 1.3

         Raspberry Pi Software Configuration Tool (raspi-config)

 1 Change User Password Change password for the current user
 2 Network Options      Configure network settings
 3 Boot Options         Configure options for start-up
 4 Localisation Options Set up language and regional settings to match your
 5 Interfacing Options  Configure connections to peripherals
 6 Overclock            Configure overclocking for your Pi
 7 Advanced Options     Configure advanced settings
 8 Update               Update this tool to the latest version
 9 About raspi-config   Information about this configuration tool

              <Select>                    <Finish>
```

Figure 48. Sudo raspi config screen

3. Use the arrows to navigate

Use the up and down arrow to navigate to 5, Interfacing Options.

Once you select Interfacing Options use the left arrow and right arrow and select Finish to enter into Interfacing options.

Using the navigation above, enable:

   a. Interfacing Options - I2C
   b. Interfacing Options - Camera
   c. Advanced Options - Expand Filesystem (use the whole sd-card storage). Once you have these parts enabled, reboot

4. Install dependencies

Copy and paste the text below into you terminal; once you have copied the text just right click in the terminal to paste the text then press enter

```
sudo apt-get install build-essential python3 python3-dev
python3-pip python3-virtualenv python3-numpy
python3-picamera python3-pandas python3-rpi.gpio i2c-tools
avahi-utils joystick libopenjp2-7-dev libtiff5-dev gfortran
libatlas-base-dev libopenblas-dev libhdf5-serial-dev git
ntp
```

5. Install OpenCV dependencies

```
sudo apt-get install libilmbase-dev libopenexr-dev
libgstreamer1.0-dev libjasper-dev libwebp-dev libatlas-base-dev
```

```
libavcodec-dev libavformat-dev libswscale-dev libqtgui4
libqt4-test
```

6. SetUp Virtual Env

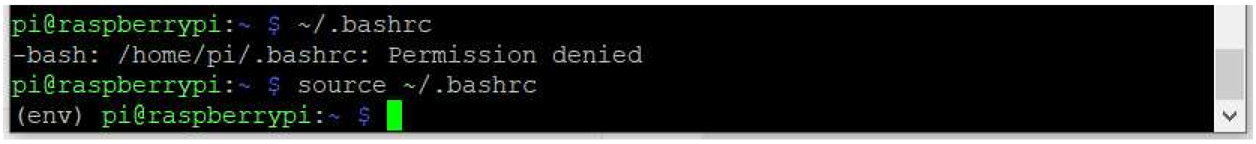
Copy and paste into terminal

```
python3 -m virtualenv -p python3 env --system-site-packages
echo "source env/bin/activate" >> ~/.bashrc
source ~/.bashrc
```

7. Install Donkeycar Python Code

```
mkdir projects
cd projects
git clone https://github.com/autorope/donkeycar
cd donkeycar
git checkout master
pip install -e .[pi]
pip install tensorflow==1.13.1
```

You should be in the coding environment right now, but to activate coding environment type ***source ~/.bashrc*** when you are in the root directory. To make sure you are there type cd and when you want to exit that environment type ***deactivate*** all of this take place in the terminal as shown in ***Figure 49.***



```
pi@raspberrypi:~ $ ~/.bashrc
-bash: /home/pi/.bashrc: Permission denied
pi@raspberrypi:~ $ source ~/.bashrc
(env) pi@raspberrypi:~ $
```

Figure 49. Activating and deactivating virtual environment

8. In the environment type the following command and press enter after every line

   mkdir projects

   cd projects

9. Download the latest donkey car code from Github; after every line press enter

After entering the code below you can check tensor flow by typing python -c “import tensorflow”

```
git clone https://github.com/autorope/donkeycar
```

```
cd donkeycar
git checkout master
pip install -e .[pi]
pip install tensorflow==1.13.1
```

10. Install OpeCV

```
sudo apt install python3-opencv
```

If that fails try:

```
pip install opencv-python
```

11. Create Donkeycar application

Copy code below and paste it into terminal:

```
donkey createcar --path ~/mycar
```

12. Configure Options

Copy and paste the code below into the terminal:

```
cd ~/mycar
nano myconfig.py
```

The myconfig.py file is the file used to be able to edit the default values such as steering PWM values. This will come in handy in the calibration section of this module.

Setting Up Joystick Xbox One

Establishing a connection can take anywhere from 2 - 30 minutes; do not be discouraged just keep troubleshooting if your connection is not being established.

For the Logitech Gamepad F710 just plug in the dongle into the Raspberry Pi, and you should be able to start driving without following these steps.

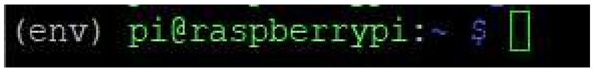
The following instructions are for a Xbox controller.

1. You should use one of the remote desktop top software with a GUI for this step. Navigate to your file system on the Raspberry Pi. Go to myconfig.py and find

CONTROLLER_TYPE. There will be a list of string identifiers after that line of code; make sure to leave those commented out meaning keep a # in front of them. The string identified will have a list of controllers, but in your case you are either selecting ‘Xbox’ or ‘F710’. Type the string that represents the controller you selected after CONTROLLER_TYPE. Make sure to include the ‘’ for whatever item you selected. The # before CONTROLLER_TYPE should be removed. Finally save your changes and exit. Reboot the Pi. Below is this step broken down into smaller chunks.

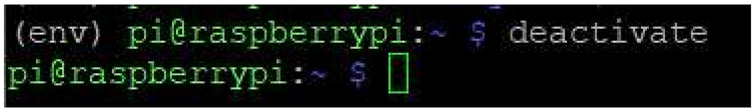
  a. Open the “myconfig.py” file in /home/pi/mycar
  b. Find the line that starts with “# CONTROLLER_TYPE=’ps3’” (should be line 120)
  c. Replace ‘ps3’ with ‘xbox’ and get rid of the # symbol at the beginning of the line. Make sure there is no space in front of
     i. CONTROLLER_TYPE=’xbox’
  d. Save the file, and reboot the pi

2. A linux driver for Xbox Wireless Controller should come pre-installed on Raspbian
3. If you’re in the environment(“(env)” is at the beginning of the terminal), then type “deactivate” and then hit enter



```
(env) pi@raspberrypi:~ $ 
```

Terminal in environment



```
(env) pi@raspberrypi:~ $ deactivate
pi@raspberrypi:~ $ 
```

Exiting the environment

Figure 50. Terminal exiting virtual environment

4. Type “cd” and hit enter
5. sudo nano /etc/modprobe.d/xbox_bt.conf
6. In the nano file paste:

```
options bluetooth disable_ertm=1
```

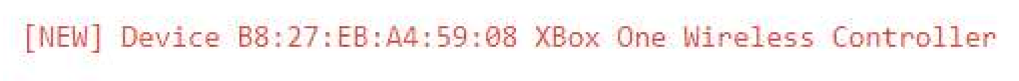
7. The press ctrl + x and then y and then enter
8. Then reboot the pi
9. Check that disable_ertm is set to true by pasting this into the root terminal
    a. First, deactivate the environment
    b. Then, Paste this command if it returns ‘Y’ then you did it correctly

```
cat /sys/module/bluetooth/parameters/disable_ertm
```

10. Now type the following command into the terminal

```
sudo bluetoothctl
```

11. Then type in the following commands
    a. Agent On
    b. default-agent
    c. Scan on
12. Turn on the Xbox one controller and Press the sync button on the front of the controller Pictures
13. Should see a MAC address similar to the one below



```
[NEW] Device B8:27:EB:A4:59:08 XBox One Wireless Controller
```

Figure 50.2: MAC Address Example

    a. Once you see the xbox one wireless controller pop up, type “scan off” to stop the pi from scanning for more addresses

14. Type “connect YOUR_MAC_ADDRESS”
15. The big X button (the home button) on the controller should be solid white
16. Enter in the command “trust YOUR_MAC_ADDRESS”
17. Then type quit
18. Joystick is connected!

Calibrating Car

1. Make sure the car is turned on not just the raspi, also make sure the car wheels have no contact with the surface so that the car does not drive off during calibration
2. At the root of the path you can edit car values one you enter the command
   Nano ~/mycar/myconfig.py
3. Steering Calibration:
   a. Find the servo cable on your car and see what channel it's plugged into on the arduino shield
   b. Enter the command
      Donkey calibrate --channel <your_steering_channel> --bus=1 (your steering channel which is either 1 or 0)
   c. Enter 360 and you should see the wheels on the car move slightly
   d. Enter values +/- 10 for your starting value to find the PWM settings that mak your car turn all the way left and all the way right. Take not of those values
   e. Enter the myconfig.py file and enter in the values to the STEERING_RIGHT_PWM and STEERING_LEFT_PWM
      i. Make sure to uncomment these lines of code

Also, if you try to drive your car, and the steering is reversed, set the Joystick Steering Scale to be negative in the myconfig.py file

Throttle Calibration

1. Repeat the same a -b excepted us the channel that you did not use last time(so if you used 0 go to 1 or vice versa)
2. Enter 370 when prompted for a PWM value; you should hear the ESC beep indicating that it’s calibrated

3. Enter 400 and you should see the car's wheel move forward; if not its likely in reverse so enter 330 instead
4. Keep trying different values until you have found a reasonable max speed for the car
5. Once you have your values open myconfig.py and enter the PWM values for the car into the throttle_controller part:

   `THROTTLE_FORWARD_PWM` = PWM value for full throttle forward

   `THROTTLE_STOPPED_PWM` = PWM value for zero throttle

   `THROTTLE_REVERSE_PWM` = PWM value at full reverse throttle

Driving Car

1. Navigate to the mycar directory
2. Enter this command into the terminal command 1 is if you want to drive without a joystick and command 2 if you want to drive with a joystick

   1.Python manage.py drive

   2.`python manage.py drive` *--js*

   a. Note: Make sure you turn on the car's esc after entering this command
3. Driving without the Joystick:
   a. Enter *ip address of your car*:8887 into a web browser. If should bring you into a window like the image below

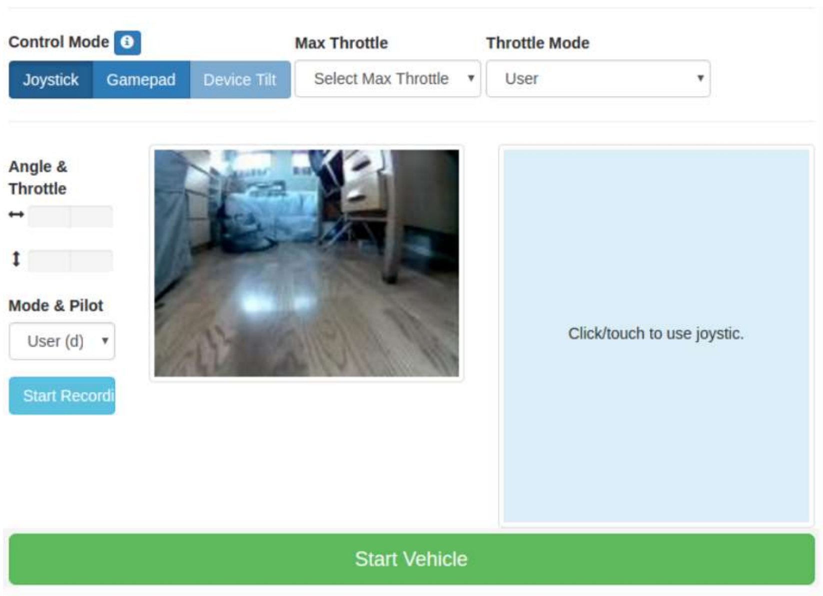


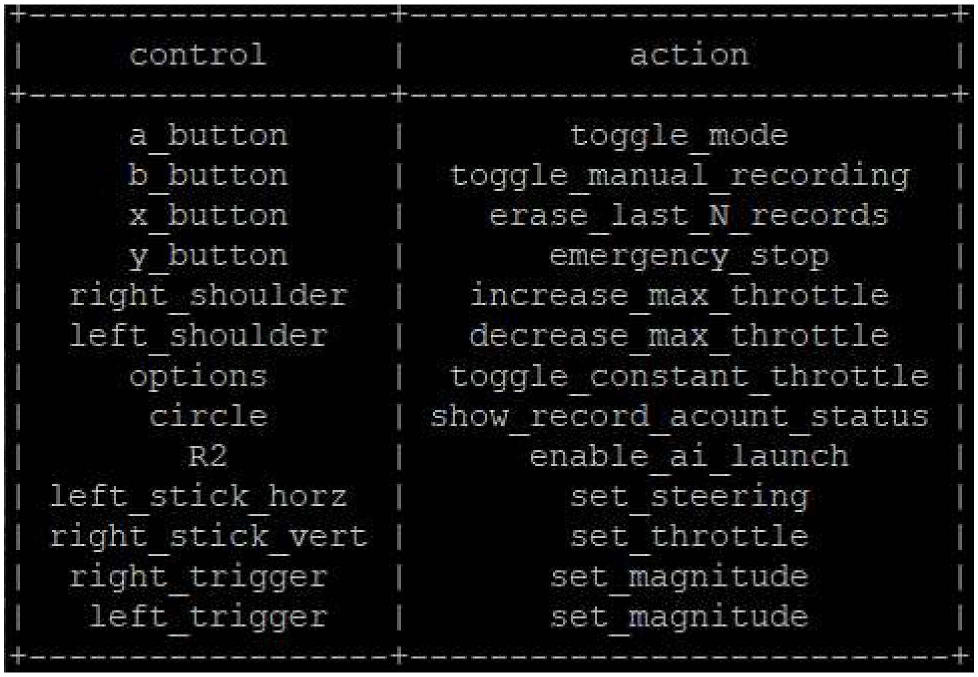
Figure 51: Snapshot of DonkeyCar web application that shows the camera at a ground view

b. In modes you can switch between different ways of driving whether you want the user to drive the whole car, or the car does all the steering, or you want to drive from the computer. If you want to drive from the computer you can use the joystick pad or used the following keyboard shortcuts

   i. space : stop car and stop recording
   ii. r : toggle recording
   iii. i : increase throttle
   iv. k : decrease throttle
   v. j : turn left
   vi. l : turn right

4. Driving with the joystick:

   a. Turn on your joystick(Xbox 1 controller) and you should be able to start driving
   b. The controls should be listed in the terminal



```
+------------------+---------------------------+
|     control      |          action           |
+------------------+---------------------------+
|     a_button     |        toggle_mode        |
|     b_button     |  toggle_manual_recording  |
|     x_button     |    erase_last_N_records   |
|     y_button     |       emergency_stop      |
|  right_shoulder  |   increase_max_throttle   |
|  left_shoulder   |   decrease_max_throttle   |
|     options      |  toggle_constant_throttle |
|      circle      | show_record_acount_status |
|        R2        |      enable_ai_launch     |
| left_stick_horz  |        set_steering       |
| right_stick_vert |        set_throttle       |
|  right_trigger   |       set_magnitude       |
|   left_trigger   |       set_magnitude       |
+------------------+---------------------------+
```

Figure 52: Example of controller indication running on DonkeyCar.

i. Set_throttle is for the throttle
ii. Set_steering is for steering
iii. erase_last_N_records is for getting rid of some of the training records that you just took - say you drive off of the track, press this button so the car doesn't save that as part of your training data

Note: DonkeyCar automatically collects training data when the throttle is not 0

Collecting Training Data

Tips

1. 10 - 20 laps
2. Divide training data into 4 different styles
    2.1. Accuracy not worried about speed roughly 10% of data - 2 laps
    2.2. Small oscillations around the center so that the car can see other parts of the track and learn to correct itself - 2-3 laps
    2.3. Balance back and forth between the extremes of the lanes, don't overcorrect, stay on one side, and then go to the other - 2-3 laps

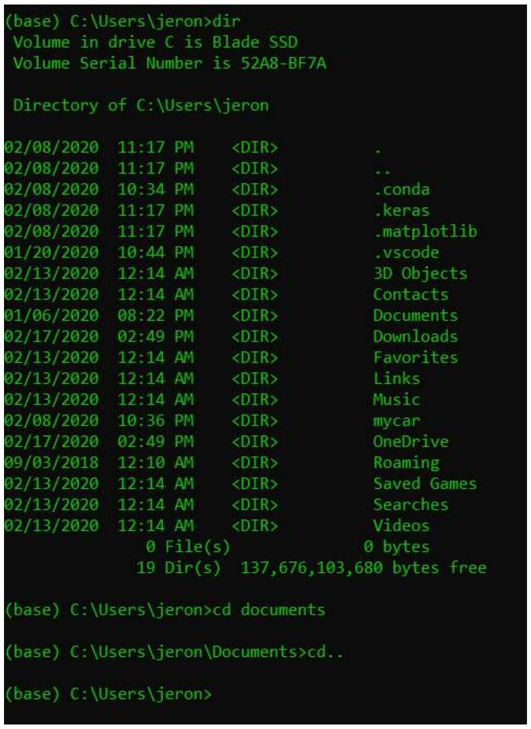
    2.4. Drive normally, so the model learns to drive faster - 4 laps

3. Try to get 5k to 10k records
    3.1. Note: Donkey car takes a photo about every 3- 5 seconds when driving
4. When driving if you crash or run off the course tap Y on the joystick and the last 5 seconds of records will be erased. During the drive the data is saved to a data folder(the most recent tub folder)
5. Go to Training AI if you are ready to start training the Donkey Car.

Installing Software for Training Car on Host Computer

Basic Windows Terminal Commands:

cd (FolderName)	Changes to the folder that you desire to go to in the desired directory(file path).
cd..	This allows you to back one level in your directory.
mkdir	This makes a new folder in the directory that you are currently working in.
dir	This command lists the files that are in the current folder that you are in.



```
(base) C:\Users\jeron>dir
 Volume in drive C is Blade SSD
 Volume Serial Number is 52A8-BF7A

 Directory of C:\Users\jeron

02/08/2020  11:17 PM    <DIR>          .
02/08/2020  11:17 PM    <DIR>          ..
02/08/2020  10:34 PM    <DIR>          .conda
02/08/2020  11:17 PM    <DIR>          .keras
02/08/2020  11:17 PM    <DIR>          .matplotlib
01/20/2020  10:44 PM    <DIR>          .vscode
02/13/2020  12:14 AM    <DIR>          3D Objects
02/13/2020  12:14 AM    <DIR>          Contacts
01/06/2020  08:22 PM    <DIR>          Documents
02/17/2020  02:49 PM    <DIR>          Downloads
02/13/2020  12:14 AM    <DIR>          Favorites
02/13/2020  12:14 AM    <DIR>          Links
02/13/2020  12:14 AM    <DIR>          Music
02/08/2020  10:36 PM    <DIR>          mycar
02/17/2020  02:49 PM    <DIR>          OneDrive
09/03/2018  12:10 AM    <DIR>          Roaming
02/13/2020  12:14 AM    <DIR>          Saved Games
02/13/2020  12:14 AM    <DIR>          Searches
02/13/2020  12:14 AM    <DIR>          Videos
               0 File(s)              0 bytes
              19 Dir(s)  137,676,103,680 bytes free

(base) C:\Users\jeron>cd documents

(base) C:\Users\jeron\Documents>cd..

(base) C:\Users\jeron>
```

Figure 53: Example of the usage of some basic windows terminal commands

1. Install miniconda Python 3.7; choose the installer that matches your machine. For example, if your computer has a Windows computer with 64-bit operating system then you would choose Miniconda3 Windows 64-bit.
   - https://docs.conda.io/en/latest/miniconda.html
2. Once downloaded open the Anaconda Prompt via Start Menu
3. Type git in the prompt, if the command is not found then install git
   - https://git-scm.com/download/win
4. In the anaconda terminal type
   - mkdir projects
   - cd projects
5. Get the latest donkey from Github by typing

```
git clone https://github.com/autorope/donkeycar
cd donkeycar
```

```
git checkout master
```

- If this is not your first install type:

  ```
  conda update -n base -c defaults conda
  conda env remove -n donkey
  ```

6. Create the python anaconda environment:

   ```
   conda env create -f install\envs\windows.yml
   conda activate donkey
   pip install -e .[pc]
   ```

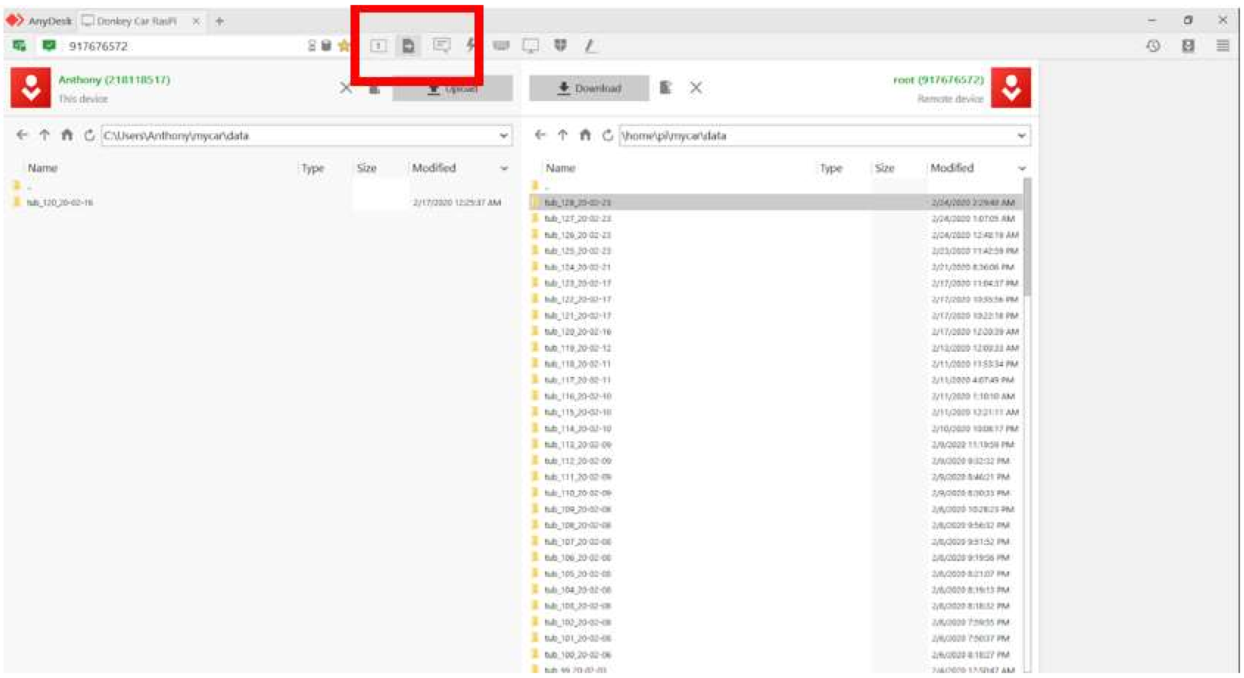
7. create your local working directory

   ```
   donkey createcar --path ~/mycar
   ```

8. Your Host Computer should be all set!

Training AI:

1. Once you have finished recording data go hit crtl+C in the donkey car terminal to get it to shut off
2. The get into the pi through Anydesk
   a. Click the button in the figure below



C:\Users\Anthony\mycar\data

Name	Type	Size	Modified
..			
tub_120_20-02-16			2/17/2020 12:25:37 AM

\home\pi\mycar\data

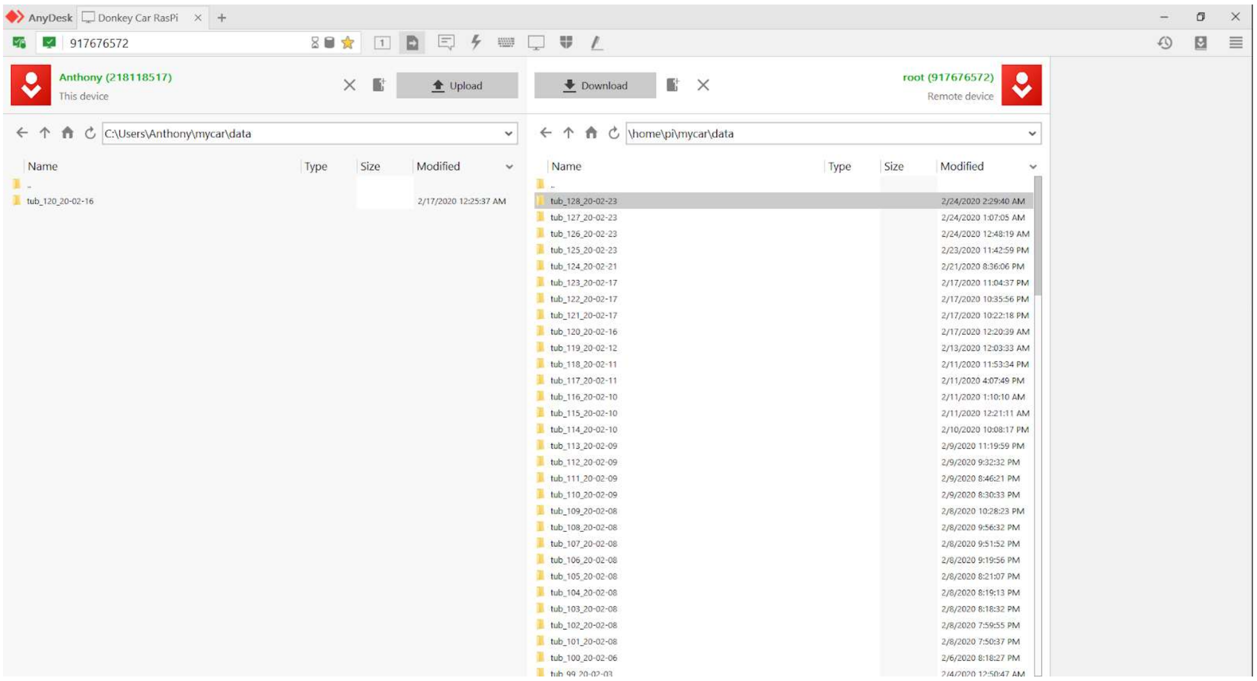
Name	Type	Size	Modified
..			
tub_128_20-02-23			2/24/2020 2:29:48 AM
tub_127_20-02-23			2/24/2020 1:07:05 AM
tub_126_20-02-23			2/24/2020 12:42:18 AM
tub_125_20-02-23			2/23/2020 11:42:59 PM
tub_124_20-02-21			2/21/2020 8:36:06 PM
tub_123_20-02-17			2/17/2020 11:04:37 PM
tub_122_20-02-17			2/17/2020 10:55:56 PM
tub_121_20-02-17			2/17/2020 10:22:18 PM
tub_120_20-02-16			2/17/2020 12:20:39 AM
tub_119_20-02-12			2/12/2020 12:09:22 AM
tub_118_20-02-11			2/11/2020 11:53:34 PM
tub_117_20-02-11			2/11/2020 4:07:49 PM
tub_116_20-02-10			2/11/2020 1:10:10 AM
tub_115_20-02-10			2/11/2020 12:21:11 AM
tub_114_20-02-10			2/10/2020 10:08:17 PM
tub_113_20-02-09			2/9/2020 11:19:58 PM
tub_112_20-02-09			2/9/2020 9:32:32 PM
tub_111_20-02-09			2/9/2020 8:46:21 PM
tub_110_20-02-09			2/9/2020 8:30:33 PM
tub_109_20-02-08			2/8/2020 10:28:23 PM
tub_108_20-02-08			2/8/2020 9:56:32 PM
tub_107_20-02-08			2/8/2020 9:51:52 PM
tub_106_20-02-08			2/8/2020 9:19:56 PM
tub_105_20-02-08			2/8/2020 8:21:07 PM
tub_104_20-02-08			2/8/2020 8:19:13 PM
tub_103_20-02-08			2/8/2020 8:18:32 PM
tub_102_20-02-08			2/8/2020 7:59:55 PM
tub_101_20-02-08			2/8/2020 7:50:37 PM
tub_100_20-02-06			2/6/2020 8:18:27 PM
tub_99_20-02-01			2/4/2020 12:50:47 AM

   b. Go to the **home/pi//mycar/data** directory

i. And select the recent tub files that you would like to transfer

c. On your host computer navigate to the folder where your car is located and enter into the data folder

d. Once navigated in the correct locations of both your host computer and the raspi; with raspi files selected press upload



C:\Users\Anthony\mycar\data

Name	Type	Size	Modified
..			
tub_120_20-02-16			2/17/2020 12:25:37 AM

\home\pi\mycar\data

Name	Type	Size	Modified
..			
tub_128_20-02-23			2/24/2020 2:29:40 AM
tub_127_20-02-23			2/24/2020 1:07:05 AM
tub_126_20-02-23			2/24/2020 12:48:19 AM
tub_125_20-02-23			2/23/2020 11:42:59 PM
tub_124_20-02-21			2/21/2020 8:36:06 PM
tub_123_20-02-17			2/17/2020 11:04:37 PM
tub_122_20-02-17			2/17/2020 10:35:56 PM
tub_121_20-02-17			2/17/2020 10:22:18 PM
tub_120_20-02-16			2/17/2020 12:20:39 AM
tub_119_20-02-12			2/13/2020 12:03:33 AM
tub_118_20-02-11			2/11/2020 11:53:34 PM
tub_117_20-02-11			2/11/2020 4:07:49 PM
tub_116_20-02-10			2/11/2020 1:10:10 AM
tub_115_20-02-10			2/11/2020 12:21:11 AM
tub_114_20-02-10			2/10/2020 10:08:17 PM
tub_113_20-02-09			2/9/2020 11:19:59 PM
tub_112_20-02-09			2/9/2020 9:32:32 PM
tub_111_20-02-09			2/9/2020 8:46:21 PM
tub_110_20-02-09			2/9/2020 8:30:33 PM
tub_109_20-02-08			2/8/2020 10:28:23 PM
tub_108_20-02-08			2/8/2020 9:56:32 PM
tub_107_20-02-08			2/8/2020 9:51:52 PM
tub_106_20-02-08			2/8/2020 9:19:56 PM
tub_105_20-02-08			2/8/2020 8:21:07 PM
tub_104_20-02-08			2/8/2020 8:19:13 PM
tub_103_20-02-08			2/8/2020 8:18:32 PM
tub_102_20-02-08			2/8/2020 7:59:55 PM
tub_101_20-02-08			2/8/2020 7:50:37 PM
tub_100_20-02-06			2/6/2020 8:18:27 PM
tub_99_20-02-03			2/4/2020 12:50:47 AM

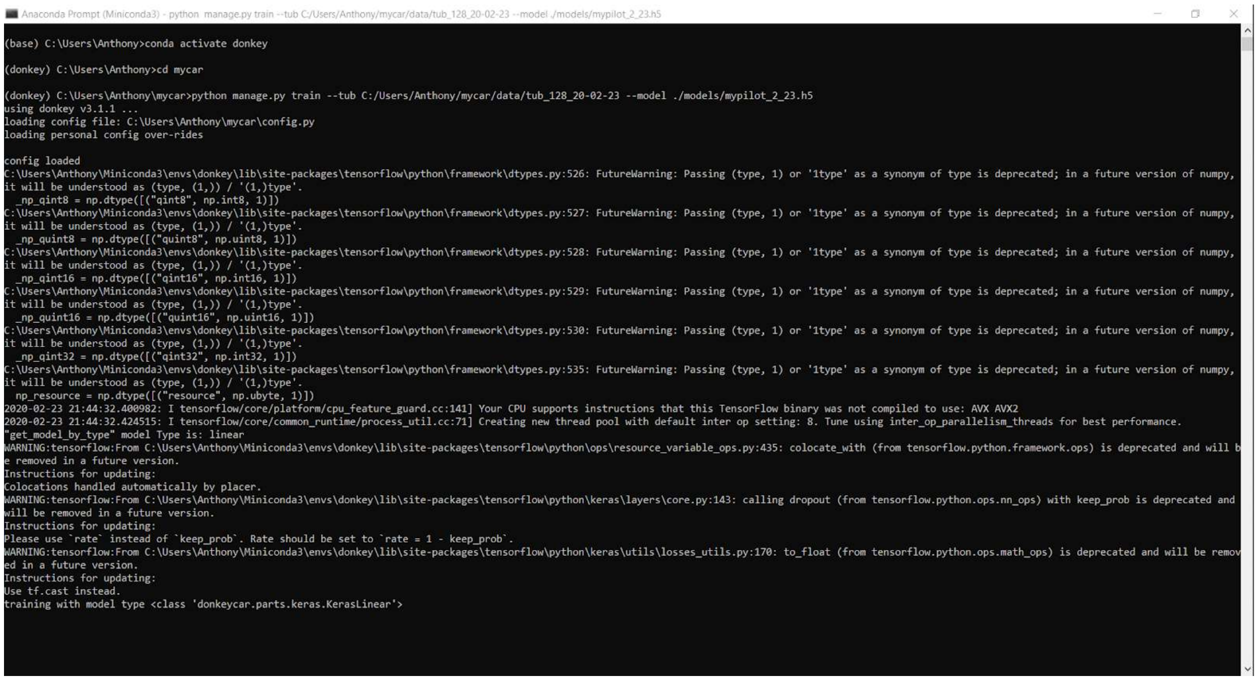
3. Once the files are uploaded onto your home computer

a. Navigate to the location of your mycar folder on your host computer and type the following command into the terminal:

```
python manage.py train --tub C:/Users/ --model models/mypilot.h5
```

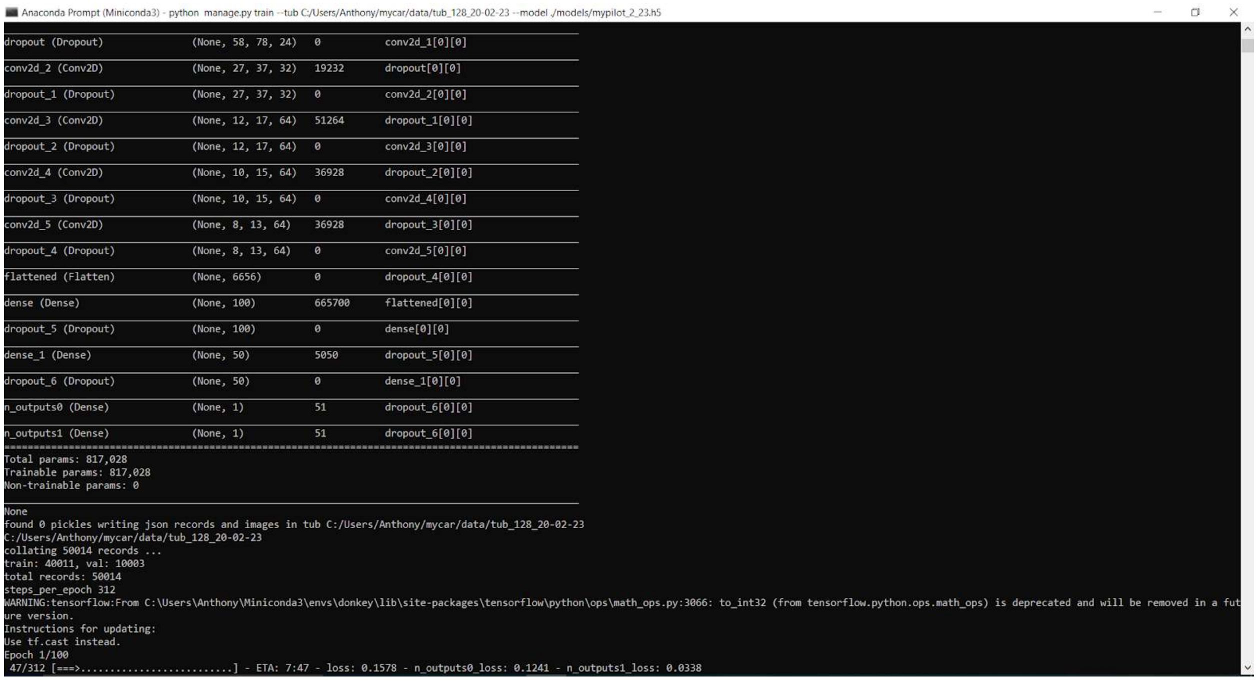
You can use multiple tubs comma separated.

b. You then should have a screen that showing the NN training that looks like the images below:



Anaconda Prompt (Miniconda3) - python manage.py train --tub C:/Users/Anthony/mycar/data/tub_128_20-02-23 --model ./models/mypilot_2_23.h5

```
(base) C:\Users\Anthony>conda activate donkey

(donkey) C:\Users\Anthony>cd mycar

(donkey) C:\Users\Anthony\mycar>python manage.py train --tub C:/Users/Anthony/mycar/data/tub_128_20-02-23 --model ./models/mypilot_2_23.h5
using donkey v3.1.1 ...
loading config file: C:\Users\Anthony\mycar\config.py
loading personal config over-rides

config loaded
C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\framework\dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\framework\dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\framework\dtypes.py:528: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\framework\dtypes.py:529: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\framework\dtypes.py:530: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\framework\dtypes.py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
2020-02-23 21:44:32.400982: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2
2020-02-23 21:44:32.424515: I tensorflow/core/common_runtime/process_util.cc:71] Creating new thread pool with default inter op setting: 8. Tune using inter_op_parallelism_threads for best performance.
"get_model_by_type" model Type is: linear
WARNING:tensorflow:From C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from tensorflow.python.framework.ops) is deprecated and will b
e removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\keras\layers\core.py:143: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and
will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\keras\utils\losses_utils.py:170: to_float (from tensorflow.python.ops.math_ops) is deprecated and will be remov
ed in a future version.
Instructions for updating:
Use tf.cast instead.
training with model type <class 'donkeycar.parts.keras.KerasLinear'>
```



Anaconda Prompt (Miniconda3) - python manage.py train --tub C:/Users/Anthony/mycar/data/tub_128_20-02-23 --model ./models/mypilot_2_23.h5

```
dropout (Dropout)            (None, 58, 78, 24)   0          conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, 27, 37, 32)   19232      dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)          (None, 27, 37, 32)   0          conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 17, 64)   51264      dropout_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)          (None, 12, 17, 64)   0          conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 15, 64)   36928      dropout_2[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)          (None, 10, 15, 64)   0          conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 13, 64)    36928      dropout_3[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)          (None, 8, 13, 64)    0          conv2d_5[0][0]
__________________________________________________________________________________________________
flattened (Flatten)          (None, 6656)         0          dropout_4[0][0]
__________________________________________________________________________________________________
dense (Dense)                (None, 100)          665700     flattened[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)          (None, 100)          0          dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)              (None, 50)           5050       dropout_5[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)          (None, 50)           0          dense_1[0][0]
__________________________________________________________________________________________________
n_outputs0 (Dense)           (None, 1)            51         dropout_6[0][0]
__________________________________________________________________________________________________
n_outputs1 (Dense)           (None, 1)            51         dropout_6[0][0]
==================================================================================================
Total params: 817,028
Trainable params: 817,028
Non-trainable params: 0
__________________________________________________________________________________________________
None
found 0 pickles writing json records and images in tub C:/Users/Anthony/mycar/data/tub_128_20-02-23
C:/Users/Anthony/mycar/data/tub_128_20-02-23
collating 50014 records ...
train: 40011, val: 10003
total records: 50014
steps_per_epoch 312
WARNING:tensorflow:From C:\Users\Anthony\Miniconda3\envs\donkey\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a fut
ure version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/100
 47/312 [===>..........................] - ETA: 7:47 - loss: 0.1578 - n_outputs0_loss: 0.1241 - n_outputs1_loss: 0.0338
```

c. Once it is training the terminal will let you know when it is finished expect anywhere between 1 - 2 hours for 30 min - 1 hours worth of training data.

d. Once it is complete go to the models folder and ***upload*** the mypilot.h5 file onto the Pi's model folder; following the previous steps that ***download*** the data onto your computer.

e. In a terminal on the raspi (in the environment), enter:
    i. cd ~/mycar
    ii. python manage.py drive --model ~/mycar/models/mypilot_2.h5
        1. Change mypilot_2.h5 to the model that you just trained
f. Then enter the web controller:
    i. Enter the raspberry pi's ip address followed by :8887 into your web browser
    ii. Ex: 130.215.221.24:8887
g. Under mode and pilot, select local pilot, and watch it drive!
h. Once back on the Raspberry Pi enter into theRaspberry Pi terminal the command below:

   Please note that the below command will automatically get the car to start driving on its own: make sure that the car is on the ground and have the car's webpage ***car's ipaddress:8887*** *ready to press the stop driving button, and be ready to catch/ turn of the esc in the car in case it rams into a wall.*

```
python manage.py drive --model ~/mycar/models/mypilot.h5
```

10.2 Sensor User Guide

Modular Sensor Package User Guide

Team Members
Thomas Kim (RBE/ME)
Kyle Wood (ECE)

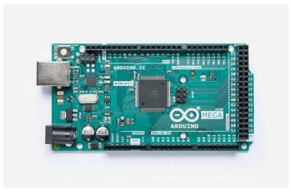
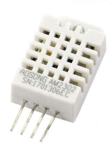
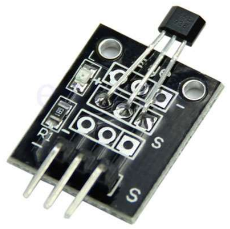
Advisors
Professor Pradeep Radhakrishnan (RBE/ME)
Professor Kaveh Pahlavan (ECE/CS)

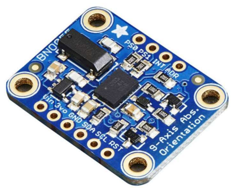
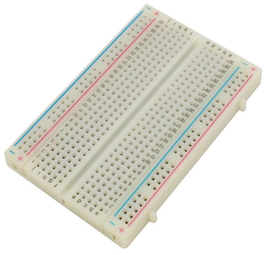
Table of Contents

Introduction

Hello! Welcome to the Sensor Package Setup Guide. Following these instructions will allow you to complete the Arduino and electronic sensor setup for the modular sensor package. This includes the wiring setup for the sensors, and a brief introduction in running and installing the software and code onto your Arduino board to begin the collection of data of five total sensors: three thermal sensors, one inertial measurement unit (IMU), and one hall-effect sensor.

Parts List

Part	Quantity	Picture	Purchasing Link
Arduino Mega	1		Amazon Link
Raspberry Pi 3B+	1		Amazon Link
Micro USB Cable	1		Amazon Link
ASAIR AM2302 Temperature and Humidity Sensors	3		Amazon Link
Hall Effect Magnetic Sensor Module for Arduino	1		Amazon Link

BNO055 9-DOF Absolute Orientation IMU Fusion Breakout Board	1		Amazon Link
Arduino Jumper Wire (assort connection types: Male-Male, Female-Male, Female-Female)	20+		Amazon Link
Solderless Breadboard	1		Amazon Link
Portable Battery Bank	1		Amazon Link

PLA for 3D Printing	1		Amazon Link

Printing the Enclosure

The enclosure is fully designed to be 3D printed, and STL's of the sensor box can be found here. The parts list includes one kilogram of Hatchbox PLA, but does not include a 3D printer, or instructions on how to utilize a 3D printer. If this kit is being completed, but you do not own or have access to a 3D printer, try looking for a local maker space that has printers that can be used. This website has a vast network of makerspaces in the world, and hopefully one is close by. During our testing, we used a 0.10mm layer height with 20% infill, with support over built plate for only the bottom piece.

Arduino Setup

Wire up the Circuit

As demonstrated by the diagram below, wire the circuit as shown.

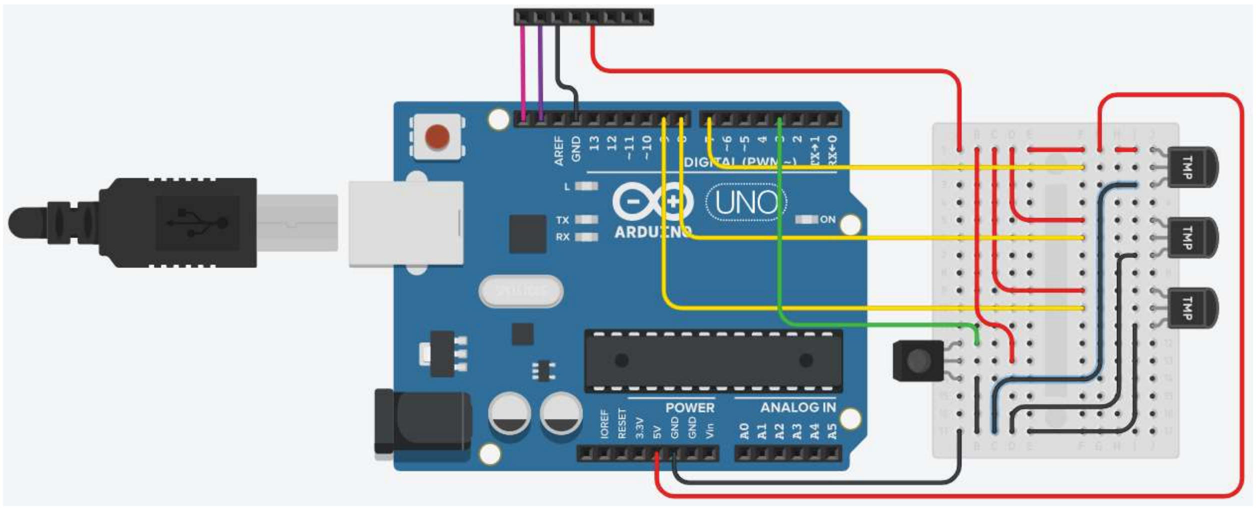


Figure 1: Arduino Sensor Wiring Guide

Red wires are +(positive) power in from the 5V port of the Arduino. The red wires are connected together in parallel, so they all receive the 5V. The 5V wire goes into the board, and branches off to power all of the sensors.

Black wires are –(negative) from the GND port of the Arduino.

Yellow wires are the signal output wires of the thermal sensors, which are sent to individual channels on the Arduino (in this case, the thermal sensors are sent to channels 7, 8, and 9)

Green wires are signal output wires of the Hall-Effect sensor

Pink wires are the SCL wires, which connect the SCL port on the Arduino with the SCL on the IMU

Purple wires are the SDA wires, which connect the SDA port on the Arduino with the SDA on the IMU

The three objects labeled TMP are representative of the thermal sensors and their corresponding pin layouts

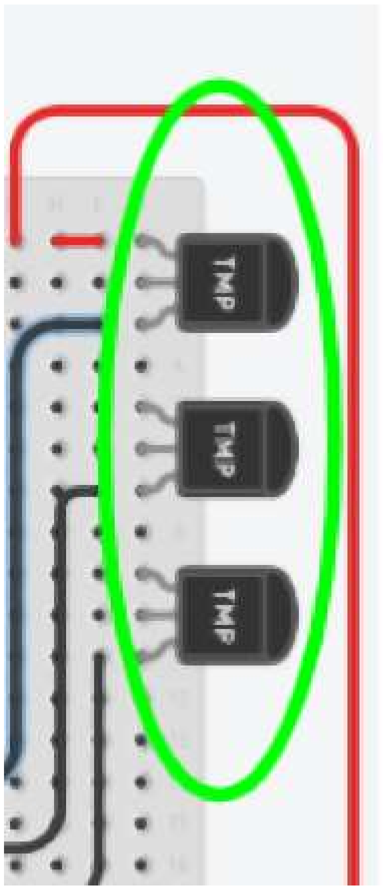


Figure 2: Thermal Sensors in Wire Diagram

The object with the circle in the middle represents the Hall-Effect sensor and its representative pin layout

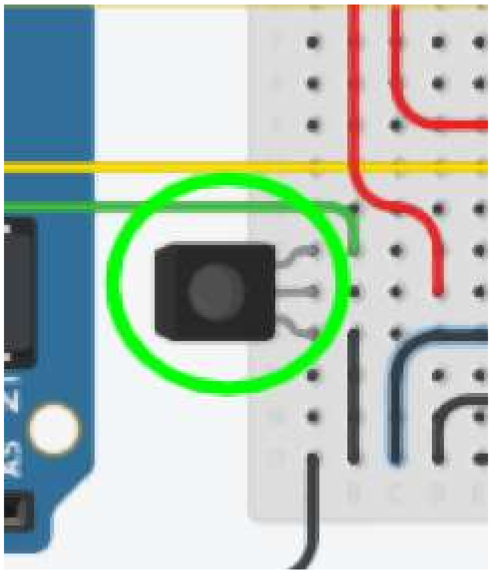


Figure 3: Hall-Effect Sensor in Wire Diagram

The long bar of 8 ports represents the IMU, (although the BNO055 actually has 10 ports, the ones we need are specifically labeled)

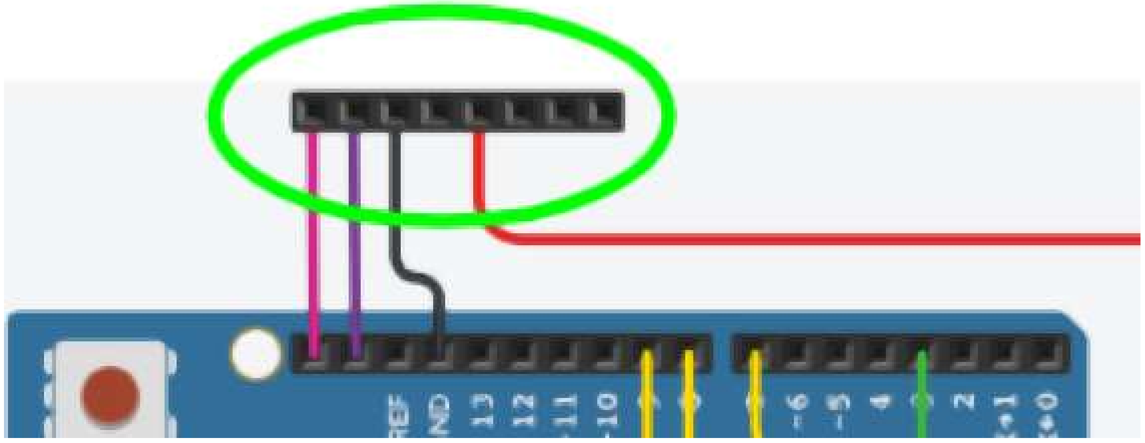


Figure 4: IMU Representation in Wire Diagram

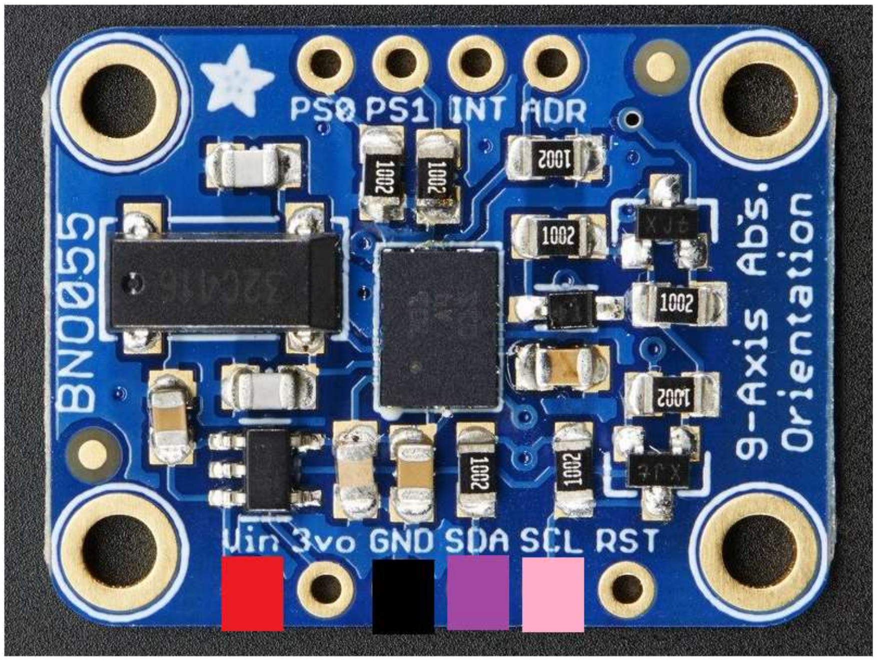


Figure 5: IMU Pins and Corresponding Wire Colors

- To wire up IMU, there needs to be at least 4 pins wires soldered on:
  - Vin
  - GND
  - SDA
  - SCL
- Vin connects in parallel along the red powerline from above, as are the other sensors
- GND, similarly, connects to the GND black wires
- SDA gets plugged directly into the SDA pin on the Arduino board
- SCL gets plugged directly into the SCL pin on the Arduino board

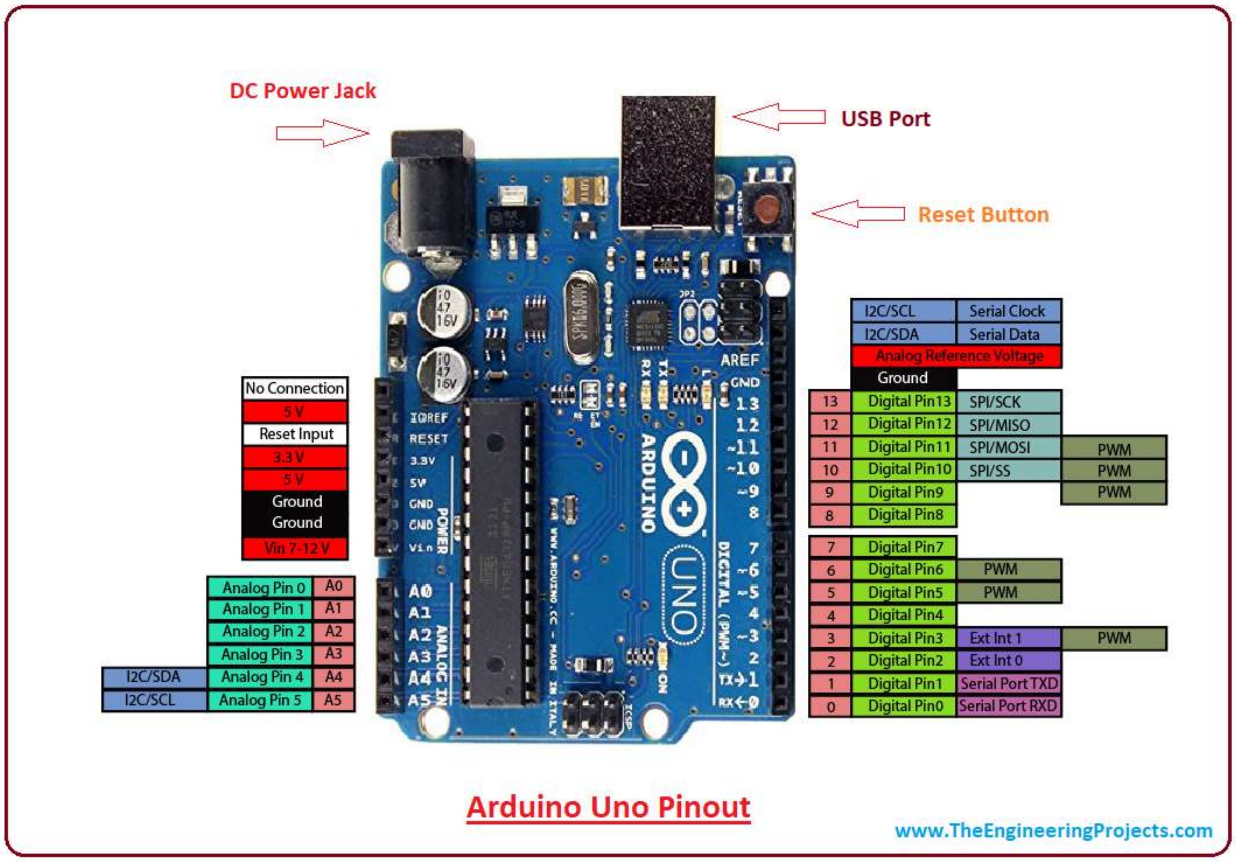
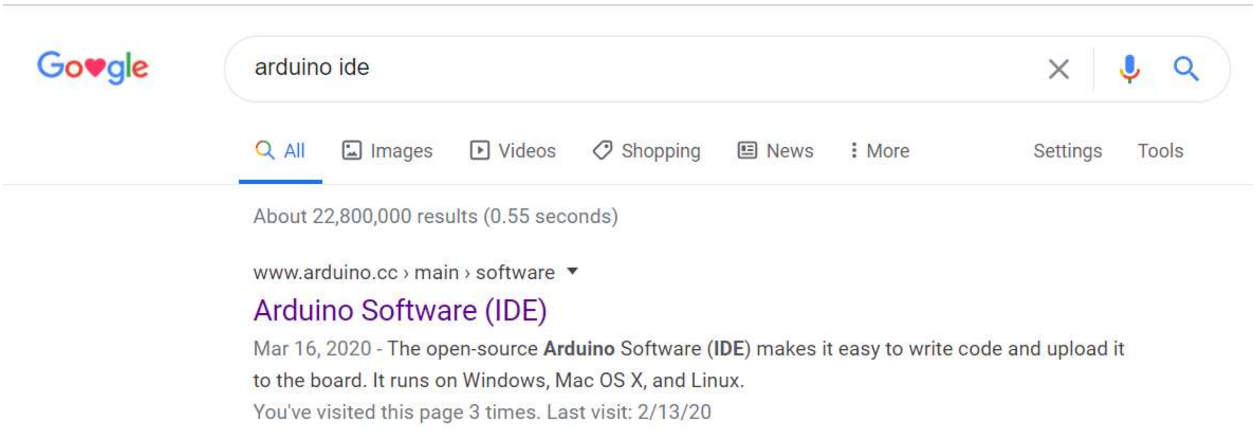


Figure 6: Reference for Arduino Pin Diagram

Arduino IDE

a. Download the Arduino IDE from this link: https://www.arduino.cc/en/main/software



About 22,800,000 results (0.55 seconds)

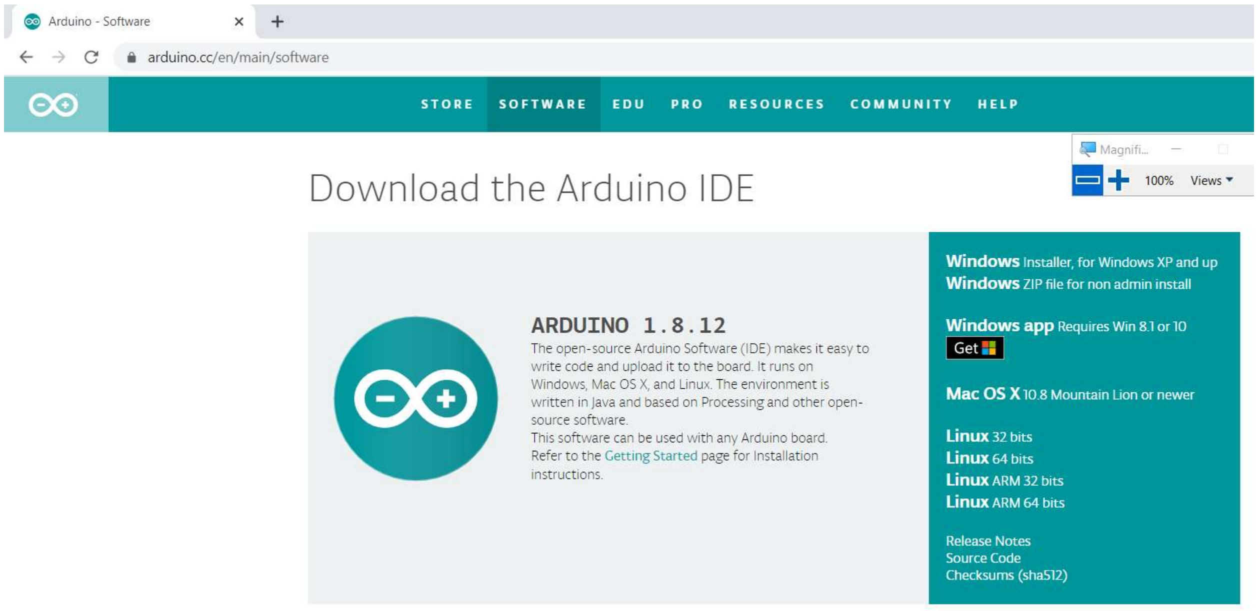
www.arduino.cc › main › software

Arduino Software (IDE)

Mar 16, 2020 - The open-source **Arduino** Software (**IDE**) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux.

You've visited this page 3 times. Last visit: 2/13/20

Figure 7: Finding Arduino IDE through Search Engine



Download the Arduino IDE

ARDUINO 1.8.12

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
Get

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

Release Notes
Source Code
Checksums (sha512)

Figure 8: Arduino IDE Download Page

2.) Download the Sensor_Package.ino into the Arduino IDE from Github

3.) Install these libraries:

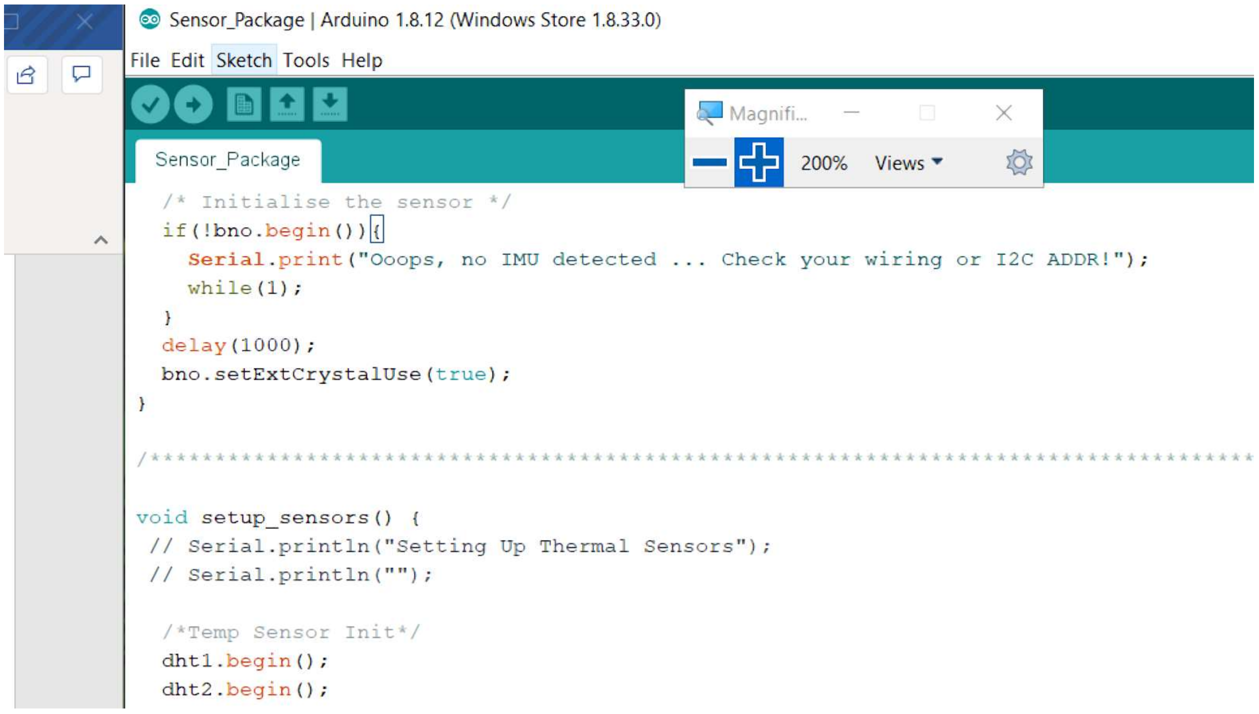
a. BNO055 Libraries

b. dht.h

i. Libraries should be linked in GitHub, this is just in case the download does not work

To add libraries:

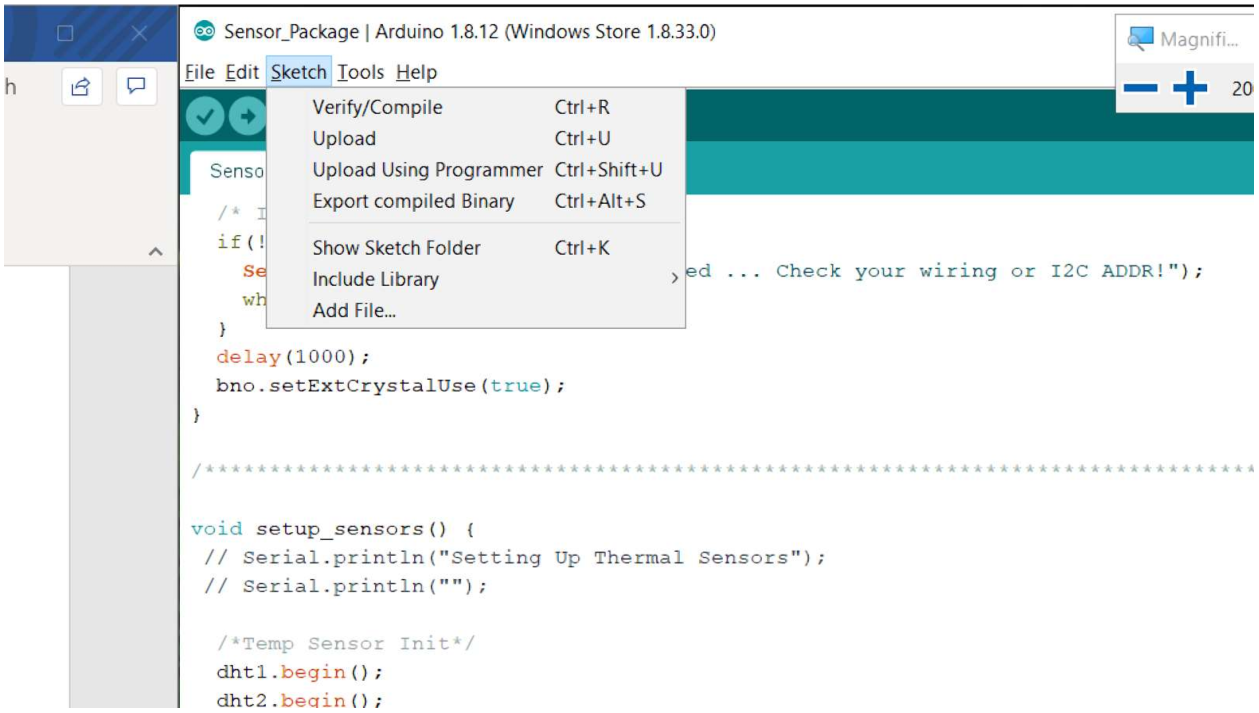
In the Arduino IDE, click "Sketch"


Sensor_Package | Arduino 1.8.12 (Windows Store 1.8.33.0)
File Edit Sketch Tools Help
Sensor_Package
Magnifi...
200%
Views

```
  /* Initialise the sensor */
  if(!bno.begin()){
    Serial.print("Ooops, no IMU detected ... Check your wiring or I2C ADDR!");
    while(1);
  }
  delay(1000);
  bno.setExtCrystalUse(true);
}

/*************************************************************************************

void setup_sensors() {
 // Serial.println("Setting Up Thermal Sensors");
 // Serial.println("");

  /*Temp Sensor Init*/
  dht1.begin();
  dht2.begin();
```

Figure 9: Finding Sketch Option

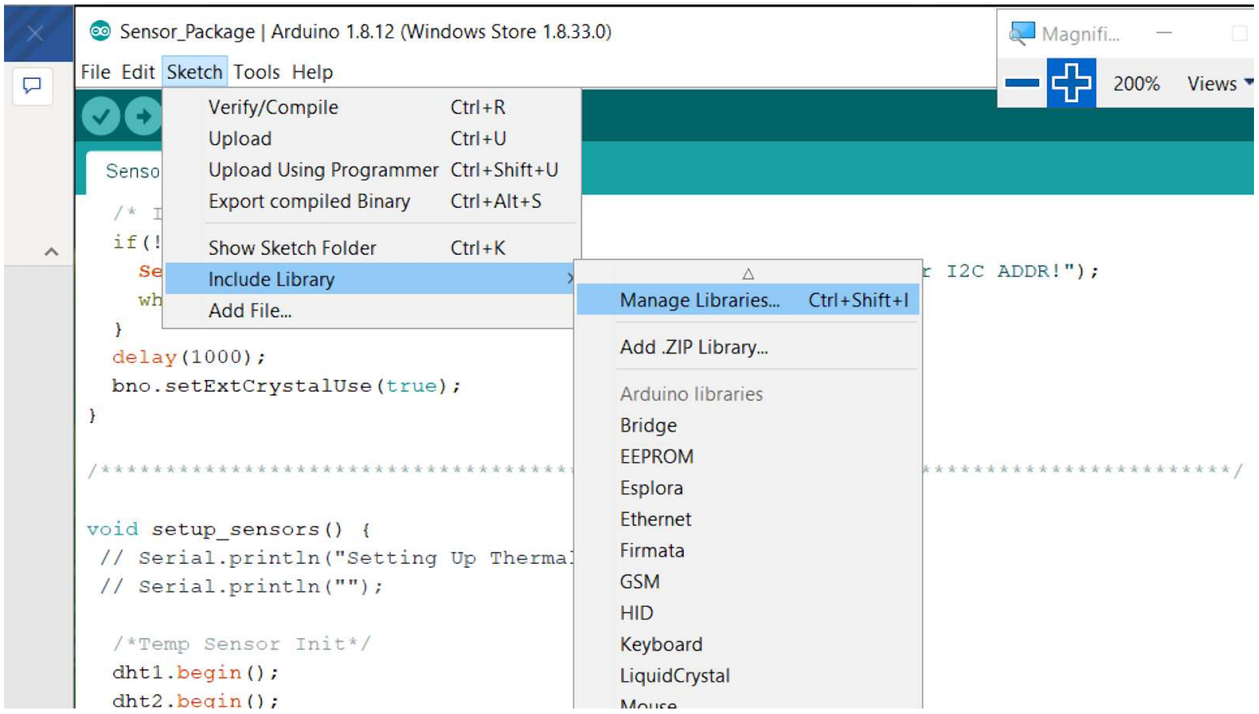
Click "Include Libraries"


Sensor_Package | Arduino 1.8.12 (Windows Store 1.8.33.0)
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Magnifi...

```
  /* I
  if(!
    Se                                          ed ... Check your wiring or I2C ADDR!");
    wh
  }
  delay(1000);
  bno.setExtCrystalUse(true);
}

/*****************************************************************************************

void setup_sensors() {
 // Serial.println("Setting Up Thermal Sensors");
 // Serial.println("");

  /*Temp Sensor Init*/
  dht1.begin();
  dht2.begin();
```

Figure 10: Finding Library Manager in IDE

Click “Manage Libraries”

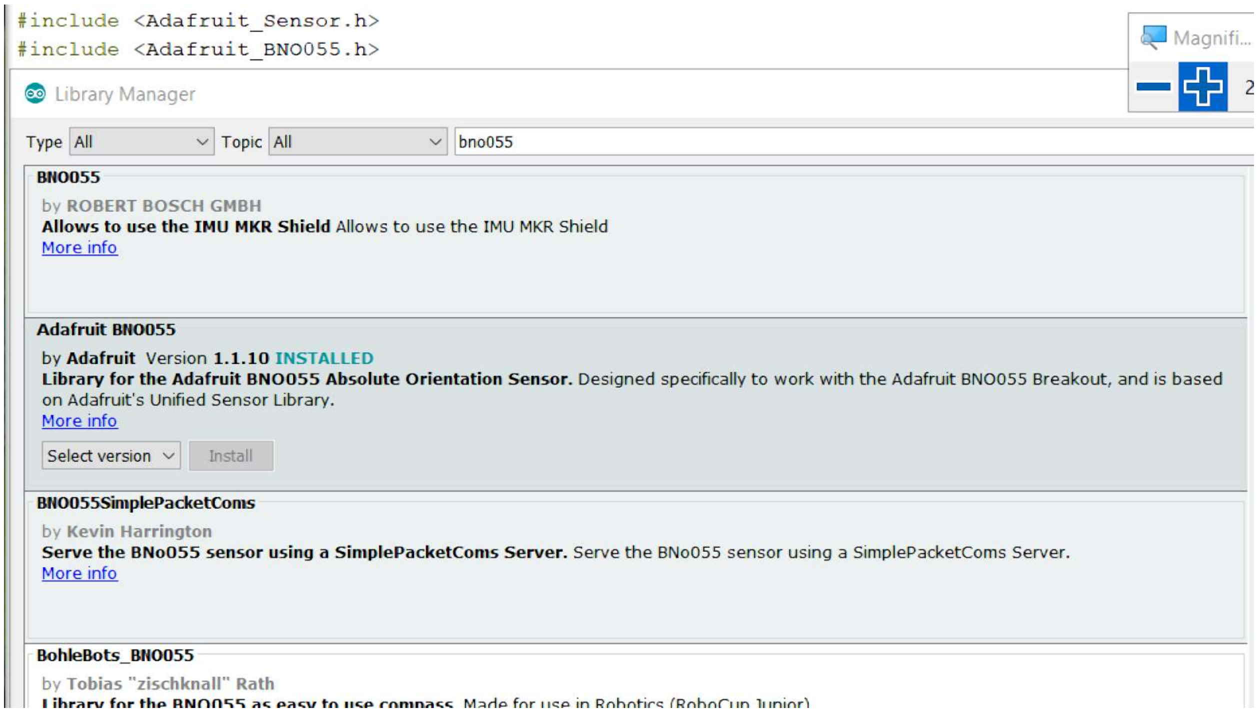

Sensor_Package | Arduino 1.8.12 (Windows Store 1.8.33.0)
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries... Ctrl+Shift+I
Add .ZIP Library...
Arduino libraries
Bridge
EEPROM
Esplora
Ethernet
Firmata
GSM
HID
Keyboard
LiquidCrystal
Magnifi...
200%
Views

```
  /* I
  if(!
    Se                                          r I2C ADDR!");
    wh
  }
  delay(1000);
  bno.setExtCrystalUse(true);
}

/**********************************          ************************/

void setup_sensors() {
 // Serial.println("Setting Up Thermal
 // Serial.println("");

  /*Temp Sensor Init*/
  dht1.begin();
  dht2.begin();
```

Figure 11: Managing Libraries in IDE

Alternatively, you can press the combination CTRL + SHIFT + I to open the "Manage Libraries" window directly from the working environment.

Then, type into the search bar "BNO055" and select the "Adafruit BNO055" Library, and click install



```
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
```

Library Manager

Type All | Topic All | bno055

BNO055
by ROBERT BOSCH GMBH
Allows to use the IMU MKR Shield Allows to use the IMU MKR Shield
More info

Adafruit BNO055
by **Adafruit** Version **1.1.10** INSTALLED
Library for the Adafruit BNO055 Absolute Orientation Sensor. Designed specifically to work with the Adafruit BNO055 Breakout, and is based on Adafruit's Unified Sensor Library.
More info

BNO055SimplePacketComs
by Kevin Harrington
Serve the BNo055 sensor using a SimplePacketComs Server. Serve the BNo055 sensor using a SimplePacketComs Server.
More info

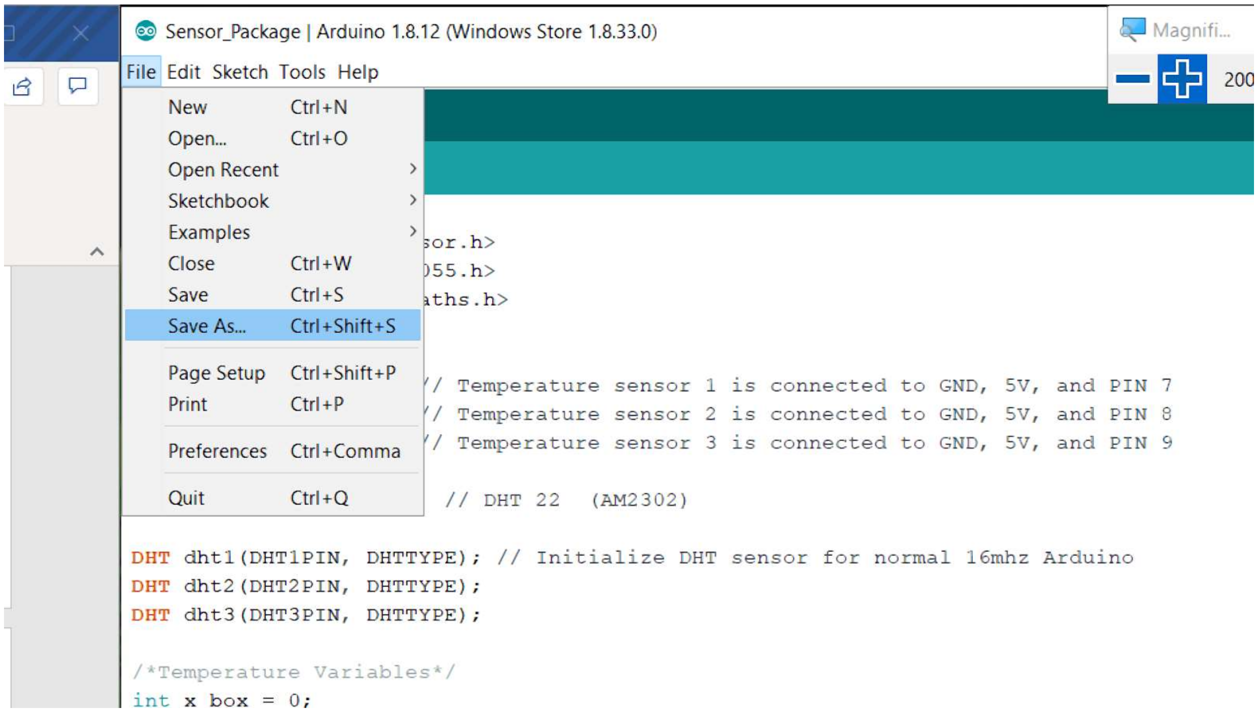
BohleBots_BNO055
by Tobias "zischknall" Rath
Library for the BNO055 as easy to use compass. Made for use in Robotics (RoboCup Junior)

Figure 12: Searching Library in Library Manager

c. If any others are missing, check the #includes, copy and paste them into the search menu in the library, and see what comes up

d. Don't have to worry about any other header files

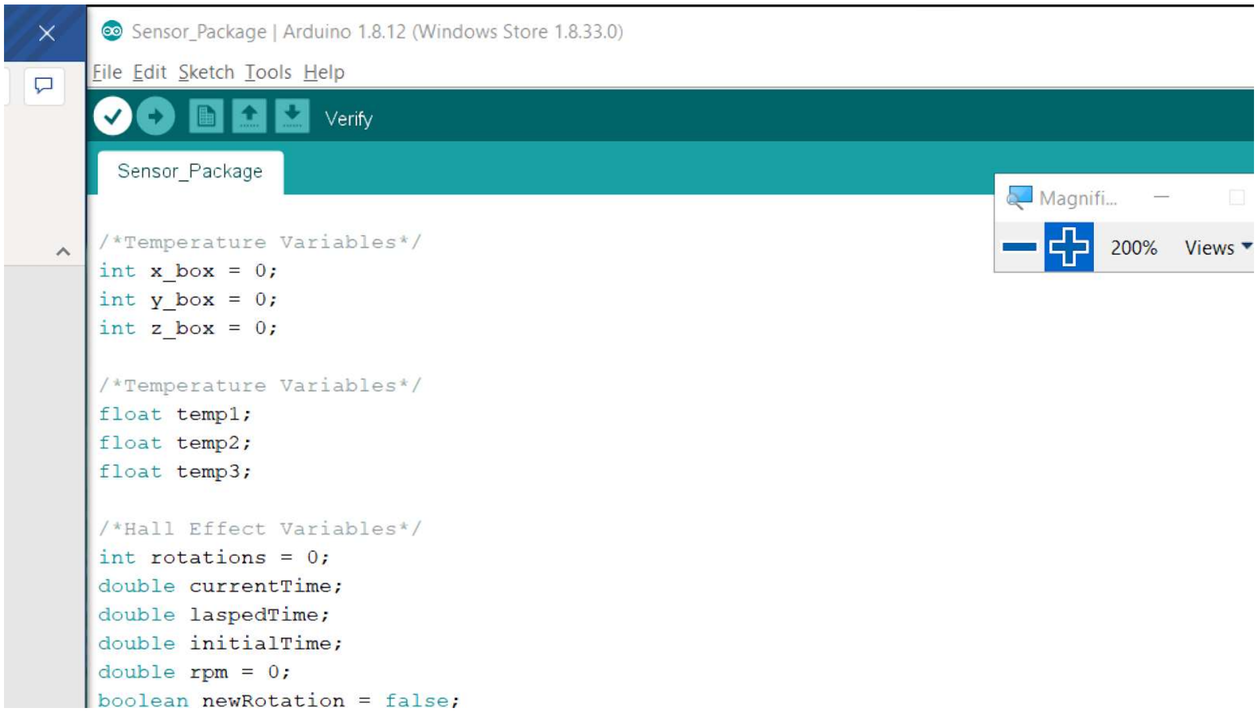
4.) Save the IDE script



```
sor.h>
)55.h>
aths.h>

// Temperature sensor 1 is connected to GND, 5V, and PIN 7
// Temperature sensor 2 is connected to GND, 5V, and PIN 8
// Temperature sensor 3 is connected to GND, 5V, and PIN 9

 // DHT 22  (AM2302)

DHT dht1(DHT1PIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduino
DHT dht2(DHT2PIN, DHTTYPE);
DHT dht3(DHT3PIN, DHTTYPE);

/*Temperature Variables*/
int x box = 0;
```

Figure 13: Saving the IDE

Either save your file in the Arduino folder (usually found in Documents after installing the IDE) or add the directory from download to the path. Similarly, once the project has been saved, clicking "Verify" in the top left corner under "File" will save the code prior to checking it.


Sensor_Package | Arduino 1.8.12 (Windows Store 1.8.33.0)
File Edit Sketch Tools Help
Verify
Sensor_Package
Magnifi...
200%
Views

```
/*Temperature Variables*/
int x_box = 0;
int y_box = 0;
int z_box = 0;

/*Temperature Variables*/
float temp1;
float temp2;
float temp3;

/*Hall Effect Variables*/
int rotations = 0;
double currentTime;
double laspedTime;
double initialTime;
double rpm = 0;
boolean newRotation = false;
```

Figure 14: Trying Verify in IDE

5.) Check PINS (or reassign if necessary)

a. Before you can collect data, make sure the pins are set up properly on the Arduino in the PWM section of the board.

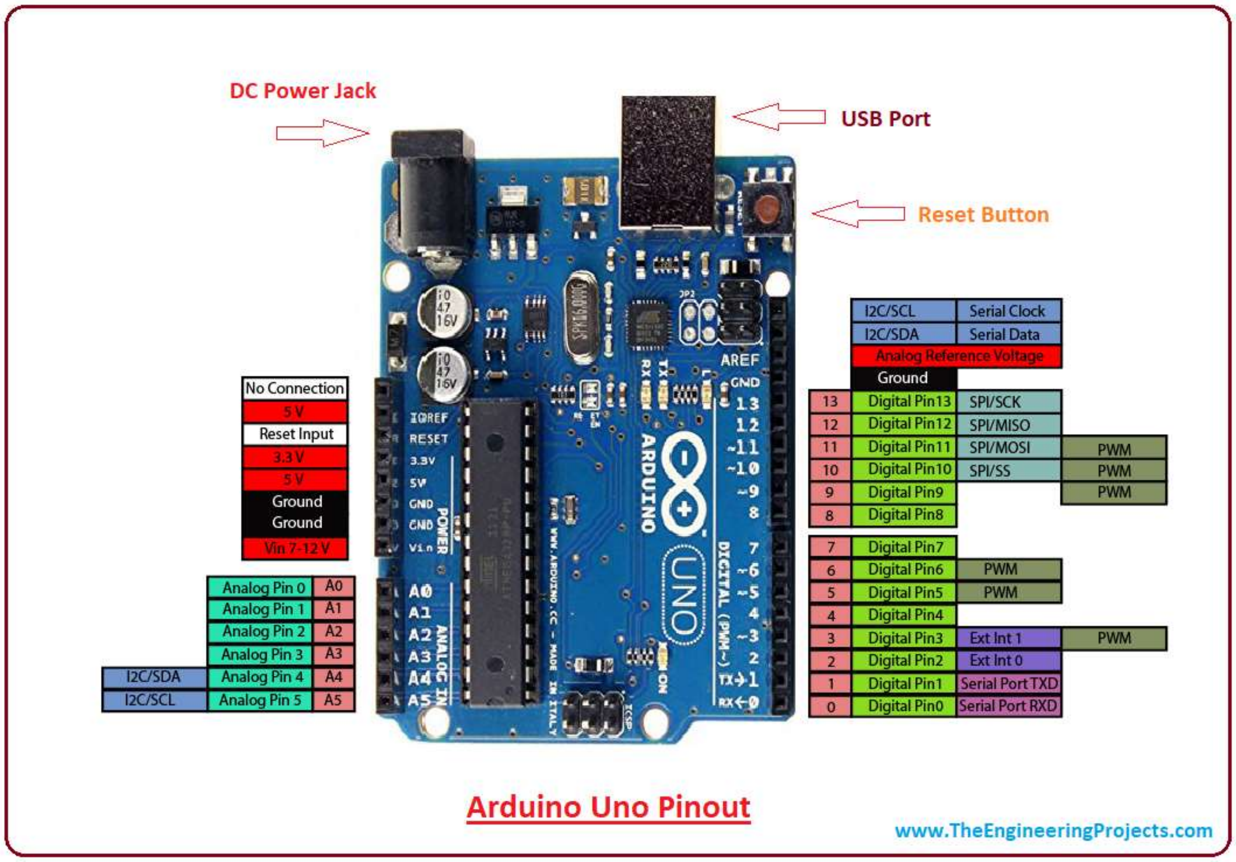


Figure #15: Arduino Pin Reference

b. The pins defined by the script are:

   i. PWM 7, 8, 9 are assigned to the temperature and humidity sensors

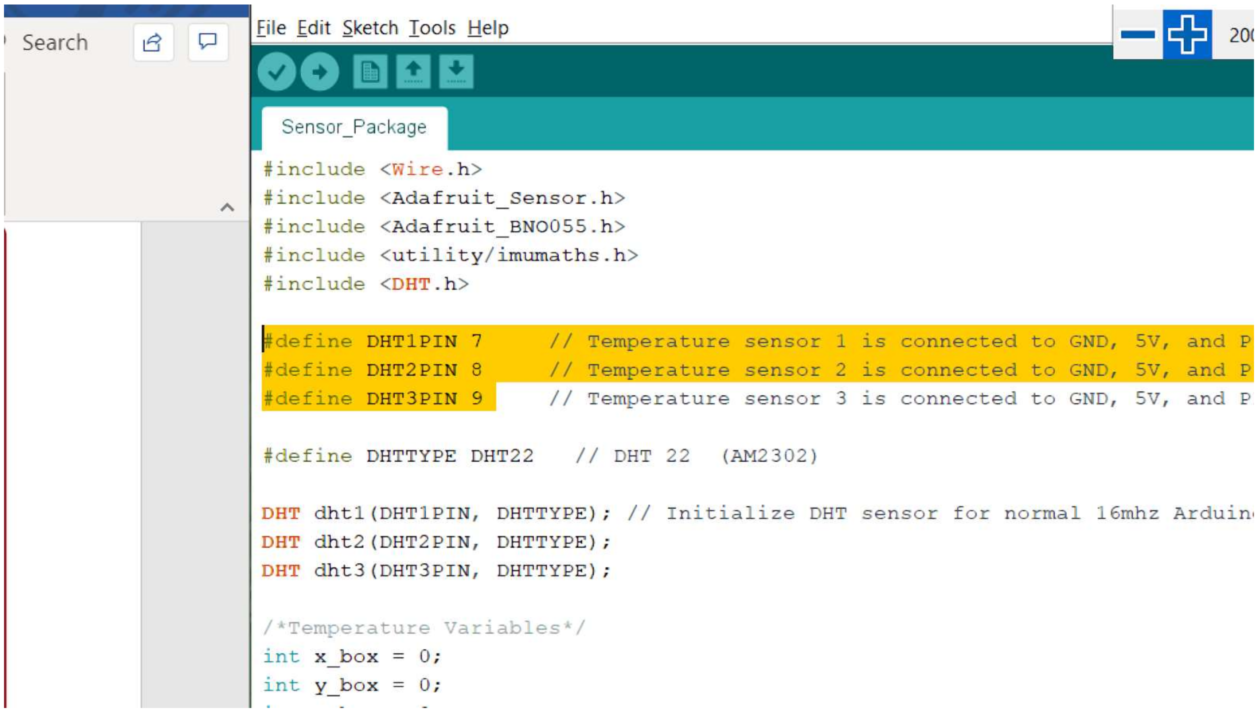
   ii. PWM 3 is assigned for the Hall-Effect Sensor

   iii. I2C Pins SCL and SDA are REQUIRED for the BNO055 IMU (do NOT change)

   iv. 5 V is the five volt output pin of the board.

   v. Ground (GND) is the ground for all sensor connections

c. If you need to reassign pin numbers, the location in the code is here:

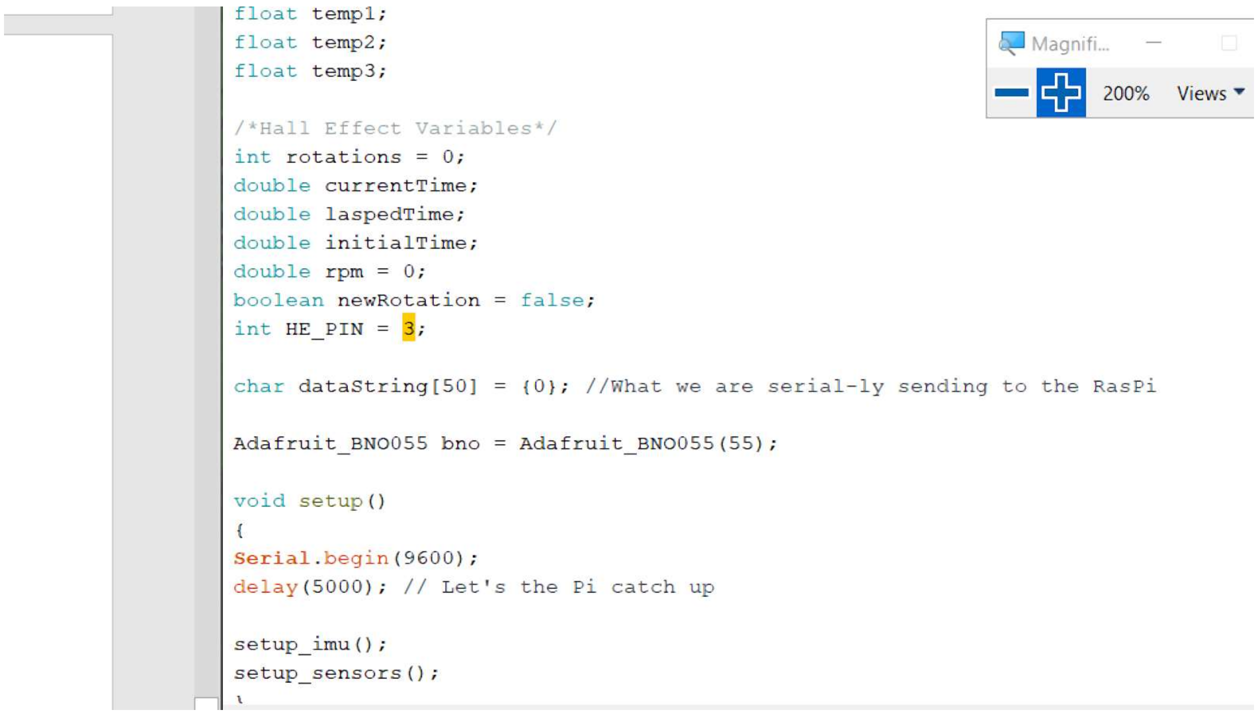


```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <DHT.h>

#define DHT1PIN 7      // Temperature sensor 1 is connected to GND, 5V, and P
#define DHT2PIN 8      // Temperature sensor 2 is connected to GND, 5V, and P
#define DHT3PIN 9      // Temperature sensor 3 is connected to GND, 5V, and P

#define DHTTYPE DHT22   // DHT 22  (AM2302)

DHT dht1(DHT1PIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduin
DHT dht2(DHT2PIN, DHTTYPE);
DHT dht3(DHT3PIN, DHTTYPE);

/*Temperature Variables*/
int x_box = 0;
int y_box = 0;
```

Figure 16: Pin for Thermal Sensors Declaration Location

To remap the Temperature sensors, change ONLY the integers of 7, 8, and 9 respectively to another integer on the PWM pins that you have open.

To reassign the Hall-Effect sensor, the location exists here:

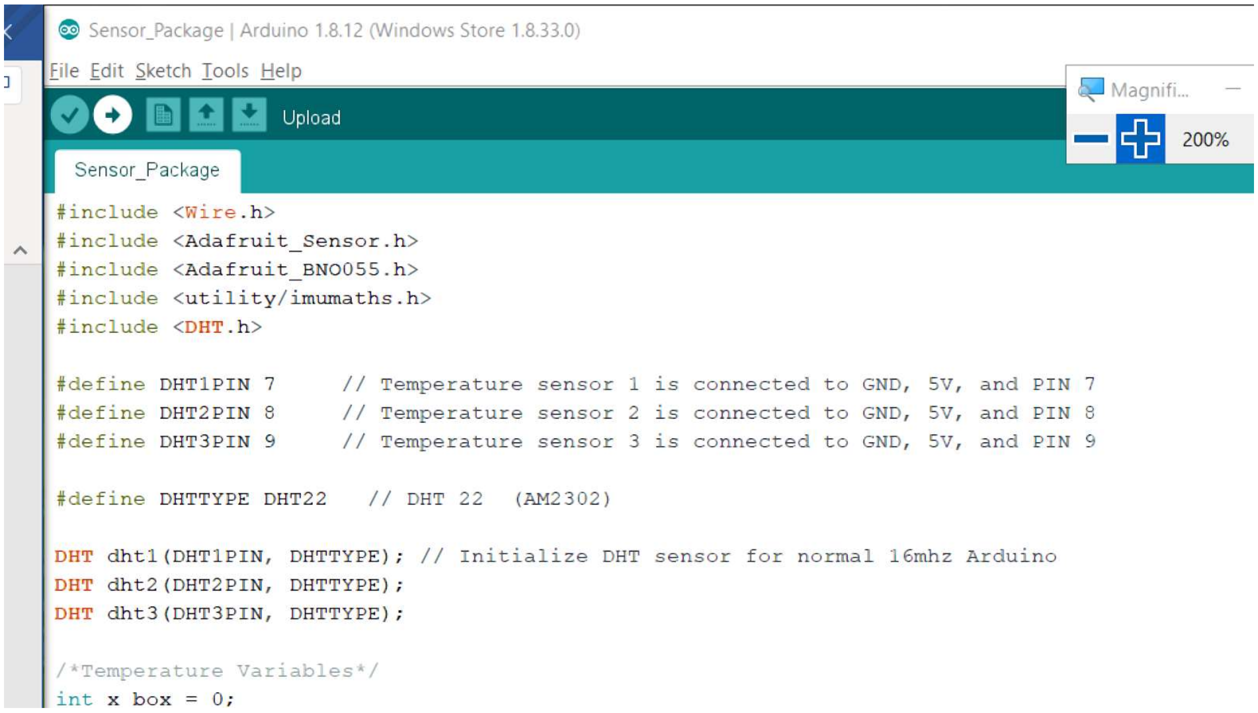


```
float temp1;
float temp2;
float temp3;

/*Hall Effect Variables*/
int rotations = 0;
double currentTime;
double laspedTime;
double initialTime;
double rpm = 0;
boolean newRotation = false;
int HE_PIN = 3;

char dataString[50] = {0}; //What we are serial-ly sending to the RasPi

Adafruit_BNO055 bno = Adafruit_BNO055(55);

void setup()
{
Serial.begin(9600);
delay(5000); // Let's the Pi catch up

setup_imu();
setup_sensors();
```

Figure 17: Pin Declaration for Hall-Effect Location

Again, to remap this pin, simply change the integer “3” to any of your available PWM pin numbers.

6.) Click UPLOAD


Sensor_Package | Arduino 1.8.12 (Windows Store 1.8.33.0)
File Edit Sketch Tools Help
Upload
Sensor_Package
Magnifi...
200%

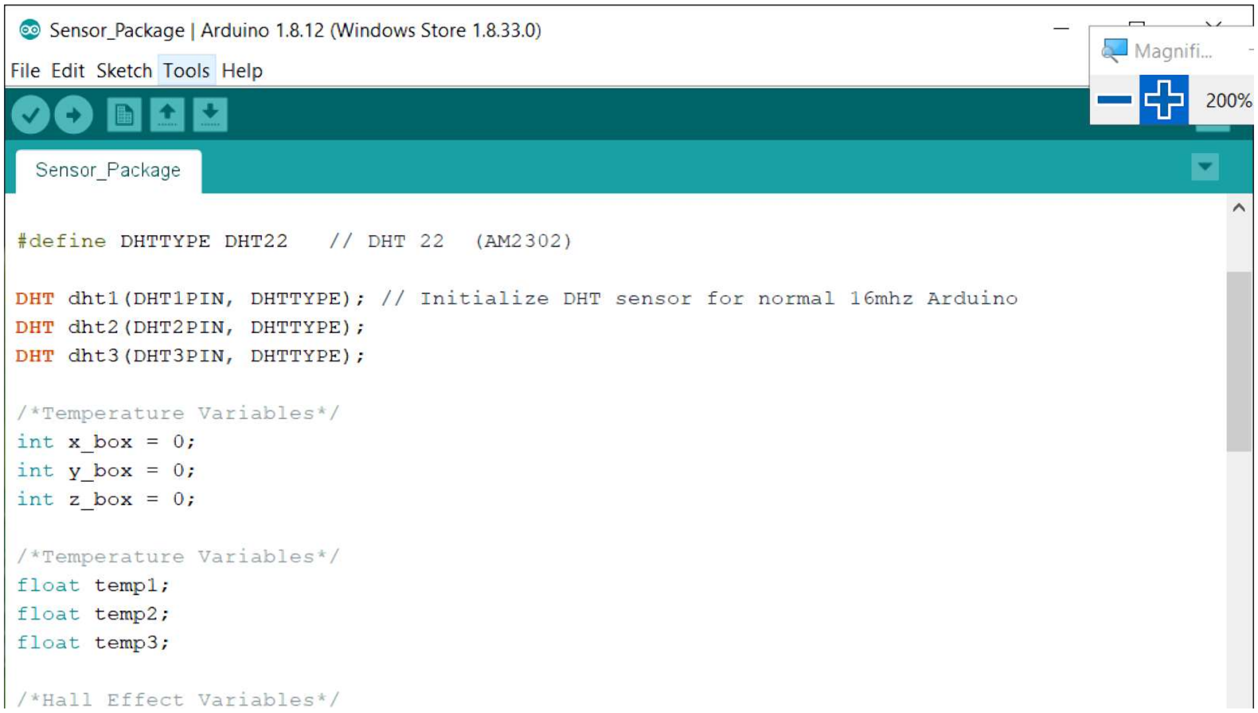
```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <DHT.h>

#define DHT1PIN 7     // Temperature sensor 1 is connected to GND, 5V, and PIN 7
#define DHT2PIN 8     // Temperature sensor 2 is connected to GND, 5V, and PIN 8
#define DHT3PIN 9     // Temperature sensor 3 is connected to GND, 5V, and PIN 9

#define DHTTYPE DHT22   // DHT 22  (AM2302)

DHT dht1(DHT1PIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduino
DHT dht2(DHT2PIN, DHTTYPE);
DHT dht3(DHT3PIN, DHTTYPE);

/*Temperature Variables*/
int x box = 0;
```

Figure 18: Uploading Code to Arduino

a. Arduino will keep the code stored locally, and when powered on it will run the code once that was last uploaded.. We can view the data that eventually will be sent to the Raspberry Pi via the USB-A to USB-B cable. For now, power the Arduino from a computer USB so that the serial output can be read from the users computer. Once the user wants to send the data from the Arduino to the Raspberry Pi, simply take the USB cable from the user's computer and insert the USB cable into any of the Raspberry Pi's USB ports.

7.) Open Serial Monitor

To open the Serial Monitor and to see what is being said, first click on "Tools"


Sensor_Package | Arduino 1.8.12 (Windows Store 1.8.33.0)
File Edit Sketch Tools Help
Sensor_Package
Magnifi...
200%

```
#define DHTTYPE DHT22   // DHT 22  (AM2302)

DHT dht1(DHT1PIN, DHTTYPE); // Initialize DHT sensor for normal 16mhz Arduino
DHT dht2(DHT2PIN, DHTTYPE);
DHT dht3(DHT3PIN, DHTTYPE);

/*Temperature Variables*/
int x_box = 0;
int y_box = 0;
int z_box = 0;

/*Temperature Variables*/
float temp1;
float temp2;
float temp3;

/*Hall Effect Variables*/
```

Figure 19: Using Tools Option

Then click “Serial Monitor”

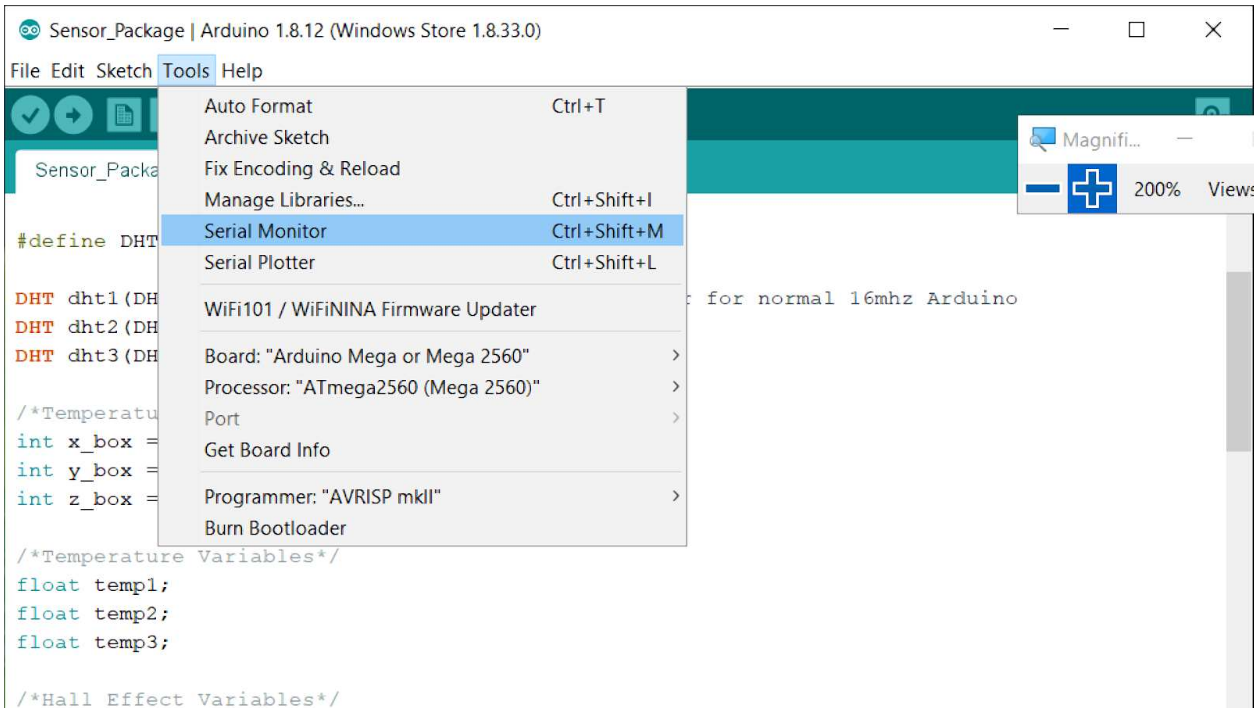
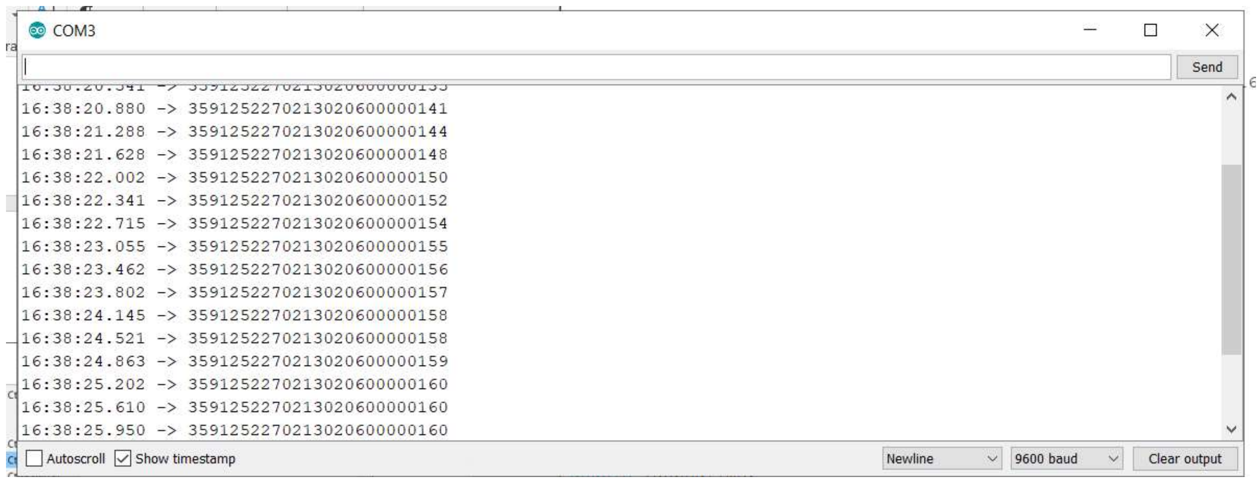

Sensor_Package | Arduino 1.8.12 (Windows Store 1.8.33.0)
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Magnifi...
200%
Views

Figure 20: Accessing the Serial Monitor

And then the Serial Monitor will open on your screen



```
16:38:20.880 -> 3591252270213020600000141
16:38:21.288 -> 3591252270213020600000144
16:38:21.628 -> 3591252270213020600000148
16:38:22.002 -> 3591252270213020600000150
16:38:22.341 -> 3591252270213020600000152
16:38:22.715 -> 3591252270213020600000154
16:38:23.055 -> 3591252270213020600000155
16:38:23.462 -> 3591252270213020600000156
16:38:23.802 -> 3591252270213020600000157
16:38:24.145 -> 3591252270213020600000158
16:38:24.521 -> 3591252270213020600000158
16:38:24.863 -> 3591252270213020600000159
16:38:25.202 -> 3591252270213020600000160
16:38:25.610 -> 3591252270213020600000160
16:38:25.950 -> 3591252270213020600000160
```

Figure 21: Sample Data Stream in Serial Monitor

Make sure that your baud rate in the bottom right corner is set to "9600" or else the serial monitor will produce unintelligible gibberish.

Alternatively, you can press SHIFT + CTRL + M to open the "Serial Monitor" directly from the working environment.

8.) See results

   a. What we see is just a string of integers, and each set of those corresponds to one of the sensors and the raw data that they will be transmitting. The first 9 digits are the coordinates for x, y, and z as gathered from the IMU, the next 9 are the degrees in Celsius as detected by the temperature sensors (00.0° format for temperatures <100°C), and the last is the rotation count as detected by the hall-effect sensor.
   b. This data is then sent over to the Raspberry Pi for data processing

Raspberry Pi Setup:

1) First, the Raspberry Pi should be set up with Raspbian and have internet capabilities. To do this, follow the instructions from the user guide for the self-driving module.
2) From this step and onward, the steps should be completed by using AnyDesk from the user's computer and accessing the RasPi remotely, rather than working directly with the Raspberry Pi itself.
3) Download the python script from Github.
4) Create a new folder on the Raspberry Pi and place the python file that was downloaded from the Github into that new folder that you just created
5) Create a text document, using Text Editor, and name it “data.csv” and place it in the same folder as the python script. This file will be where all of the data that is sensed and sent over by the Arduino gets saved to.
6) Open the terminal command window on the Raspberry Pi and locate the folder where the python and csv files are located.
7) Raspbian comes by default with the program Python3 built in, and this will be used to store the data. Simply ensure the USB cord from the Arduino is being powered by a USB port on the RasPi, and then in the newly opened terminal window type “python3 RPI_Store_Data.py” without the quotations into the character display line and press the enter key. Now, the data is being collected from any attached sensors. The sensors will sense the data, report that data to the Arduino, which then sends it over to the Raspberry Pi via serial communication along the USB cable, where it will then be stored and saved in the data.csv file that you have created.
8) To end data collection, press “CTRL” + “C” to end the collection process.

10.3 IMU Bump Test Data

X	Y	Z	Temp1	Temp2	Temp3	RPM
172	198	170	27	24	29	148
172	198	170	27	24	29	148
172	198	170	27	24	29	153
172	198	170	27	24	29	153
0	180	180	0	24	29	0
0	180	180	0	24	29	0
34	222	216	0	24	29	0
31	191	188	0	24	29	0
33	198	168	0	24	29	0
296	199	169	0	24	29	32
296	200	169	0	24	29	32
296	200	169	0	24	29	32
294	201	169	0	24	29	32
281	200	169	0	24	29	32
264	200	168	0	24	29	149
245	200	168	0	24	29	149
250	197	167	0	24	29	149
224	196	169	0	24	29	149
221	200	168	0	24	29	149
253	205	165	0	24	29	154
258	199	169	0	24	29	154
262	200	168	0	24	29	154
256	201	168	0	24	29	154

250	201	169	0	24	29	154
236	203	169	0	24	29	153
236	201	170	0	24	29	153
234	202	169	0	24	29	153
232	205	169	0	24	29	153
225	205	169	0	24	29	153
230	206	169	0	24	29	153
231	206	170	0	24	29	153
235	206	170	0	24	29	153
242	205	171	0	24	29	153
226	206	170	0	24	29	153
224	206	170	0	24	28	154
226	206	169	0	24	28	154
235	205	169	0	24	28	154
236	206	169	0	24	28	154
230	207	169	0	24	29	154
225	206	169	0	24	29	149
221	206	169	0	24	29	149
224	206	169	0	24	29	149
235	192	170	0	24	29	149
226	200	164	0	24	28	149
221	200	164	0	24	28	149
222	200	164	0	24	28	149
222	200	164	0	24	28	149
220	202	168	0	24	28	149
214	204	171	0	24	28	149

220	191	170	0	24	28	149
214	201	170	0	24	28	149
211	201	170	0	24	28	149
222	202	170	0	24	28	149
219	201	168	0	24	28	149
218	201	169	0	24	28	149
247	193	164	0	24	28	149
277	163	132	0	24	28	149
272	209	173	0	24	28	149
272	189	174	0	24	28	149
276	194	171	0	24	28	148
276	194	171	0	24	28	148
276	193	172	0	24	28	148
276	193	172	0	24	28	148
276	193	172	0	24	28	148
276	193	172	0	24	28	149

10.4 Steering Linkage Testing Platform Arduino Code

```
#include <Servo.h>

int servoPin = 5;
int count = 0;
Servo myservo;
int pos1 = 30;
int pos2 = 150;


void setup() {
  Serial.begin(9600);
  myservo.attach(servoPin);
  while (count < 250){
    myservo.write(pos1); //turn right
    delay(1000);
    myservo.write(pos2); //turn left
    delay(1000);
    count++;
    Serial.println(count);
    }
  Serial.print("done");
}


void loop(){
  delay(1000);
}
```

10.5 Arduino Sensor Reading Code

Link to Github

Code:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <dht.h>

dht DHT_1;
dht DHT_2;
dht DHT_3;

#define DHT11_PIN_1 7
#define DHT11_PIN_2 8
#define DHT11_PIN_3 9

/*Temperature Variables*/
int x_box = 0;
int y_box = 0;
int z_box = 0;

/*Temperature Variables*/
float temp1;
float temp2;
float temp3;
```

```
/*Hall Effect Variables*/

int HE_PIN = 3;
unsigned long timeold = 0;
volatile byte rotations = 0;
unsigned int rpmilli = 0;
unsigned int rpm;
float speed;

char dataString[50] = {0}; //What we are serial-ly sending to the RasPi

Adafruit_BNO055 bno = Adafruit_BNO055(55);



void setup()
{
  Serial.begin(9600);
  delay(5000); // Let's the Pi catch up

  setup_imu();
  setup_sensors();
}

void loop()
{
  loop_imu();
  loop_sensor();
  countRotations();
  delay(250);
}

/*****************************************************************************************/
void setup_imu() {
  //Serial.println("Setting Up IMU");
  //Serial.println("");

  /* Initialise the sensor */
  if (!bno.begin()) {
    Serial.print("Ooops, no IMU detected ... Check your wiring or I2C ADDR!");
    while (1);
  }
  delay(1000);
  bno.setExtCrystalUse(true);
}

/**************************************************************************************/
void setup_sensors() {
  //Serial.println("Setting Up Hall-Effect Sensor");
  //Serial.println("");

  /*HE Sensor Init*/
  pinMode(HE_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(HE_PIN), countRotations, FALLING);
}
```

```
/****************************************************************
*********************/
void loop_sensor() {
  int chk0 = DHT_1.read11(DHT11_PIN_1);
  int chk1 = DHT_2.read11(DHT11_PIN_2);
  int chk2 = DHT_3.read11(DHT11_PIN_3);

  if (rotations >= 5) { //increase rpmCount for better RPM resolution, decrease for faster
update
    rpm = 60.0 * 1000.0 * rotations / (millis() - timeold); //60*1000 is the factor to convert
ms to minutes
    timeold = millis();
    rotations = 0;
    //Serial.println(rpm, DEC);
  }


sprintf(dataString, "%03d%03d%03d%04d%04d%04d%04d", (int)(x_box), (int)(y_box),
(int)(z_box), (int)(DHT_1.temperature * 10), (int)(DHT_2.temperature * 10),
(int)(DHT_3.temperature * 10), (int)(rpm)) ;
//When adding more variables, make sure to leave RPM at the end so as to make it
"infinitely expand" incase RPM get to high
Serial.println(dataString);
}


/****************************************************************
***************/
```

```
void countRotations() {
  rotations++;
}

/*****************************************************************
*****************/
void loop_imu(void)
{
  /* Get a new sensor event */
  sensors_event_t event;
  bno.getEvent(&event);

  x_box = event.orientation.x;
  y_box = event.orientation.y;
  z_box = event.orientation.z;

  x_box += 180;
  y_box += 180;
  z_box += 180;

  delay(100);
}
```

10.6 Raspberry Pi Data Collection Code

Link to Github

Code:

```
import serial
import csv

ser = serial.Serial(port='/dev/ttyACM0',
                    baudrate=9600,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,
                    bytesize=serial.EIGHTBITS,
                    timeout=0.5)

with open("data.csv", "w") as new_file:
    csv_writer = csv.writer(new_file)
csv_writer.writerow(["x", "y", "z", "temp1", "temp2", "temp3", "rpm"])

while True:
    if (ser.in_waiting > 0):
        line = ser.readline().strip().decode('utf-8')
        x = int(line[0:3])
        y = int(line[3:6])
        z = int(line[6:9])
        temp1 = int(line[9:13]) / 10
        temp2 = int(line[13:17]) / 10
        temp3 = int(line[17:21]) / 10
        rpm = int(line[21:25])
```

```
# print("T1=" + temp1 + ", T2=" + temp2 + ", T3=" + temp3 + ", rpm=" + rpm)
#updated in pycharm cause RPi editor is dog
csv_writer.writerow([x, y, z, temp1, temp2, temp3, rpm])
```

10.7 Raw Temperature Test Data

10.7.1 Temperature Test 1

T1	T2	T3
20.99274534	19.50735859	19.99593948
20.52159956	19.85341927	21.56226443
19.63265783	21.99953657	19.6469824
21.00992494	20.36394557	20.27725049
21.25592402	21.51283263	19.84796671
21.92545792	21.46798502	21.21487238
19.21830626	21.77921843	20.64019809
21.47153157	21.65589843	21.18949265
21.66450576	20.28432874	19.7042289
20.09920488	21.23626819	21.93662301
19.26418121	20.37212331	19.88302778
20.71370862	19.2078289	19.61048975
19.51070217	21.56959001	20.89715237
21.59869792	19.36793171	21.17607682
20.58409151	19.50543515	19.31661331
19.95682851	19.70252272	21.06557122
20.12770462	20.4697221	19.44068048
19.14569353	19.19619147	21.85540781
19.70422177	20.37355774	21.58766058
19.0181597	20.6455625	21.88183589
21.69123024	21.74712036	20.34429371
20.68041598	20.95964116	21.12178235
20.97641653	19.7291207	21.26711799
20.27503834	19.90681241	21.43408164
21.30916449	20.9103778	21.57979693
20.03612198	21.20576188	19.88679672
20.86678408	19.19836931	19.05049196
21.69573177	20.65801767	20.03293425
19.5729147	19.75882093	20.95700644
20.8328615	19.35008116	19.17353507

21.5517287	21.89567443	19.11537084
21.82571806	19.49585324	20.03870269
20.66889343	20.67381359	20.13615097
19.71399893	20.6935031	21.48905064
21.26895312	19.03201871	21.42402591
20.40910045	20.19894082	19.49835409
20.60460657	20.32971124	21.00773064
21.85667879	21.50013522	19.41956785
21.21765049	19.11368006	20.90251973
20.89811688	21.25777933	19.5981199
19.09112298	20.66433878	19.7047554
19.77628501	20.56899111	21.40644228
20.18003491	21.5099517	19.10255087
20.45819639	21.03821581	21.50061064
19.14226071	19.89711023	20.57771764
21.81506132	19.09525495	21.20291004
20.4455417	21.63260577	20.15832361
20.21175796	19.1920121	20.69293587
20.50062686	19.11239679	20.55933018
21.35642773	19.5103463	20.83489863
20.05945403	20.71492238	20.58803157
20.38093731	21.71224347	21.81119036
21.58170323	19.43649607	20.47399214
19.73914354	19.41483023	19.99366456
20.88939677	19.20447691	21.75810271
21.68767197	20.84669406	19.02678745
19.73947659	19.12854395	20.00612463
21.05933701	20.6742713	19.67376441
20.50606483	21.09149138	19.62815578
19.84906461	20.92909399	19.48148619
19.30066409	19.97895553	19.23194757
19.54353676	19.23543434	20.53685902
21.40873392	20.08342652	19.72567196
19.2666887	21.0659737	19.9405755

21.98939075	21.26347129	20.84094928
21.95220806	21.23879201	20.93009214
19.69153265	20.40660574	20.16169392
19.05049161	21.52608834	21.79754851
20.0702317	19.13662919	20.42622932
19.5309154	21.55309965	21.71727676
20.84313213	20.16703087	21.89241472
21.34119315	19.61979782	20.80871349
21.77483312	19.76547412	20.5070751
20.36509735	19.93885744	19.09186571
19.00044099	21.63516648	19.35632652
21.49956907	19.82746521	20.46255841
21.34588458	21.70255803	21.26256326
20.61455977	21.65978036	20.33267396
20.63607257	20.53333593	19.07265344
19.11675207	19.93886719	20.69630178
20.33201706	21.97677773	21.30165498
20.00515866	20.50495431	20.03656284
21.34978076	21.39949068	20.774096
19.96667428	19.99978988	19.76092915
19.59825702	21.14133848	20.17318517
21.13776164	21.85915859	21.71678924
21.15574032	20.37748101	19.16547192
21.97001756	21.92741222	21.40613334
19.87897183	21.06012328	21.72832514
20.63844499	21.66489327	20.79146331
20.4104786	19.34157307	21.44963959
20.96853063	20.75276049	21.20794245
21.59379512	20.53699167	21.88852739
20.91899883	20.85694979	21.9035523
21.31044278	21.10182549	21.34314775
20.61931351	19.57644455	20.79443699
21.07182716	20.9970218	21.84364139
19.34318226	21.11892311	19.21210915

20.70361966	21.55705466	19.41635903
20.01066279	21.48306926	19.9388258
21.28545877	21.31445355	19.20781908
21.90178504	20.61086403	19.77389704
19.80630105	21.06348581	19.23634841
21.67431176	20.85014126	20.98426523
20.68247204	21.54908182	19.19500748
21.08842193	19.64832805	19.39532299
21.2940215	20.96980426	19.96568378
20.52339484	20.70897344	20.84117669
20.72362679	19.80109127	21.44125667
19.53810134	19.17537945	19.37658641
20.1448705	20.23318783	20.77625015
21.88120336	21.06801717	20.85543144
19.28274478	21.62381668	20.69775107
20.14197786	21.26766603	20.39319033
19.55381541	19.27601873	19.6745833
19.18279692	19.39323345	20.18464051
21.14736055	19.44465347	20.24639972
21.78431544	21.82648349	19.36445446
21.20834676	21.76934151	19.10373367
19.12977218	20.75554261	19.93990489
19.47030968	19.2363385	21.54900428
19.56155448	20.08080273	21.54441316
20.21869119	20.83674865	19.98648571
19.05798692	21.50207368	19.25174773
19.16618297	19.25405913	21.33144959
19.33849476	19.50153935	20.237472
21.03001356	21.26124452	20.70581374
20.0387764	20.68586156	19.60156732
19.92137718	20.0083444	20.02604435
21.77297154	19.25089849	21.48943956
20.70946204	19.64775546	20.8807577
21.77619332	21.89782275	20.22381766

19.69252882	20.7794326	19.24806668
20.49802033	19.33943256	20.83404277
20.09758834	20.14531938	21.68945943
19.6860442	20.58526761	20.1292395
20.30542324	21.8958014	20.7667512
19.98939331	20.24450446	21.68577718
19.44160502	19.26560675	20.91590346
19.64698571	20.41288887	21.32583352
21.03502249	21.69779028	21.24638216
21.76640665	20.85615476	21.26193499
20.28712217	19.5482927	21.85296032
19.79969047	21.72509349	19.12188229
19.84706502	19.58673625	21.87983785
20.97546612	20.25992306	21.98989558
19.59543443	21.31891491	20.68163762
21.43707874	20.70272737	19.98419683
21.31274002	21.8207854	19.91486118
20.08823458	21.04098118	20.21025645
19.27511006	20.56122119	19.96499521
20.47686001	19.37266986	20.29488002
20.07024797	21.760537	20.98251017
19.410273	20.49999052	20.62212431
19.56956299	20.98362692	21.44498906
20.2371163	21.22545559	21.50126572
21.10061083	20.09091706	19.07767309
20.34110345	21.40839687	19.33387729
19.0264679	21.6982704	21.41156025
20.24640405	19.824011	20.58631098
19.00990153	21.15017134	19.01321165
21.85300967	21.69943693	19.58095301
21.09030892	21.5512698	20.52559201
19.61184196	21.44577256	21.50058447
21.96611635	20.88261009	20.17222532
19.4635081	19.79092902	21.15170845

19.70603256	20.04415448	21.12091182
21.04189572	21.49088239	19.19211241
20.92621926	20.09872896	19.41711057
21.06528941	20.06697993	19.32313944
20.69970298	20.96905962	20.68531992
21.97506142	19.11083871	21.14671041
21.30744075	20.67457166	21.07951727
21.3475478	21.4080418	19.6502412
19.11124104	20.34414038	21.42668087
20.40139197	20.26490647	20.90904566
20.30671971	20.05169426	21.05198469
20.19398247	20.55230897	20.27603177
20.09475368	19.98077905	20.56741923
21.88080451	20.1957726	20.89354379
19.65323082	21.9098309	20.12996999
20.37537278	20.49624868	19.53375389
20.47364509	21.04052556	19.27931488
21.96034627	19.13136412	21.91436775
21.69124439	19.17635466	21.9121599
21.06849631	19.09811516	19.49851185
19.54062545	21.24733042	19.97029879
20.67209136	20.86642018	20.26506787
21.33412121	19.6098681	21.75738832
21.22827494	21.53559952	21.81259292
19.22691751	19.00949307	20.96600386
21.0540805	19.75534651	20.86111585
19.60504883	21.04975177	21.51815989
20.61640317	20.28732488	21.55696486
19.33038706	19.5758703	20.49113116
19.67871039	20.55705362	21.92240433
19.18618057	21.05923915	20.02371166
19.34732843	19.85769879	19.17093298
21.08900637	21.50225192	20.24924606
21.99622002	19.51291462	20.91791083

19.63514337	19.10658116	19.56972224
21.15132081	19.5456913	20.43268245
19.01398606	20.13643186	20.8244225
21.90541328	21.01065552	21.02109117
20.69643558	21.62710245	20.98720958
20.1744875	21.17041021	21.30542431
21.5560937	21.54436805	20.58421088
21.34950061	20.18992197	19.132186
20.43503946	21.28844748	21.20983716
19.03695328	20.80990139	21.02710503
19.86910513	20.29494127	21.75070659
21.90193414	20.87944471	21.61248925
20.75062905	19.54205313	21.63677809
20.05304653	21.28440503	20.17613195
21.65601419	19.79719276	20.82505349
21.25535136	19.32115768	19.24574756
21.19305822	21.08587984	20.99746618
20.98119718	20.32135464	19.58130973
21.83655768	20.44153426	20.60051972
21.25623836	19.23330007	20.59322295
21.89336781	21.18667229	19.86499403
19.41636937	21.07670994	21.86117376
20.5587908	20.63529428	19.58033885
20.63866075	21.89246528	20.84157258
20.84935714	19.91451609	19.43659546
20.96422054	19.95576891	19.07615027
21.89287771	21.9430795	20.72881301
20.90240778	21.07373853	20.45656221
20.76081722	19.22819656	19.70258224
19.64379389	20.47641022	20.45646368
19.39194349	20.94379908	19.12773092
19.85413271	20.8784939	19.45075137
21.09683119	20.51685934	21.65928142
21.55441976	20.2487703	21.42945478

21.58276093	19.09881472	19.28495709
20.18456849	20.70110349	21.84965157
21.09609813	19.65171435	20.78695307
21.71261811	20.00766539	21.2119657
20.35822699	19.88954926	19.54810298
21.09063743	21.83873787	19.99847967
21.951045	19.83101585	19.61034387
19.1161613	19.02762333	20.05668814
19.18401329	19.72800133	21.31349739
20.36826691	20.53262001	19.69049248
20.50754301	20.51229582	20.78746564
21.52507415	20.17519308	19.84586617
21.00345379	21.66998188	21.48922571
19.59996477	19.54452162	21.71800803
20.42435171	21.51781783	19.83136378
20.16321447	21.30925756	20.11148475
19.72766312	21.76850544	20.97081426
19.04070136	20.71897292	21.34858565
20.87996694	19.97246135	21.26007375
21.53919613	21.727377	19.7280687
19.23205933	21.99363679	20.7191892
20.5663853	21.35958507	20.4114533
20.58935523	20.08773987	20.69164095
21.86178117	21.87474644	21.47245841
21.84589283	21.01520912	20.39284558
21.59121469	20.6570642	19.14313898
20.0991101	20.94139638	19.15400998
19.08248847	21.58215602	19.37442039
21.41806473	20.79619929	20.37971509
21.18342576	20.84692689	20.68637957
19.06687205	20.31901352	20.66376904
19.43702918	21.68039538	20.29269503
19.17044679	20.87716058	19.49902769
21.71345418	20.2466054	21.16274765

21.49940783	21.38231751	20.94316492
19.47730073	20.95085047	20.44307278
19.70909389	20.44598334	21.59951505
21.25426558	20.82456082	21.54705437
20.72354884	21.40834657	19.991483
19.98703491	20.0155948	19.67833498
21.75417184	21.78236761	21.76761045
20.6206055	21.48820253	20.16045307
20.1952448	19.10439719	21.10481748
20.85681501	20.94508108	20.17438667
19.37110089	20.44276245	19.8778519
21.38215937	19.84284215	20.46928638
20.11429175	19.04565436	20.847436
21.95804248	21.06759506	19.81265064
19.73834596	19.60914171	20.76750339
20.41510643	19.77058827	21.02808586
19.20584902	21.60565251	19.41837227
20.30055276	20.70193801	21.96578476
20.77146447	20.36385351	20.39845964
19.56729357	20.80602034	21.8377508
19.95466456	19.35420797	19.82856156
19.28494453	19.30335791	21.59202628
19.23845191	19.27289879	19.98180875
20.80424693	19.94465837	20.10380023
21.03398243	19.28320455	20.30676067
19.66830667	21.81255522	20.52295969
20.34810492	19.56136048	20.1839785
20.77648152	19.21067362	21.37470889
20.65841075	20.38760631	20.38667765
20.95089178	21.8286431	21.59024572

10.7.2 Temperature Test 2

T1	T2	T3

35.0516134	34.92887938	34.2005882
34.6839868	34.96949351	36.44912626
36.90680199	35.98717032	34.73464309
35.92072517	35.75965331	35.03384065
35.33010322	36.55603443	36.06760782
35.86808045	34.50519751	35.8141226
36.49066401	34.62883162	35.08424089
36.9217883	34.87153752	34.46870766
36.91430818	36.48953521	35.12056338
36.63024536	34.38855579	35.97613235
34.58250728	35.81164541	36.4571749
36.69191654	35.1484406	35.12494961
34.48239173	35.18626299	34.17954145
34.83925137	34.20701471	34.23867895
34.08214809	34.70993455	36.48786571
36.26574293	35.11459956	34.02208318
34.29601813	34.76764028	35.71636135
35.52210485	36.80509985	36.46831616
36.32864276	36.17625728	36.17700589
36.05229224	34.32317564	36.17816659
34.91120126	34.86896595	36.27203372
34.82957508	34.34938005	35.66173467
34.37633282	35.83945313	34.12158321
36.28456103	35.42536415	36.22625975
34.98296342	35.49261876	35.18017255
35.36653404	34.62222096	34.23643192
36.56680311	36.81014789	36.62332088
35.45044334	36.52440114	35.27845023
36.83143708	36.10722297	34.78205978
36.77617916	36.91215631	36.30754806
35.2357722	34.98141933	34.89980857
35.39436286	34.96305525	36.03002831
35.00637922	36.51514709	34.24777256
35.48185515	34.90679111	34.71249731

36.29139621	34.6457229	36.12943408
35.49674711	35.55940717	36.54854455
34.06265227	34.9472279	34.95274463
36.66109075	35.43848711	35.70029287
34.2041772	35.03882629	35.9743099
35.71250133	34.07817747	35.80088044
34.14106329	36.43638546	34.10034463
34.85361454	36.78366679	36.01399166
35.65862946	34.46739921	36.54472826
34.27991541	35.40357704	34.73611167
34.73703773	36.7708202	34.92851367
34.08851166	34.47340797	36.56245482
36.73306659	35.8790539	35.49426037
35.59795686	35.46461248	34.66284274
34.23394059	35.23275362	36.39221714
34.05533292	35.23608646	35.0016459
34.99633365	34.09202955	35.06645388
36.11268133	35.45499263	36.73683817
36.37844427	34.8902007	34.67342503
36.63986473	36.76216215	35.32596029
34.33902119	35.41636292	34.45216793
35.05251073	35.28724092	35.60817611
34.88217635	36.42703252	36.23654776
35.09741802	35.45678704	35.0066048
35.69165526	35.06928575	34.76092088
34.50740527	35.52433044	36.66679718
34.74696935	34.37061413	34.41971375
35.24489231	35.22775184	35.30681748
36.0235727	36.61473756	36.26021194
34.65847905	36.24212004	35.87122052
35.73036834	34.10657287	36.73098638
34.31751916	35.64437282	35.15274327
36.16364891	34.64028164	34.29361362
35.93238109	34.39207879	34.84734681

35.59363835	34.25677905	35.10358043
34.5570882	35.49725673	34.34405298
36.03775731	36.02435909	36.4515188
36.47878757	34.256249	35.80266755
34.40635079	34.83939171	36.35534627
34.17560697	35.88946698	35.94991751
36.87780093	36.44271689	36.13776023
35.64962535	35.51510861	35.81584671
36.2197128	35.1288467	35.99525386
34.07174464	36.49645124	35.45372282
35.82336679	36.21499042	34.65090838
35.0951691	36.28395272	36.2325413
36.86513949	36.28893858	34.32331675
34.56705777	35.54517627	36.58393258
36.10951519	34.20082342	35.28088813
34.68492858	35.21462859	36.22547364
34.48515603	34.66366942	34.48036047
36.4748948	35.38225044	36.79878198
34.71292706	35.21955395	35.70964116
35.71026507	35.40691557	34.06016886
34.66055068	35.0783124	36.72024687
34.01702471	36.22640226	36.07900742
34.91359262	34.38839478	35.1988966
34.73095162	35.42755527	35.24741724
36.66078794	34.62716129	36.69963075
34.46143294	36.83251799	36.02559042
36.7204367	36.85118382	34.64869876
34.98941161	35.19489632	35.55942186
36.41402468	35.2047101	35.00903257
35.86937716	35.33252769	36.25851883
34.50673585	34.60318083	36.61717714
35.53414767	36.85333029	34.83873778
36.26866006	35.47353042	34.50397476
36.01069294	35.46276492	36.7678399

34.50489504	36.50703248	34.80512788
36.0678926	36.03415548	35.63751044
34.5001849	35.05488846	34.23613814
34.33348853	34.73801136	36.17980544
34.0201096	34.14092797	35.32831453
36.0662909	35.07821601	36.98916639
35.80518182	34.47716555	36.93195708
35.85776624	36.88927293	36.86868066
34.33865308	36.82211523	35.32597979
34.89943959	34.88062622	35.03561043
34.71866451	34.82242085	35.01978268
34.00215526	36.04542182	35.46881306
34.32898877	34.00734681	36.08589723
35.53139341	34.00909926	36.8101298
36.99152968	35.07295767	35.16670323
36.64884165	36.70817145	35.62183051
36.66261658	34.28647157	35.9043763
35.87176096	36.64486041	36.20960563
34.47173088	34.8775287	36.15232412
34.2039267	36.54544118	34.73036208
36.41652529	35.35628851	36.34397163
34.00879202	34.69883738	35.0792801
36.94587037	36.83083774	35.67926833
35.01884613	36.41000071	35.08244667
34.56033398	35.68892619	35.40223309
35.36488126	34.16578025	34.37922562
36.71723996	35.390271	35.39887065
36.30312835	34.77837052	36.95981985
35.47866439	34.09241041	36.36166774
36.28318494	36.39517401	34.60381143
35.42192897	36.24031221	36.01549954
35.57967056	34.99929236	35.24854254
36.23299945	35.0007303	34.75417256
35.67807142	34.48026817	35.98386124

34.71645154	36.25787113	36.82415158
36.81085907	34.34166756	36.63195674
36.72724253	34.1266131	36.94253127
36.16240108	36.3952597	36.51584407
35.0084339	36.23077711	34.43342334
36.5572587	34.83869779	35.66372971
34.30740319	35.06236428	34.29715632
35.50115165	35.84562456	35.53333838
35.34623614	34.13696363	34.72967893
35.92024364	35.33804919	34.89276934
35.59829753	36.52252962	34.82809017
34.33368219	34.03780577	35.36315848
34.93396931	36.10339254	36.34818894
36.56584343	34.98719638	35.74206718
34.55885923	34.18678368	34.42036958
36.85594852	34.72192594	35.59193144
36.49520592	35.13154126	35.0332307
34.82922183	36.60868828	36.03005107
34.82262227	34.52234787	35.51518906
35.55925967	35.91490372	36.60855872
34.87176636	34.30029797	36.29159892
34.75760955	34.05191468	35.86579866
35.3811428	36.54238888	34.17988809
34.18368645	34.67131392	36.38389807
36.8132997	36.4373845	34.67268686
34.96415562	36.64603753	35.14849163
36.56559329	36.19024821	35.01662157
34.76200673	36.13554217	36.90114816
35.88896526	35.18695052	35.84325919
36.33606523	35.32308734	36.05675927
36.08546825	34.9290004	35.27265884
36.00409632	35.88124693	36.25881587
36.83435398	35.99375366	36.3662501
34.96753536	36.83081086	34.72508628

35.59330594	35.16173683	36.8345927
35.30194768	36.22285081	34.23279794
35.43487813	36.77336706	35.74037779
34.17140328	35.71169175	35.77223289
35.86053866	35.31012067	35.51432647
35.32202077	35.7687453	34.00500259
36.24119086	36.94312586	35.81108177
34.06306624	36.38765352	35.47063664
34.17204624	36.73715615	36.16654157
35.49440565	34.97710434	36.84308936
35.01278483	34.05005087	34.12324591
34.00613331	34.759287	36.1260419
36.97569845	34.21476394	35.71543568
34.70729893	35.8726613	36.74930125
35.79275466	35.44975755	36.15238493
36.16141641	36.20403333	34.67312396
34.15813202	34.3076542	36.11285635
34.1705692	36.38651689	34.48601193
34.92979302	34.36438543	36.62343049
35.48926623	34.73958868	34.58564924
36.55922039	34.87213481	36.60054144
34.72371108	35.99100548	35.93151126
35.18282786	35.17785389	34.30496482
34.81990839	36.40688168	35.94873625
34.48515363	35.85189332	35.40601306
34.8312601	34.41785778	34.42169762
36.26929274	34.01327485	36.75171103
35.0273322	36.62131122	35.9460557
34.03375803	35.69243777	36.16475862
36.13276438	36.27036027	34.59174667
36.84985806	36.4801263	35.45929939
34.91943379	34.34151932	36.07633057
36.00564579	36.99178031	35.93066796
34.69468267	36.26892078	34.24647338

34.98053847	34.97781373	36.79279099
36.19051325	34.41180782	35.13127245
34.04099484	36.45709778	35.61228101
35.04810139	35.70441231	35.95900275
36.50742653	36.09751973	35.8541497
35.67187408	36.34739437	36.34111115
35.39422207	35.84549812	35.05423697
36.68893674	36.41487722	35.4245128
35.72988786	34.47576693	36.16381546
36.16217202	35.15104049	36.12415075
34.43752166	36.3316742	35.77140251
35.96631233	34.24863451	36.07921289
35.91489113	34.92089023	34.64774252
35.32130778	34.47516667	36.56330597
35.56920297	35.66612173	34.53749087
36.1900745	34.53054513	34.13436755
35.90823013	34.75764071	34.94372629
35.47005783	36.575411	34.07534955
34.69397305	34.56474838	36.63057084
36.34993249	34.50814962	36.97107564
36.15773253	34.11541464	34.46955561
34.65228527	35.03966784	36.73762087
34.83976855	36.00409847	35.54470023
34.05648349	35.12060472	34.79366141
35.93913439	34.32212606	35.03048544
34.98747089	36.22098349	35.39222289
34.77945461	36.48606701	35.39000495
36.1884057	36.70580577	34.54793511
36.45713734	36.65851355	35.98473783
35.16845334	34.65326955	35.95925529
35.73566773	34.88187431	35.02285882
35.26037633	36.43979218	36.37522624
36.78224944	34.47669647	34.13319922
35.94266657	35.48563075	35.35840776

34.36329153	35.16722186	34.17340812
36.71279557	36.38749609	35.29399353
34.27834783	36.09206854	35.5531939
34.55480919	35.13419077	34.06225785
34.22651449	35.21894757	36.46951825
35.16716868	35.03510249	35.37796243
34.27648653	35.91914009	35.55617235
35.42478075	35.91963496	35.58489387
36.91613059	35.85814766	36.29430701
35.1900256	35.07175277	36.43313984
35.25730989	35.03971493	35.81536694
34.78686713	36.43427477	36.14600529
34.78770333	34.49563086	34.85876497
35.2635265	36.99203277	36.0657051
35.71751567	36.76580875	34.48192565
34.62455691	34.76427448	34.3433566
34.04242652	35.06271848	34.72154888
35.47242913	35.80064773	34.07867329
34.21515102	34.78999228	34.78674821
34.00001716	34.48076803	36.91558514
35.51997401	34.39380299	35.24487022
34.88486424	34.71689802	35.33737619
34.81410739	35.20236417	35.21020655
35.79815739	36.50792724	34.58253473
36.12634338	34.86672543	36.13848735
34.72839081	35.29903037	34.60220495
34.28001871	36.99613185	34.7931466
36.98235824	36.57663156	34.51186334
36.27965777	35.90293181	36.31314468
36.2430748	34.83724893	34.34170525
34.91785118	34.68280317	34.56002492
34.69233882	34.97922293	34.17843323
36.12310307	35.88151513	35.74038893
34.52986024	34.19913409	36.21001741

36.8858112	36.29326436	35.32874862
35.90428703	35.75391665	34.25701804
36.52663588	34.12964944	36.897529
34.80527252	34.55242462	36.69979836
34.17078111	35.35254291	36.39174493
36.91927388	35.24677492	35.71913346
34.07781188	36.91657137	35.66459025
35.22252006	36.53758283	35.46516415
35.16733134	34.3838284	34.66899983
36.64812255	36.22473522	36.49132129
35.34704194	36.79916052	34.23445155
36.82433112	36.51627073	35.94883833
36.96544089	35.12134728	36.9562643
34.51930018	35.3450222	35.87499159
34.70379646	34.24432117	35.69793881
36.25438858	34.48182837	36.24359022
35.02139545	34.89196022	35.82033959
35.85104971	36.82435698	36.42694918
34.85773894	35.18676565	34.47248423
35.57199663	34.08405424	34.57019613
36.86191891	34.56187611	35.52831208
35.69987772	35.59571437	36.15500595
36.10247965	35.07044571	36.65604721
35.88863878	34.99305063	36.36255472
35.85342746	36.03764737	35.13882022
34.52466103	36.09096439	34.69460236

10.7.3 Temperature Test 3

T1	T2	T3
84.81573708	85.26525115	84.73167983
86.25608833	84.97892109	85.17282694
85.64501751	85.7006665	84.90142611
85.36581901	86.87118952	84.28051261

86.31050858	86.97702046	85.24605357
87.11190394	84.18628674	85.48965636
86.23803536	85.40668873	86.88904128
85.40006441	85.6478893	85.74069581
84.62894217	84.94519383	84.55071868
85.44371909	85.93028508	85.19971828
85.75709544	86.89023557	84.55042834
85.94862996	84.93291309	87.31327737
84.60609972	84.20163319	84.61477354
87.21651704	84.43659964	87.03547961
86.18925167	85.0216351	85.49465631
85.12958597	84.47030111	86.75942664
86.30129524	86.89884944	84.80935437
84.3938645	84.23106419	87.0255265
87.2073417	84.25847465	84.10831271
84.44936228	84.16689419	87.08388975
86.29395299	84.49186015	84.02641528
84.20433468	85.62523617	86.26665291
86.75549719	84.24905846	84.73855838
86.36338709	84.66332767	85.45236499
84.45716534	84.48735668	84.67827463
86.15290531	86.06185629	85.79132049
86.29775453	85.09928244	87.3833923
84.54951278	85.48278354	85.33477971
86.76029439	86.09680553	86.18127754
86.69916199	85.9950739	84.58830431
85.30821225	86.36000953	86.01566622
84.62636097	85.38916895	85.26993229
86.50694512	84.15305691	84.18658875
84.03927225	84.31967619	85.71666518
84.47522223	86.41654051	84.99112158
85.29041764	85.21480747	86.38084569
85.22274103	85.25110857	86.68392503
84.77195745	86.15603714	84.91363624

86.68927273	86.91441106	86.7675756
85.6655343	86.74927145	85.33219544
86.16856445	86.25287081	84.16995418
84.40007838	85.0431092	85.63558883
87.18451084	84.13651013	84.3923639
84.5564392	86.31565283	84.31130768
85.16927989	84.05310481	86.84212958
86.61895881	84.79050965	84.58793837
86.20727313	87.3553973	84.39825302
86.27393766	85.39836296	85.33248659
87.11543796	85.83915365	84.75089876
85.1702113	87.39261131	87.49467797
85.29153388	84.12171288	86.3842655
84.25367977	85.12746842	84.21081729
85.11606361	85.99066114	84.01851534
85.19558903	86.0125249	84.13117493
86.63084941	85.21470772	87.14721833
85.32269715	85.5686138	85.92610053
85.35436317	86.75717121	86.11110478
86.50927515	84.60196894	87.49494096
87.14916978	84.07418418	87.2722511
86.1783612	85.61475154	85.80021468
84.22503903	86.92132695	85.73217306
84.01903279	84.7342502	85.91828458
84.68880721	87.45452404	85.58623242
86.4329877	86.82618874	86.94195925
84.3118833	84.80333549	85.69042029
84.66677446	84.19193628	87.43408004
84.75289253	87.29099225	85.32665672
85.94401278	85.31888277	84.77232364
86.19742412	84.1884404	86.08250958
84.72737427	84.76828027	85.79133495
85.96182127	85.8461427	86.28050455
86.09448996	84.25745639	86.63013487

84.48263226	85.06334211	84.32236329
85.10092854	87.11206389	84.1808134
86.8084809	84.10991128	85.00251619
87.21160631	86.66907549	84.97317487
85.73404891	85.42983132	87.44259853
86.85818698	86.26925569	85.97267192
85.42547131	84.66087212	86.0304362
87.4119897	86.15712071	86.03312112
84.86817584	86.42434899	86.69088479
84.25150917	85.35173592	87.08788691
86.40907245	84.75814061	86.27601589
85.84178037	86.65950658	87.45404847
85.72857645	85.51076144	86.74429667
84.68446608	84.99332188	85.02320471
84.77938034	86.32354632	86.48491604
86.03676133	85.16554155	85.94374461
85.3098652	86.13033787	85.86815533
85.74209037	85.93642995	84.52541724
87.39382775	85.19196001	85.82613395
86.46329366	86.27739248	86.29472611
85.73461816	85.88774181	86.55239258
85.32079343	84.7289444	85.46496665
85.57222097	85.05404454	85.29218208
84.21437623	84.17750814	85.78760543
86.69421929	84.62479819	85.5681421
84.66210161	86.10021611	84.59120604
86.83546998	86.39021227	86.16497931
85.32684795	84.94177688	84.47334381
84.622761	84.58486913	87.00005309
85.68762717	86.62759579	87.06209297
85.50735491	84.3972089	85.82926684
84.75557611	86.71973264	84.14293946
86.88434983	85.79987549	85.12574096
87.45407591	86.30051131	86.42966626

86.97896607	85.43950285	85.19534668
87.18842208	85.58779645	84.43323496
85.21852065	84.83932026	85.53283254
86.79660513	86.53783028	87.12935594
85.18887214	85.41654941	84.42604047
84.11185206	84.8693286	84.98579062
85.10594683	85.14612316	86.08622969
85.3064818	85.11610288	84.46600108
84.82861475	87.12400775	86.04900212
84.84125655	84.40618857	85.87215613
86.08723675	87.34764715	84.28351137
84.69475347	84.32126141	85.61239786
85.44279442	84.20212729	85.8748683
84.61237029	85.66054	86.62338649
87.39133785	84.67199953	85.66480791
85.32753081	84.94445885	86.23727641
86.45580747	84.10778056	87.43421599
84.79063015	86.62280807	85.65428352
85.00692638	84.41951121	84.77502884
86.78049106	85.50395801	86.55880282
84.22320962	86.71800784	86.89189781
87.29388473	84.80198733	87.15360514
85.56013888	86.83725055	86.21098913
84.15035145	85.78933921	86.87702709
85.47472714	84.17244843	85.59605837
84.94056871	85.57791857	86.12037398
84.5451104	85.72049903	85.52377581
84.07931978	86.99177498	84.97414789
86.48810167	86.88967269	87.07099099
87.13027713	86.40169454	85.16342605
84.59001064	87.11931822	84.18761106
86.49491903	87.30472472	84.18503767
86.008251	85.34818826	84.86797184
86.03662737	86.74002238	87.44647807

84.03062921	87.16065912	84.99051386
85.94209226	84.953134	84.93217343
85.30464256	84.20678249	86.24631739
86.34336642	86.27216459	86.75425418
86.10603731	85.70691611	87.0492291
86.58713321	85.13607817	86.08893855
86.84132293	86.15087367	86.66813875
86.48388588	84.16399716	86.0765979
85.08646276	84.36040106	86.20118487
86.50374674	84.34776936	87.1832152
85.31871141	84.99168077	84.05462388
86.35162538	84.26276781	85.35700902
87.09224207	87.18991201	84.77965927
85.21043321	86.97495417	84.17925752
84.33406092	86.01328312	87.16915766
84.53429852	86.77830301	84.62967975
86.26457127	86.00039694	86.23487824
85.11540013	86.95495552	86.88003603
86.59840659	84.8590288	84.36319652
86.28844852	86.94170899	86.67486818
87.30375656	85.98965883	85.72343576
86.54034171	87.11797297	86.87456833
86.04186512	84.3732599	86.74454417
84.71253862	86.26102573	84.55633167
85.19877146	86.73712676	87.15122039
84.33769025	84.62973096	84.4152004
84.23631634	86.76696343	87.10452999
85.396609	87.32483929	87.4665568
85.73315811	85.14891373	85.72094896
86.28732721	86.19617397	86.14837589
85.86479204	84.25671454	87.17057327
85.2244788	86.63952374	86.19565713
86.56666448	86.24070695	85.30445285
84.45825665	85.39063536	85.2391313

87.47031327	87.43516737	85.93372078
87.48054711	86.22357811	85.75807333
85.85949206	84.97548699	85.43202949
84.54837698	86.11576433	84.62437926
85.42346169	86.06836095	84.9369769
86.96744729	85.62344135	86.75759575
87.02663097	84.28664404	84.36646378
87.37125251	85.31747356	86.40298929
85.95763936	84.78024272	87.49959909
85.47312971	84.15921047	85.44574223
84.78349355	87.40876048	86.10774413
84.14266959	85.62627828	86.72175295
86.23376098	86.17807548	87.39501415
85.13640116	84.05585678	85.1311603
86.17090875	84.20815316	87.25603916
85.03177626	85.61262458	86.94551593
84.21204555	86.39377908	84.99561523
85.54852145	85.66582614	87.12848388
85.22752506	85.85321642	85.94516883
85.08173906	85.30716385	84.40708723
86.6276115	85.26577109	84.84642844
85.79515615	85.24232862	85.79396268
85.91076922	85.39068289	86.04659912
85.57311499	87.05440876	86.61798142
84.58875319	85.96350611	84.03149701
84.0878101	85.17916532	85.88219313
86.25374523	84.59811783	86.03490428
86.69951992	87.13881053	86.48462787
86.74206523	84.26267623	84.05180774
85.65681579	85.29260757	84.47341722
87.08480721	84.86528583	87.29968518
84.04968978	85.94603105	87.31637653
85.57700122	87.4129213	87.24169212
86.67902309	85.08363772	87.06411621

86.42902445	84.31748014	84.28610107
85.34266515	85.19903549	87.23145624
84.77399573	85.10190926	85.51186406
87.19253783	84.48477822	85.58501415
85.95636118	84.43247656	85.14646319
85.14359312	86.0386137	84.46499067
86.03494679	85.90978414	84.92161253
84.20817989	85.36477855	85.61011732
85.04012278	86.51947551	85.58875784
84.39652632	86.90104293	84.55924995
87.07264855	86.36623973	87.25014637
85.56278107	84.03659705	86.84805184
84.6410781	86.62321763	84.31450918
86.54761024	85.78183599	85.47051602
86.87668252	87.26886631	85.85769597
87.47301768	86.6894096	84.66272503
85.33764584	86.68941021	87.34805998
85.29718799	86.0638726	87.25926527
85.51731764	85.79592566	86.70070451
87.24017097	86.04379218	86.35422015
84.88063949	84.67867316	87.172142
86.61261312	85.13069529	84.74081779
86.09613848	87.14934972	86.8923002
84.62603307	86.6107695	86.74284952
87.16229521	86.24541513	84.40052371
85.05494533	84.98135255	86.76218133
86.00241345	84.9945191	84.67921761
84.56812673	87.22009016	84.23942406
84.45366388	84.105372	85.73756975
86.22222025	87.01190616	86.63751264
84.88763256	86.3222618	87.13944531
84.12847873	84.47087853	86.05306113
85.18192678	84.30099142	85.69671036
85.68961692	85.0305728	85.2194471

87.22718762	85.54552962	85.79110445
86.37438877	85.94873611	86.98423957
85.8898016	85.73742012	86.4751147
87.23261034	86.92845159	86.60649441
84.99783592	86.1583396	85.64242068
84.77195746	86.94740463	84.37532052
86.3429355	84.60354989	87.01027881
86.85308311	85.23589413	87.08777187
84.86238054	84.12783642	87.00513066
85.13358848	84.53027187	84.74339145
86.02803701	87.11240314	84.57578286
87.19038188	87.05580406	86.10554927
86.31891908	86.36171279	86.139271
84.69925251	86.06729593	87.40105155
84.01447953	87.39308685	84.80813883
86.28943978	87.07950928	84.77339912
86.04105661	85.59274014	85.49602766
87.04800012	86.88633868	84.97964346
85.83521421	86.34261997	87.29444396
84.91634979	84.55832986	84.82319919
84.10764333	86.84979682	84.47717791
84.73103603	87.3844449	84.70644754
84.67906125	85.80013572	84.29147319
85.65815921	87.22457023	87.26780563
84.99589443	87.34054425	85.2346406
87.29061526	85.85426958	87.05249309
87.27580227	84.03828767	87.29088621
84.4181677	86.22939428	85.55791618
87.09016485	86.05478576	85.09175217
84.18009562	84.89365009	87.46090079
84.39798016	87.2771351	86.99239277
86.12064774	86.19189918	84.08146553
84.54019527	87.2095577	86.17911931
87.02968205	84.08223718	86.28705916

86.8989142	86.71435838	87.29701366
85.46966875	86.93368156	85.00184618
84.08731505	86.29713647	85.34225279
85.70972005	86.62814162	85.33144711
85.65517264	86.93235487	86.1859914
86.96559166	85.90629748	86.15731961
84.46487739	87.46540819	86.29566878
85.22884731	84.90972257	84.92355516
87.18359693	84.48758891	85.80744954
85.06981414	86.70524683	84.04818472
87.17012475	86.25643453	86.36874804
84.66788406	85.76293472	85.92111479
85.05795528	85.01051961	85.25256495
87.42348124	84.9034686	85.32059215
84.79872075	85.85355812	85.51237492
84.09204452	84.99307072	84.72273717
84.55840664	85.24550377	84.86581286
86.86914044	85.72981416	85.61634425
86.23035631	86.94003187	85.01223404
85.70387406	86.4212784	85.95406959
85.74818694	84.13927106	84.99976114
85.69570414	84.36357071	86.19659369

10.8 Raw IMU Test Data

Test 1 - X-Axis	Test 2 - Y-Axis	Test 3 - Z-Axis
0	1	0
1	359	0
359	359	0
359	0	1
0	1	359
0	0	359
0	0	359
1	0	1
0	1	0
0	0	359
90	92	89
91	91	90
90	88	91
89	87	92
88	90	90
89	90	91
91	89	88
92	91	87
90	92	89
88	89	88
180	181	179
182	181	178
177	180	180
182	178	181
179	178	182
181	179	180
179	179	182
180	180	182
180	179	181
181	181	179
270	271	268

271	269	266
267	269	270
270	269	272
268	267	271
268	272	270
279	271	269
262	270	269
270	271	269
267	270	271
0	0	1
1	0	1
0	0	1
359	0	1
0	1	0
1	1	1
0	359	1
0	359	0
1	1	1
0	0	0