Autonomous Car with Android-Based Control

Brendan Behrens (CE), Richard Bustamante (CE), Madison Mastroberte (EE), Terrence Skibik (EE)

Advisor: Dr. Adegbege

ELC-496: Senior Project II

Spring 2019

Acknowledgements

We would like to thank our advisor Dr. Adegbege for his role in supporting us as we worked through the first semester. We would also like to thank the School of Engineering for the funding and support of our project.

Fulfillment Page

Autonomous Car with Android-Based Control

A senior project submitted to the faculty of The College of New Jersey by:

Brendan Behrens
Richard Bustamante
Madison Mastroberte
Terrence Skibik

In partial fulfillment of the degrees of:

Bachelor of Science in Computer Engineering and Bachelor of Science in Electrical Engineering

May 2019

Abstract

As demand increases for completely autonomous vehicles, new methods for navigation and safety need to be investigated to determine a viable solution. A small-scale car was built and modeled to explore real-time control applications. The project is intended to illustrate the behavior of an autonomous vehicle with a unique obstacle avoidance algorithm to navigate an unforeseen environment. The autonomous car is capable of following a predetermined route and adapting to avoid unforeseen obstacles. This system consists of an Android phone, Android application, Raspberry Pi, and Arduino microcontroller. The Android phone serves as the application environment, a front-end user interface with the system. The on-phone application receives video feedback from the car and provides the user options, such as car direction, speed and manual override. The Raspberry Pi processes video and broadcasts a live-stream to the Android device, while also acting as the car controller. In addition, the Raspberry Pi also serves as a mode of communication between the Arduino, via serial connection, and the Android device, via wireless communication. The Arduino provides a node for sensor readings, which are sent to the Raspberry Pi and Android phone for processing. Overall, the project simulates the methodologies being investigated today in the field of autonomous systems.

Table of Contents

I. List of Tables

II. List of Figures

III. Introduction

The initial concept for this project was to develop a car that could race autonomously. However, the team wanted to spin this concept and develop a car that would move autonomously but would also allow a person to connect to it using their phone so they can watch where the car goes, and control it if need be. One possible use for this system is navigating a dangerous area. A user will now has the ability to connect directly to an autonomous car. They could then tell it where they want the car to go and watch it travel and navigate through the terrain. Smartphones are becoming more integral to society, for that reason the team is integrating an Android device into the system. Developing an app for mobile devices would allow anyone to access the car to control and view its movements if they have the correct login information. Exploring this use case gives the team motivation for developing an autonomous system that has a quick response time and a real time video stream, giving users instant interaction with the car.

To fit this use case, the goal of this project is to design and build a small car with autonomous control. The car must also be able to stream a video stream to a mobile device which can control the car if need be. The team will do this by interfacing an Arduino board to control the hardware components, a Raspberry Pi to perform the autonomous control software and video stream, and an Android device to watch the stream with a button overlay to control the car. The interactions between the different components are seen in Figures 1.1 and 1.2

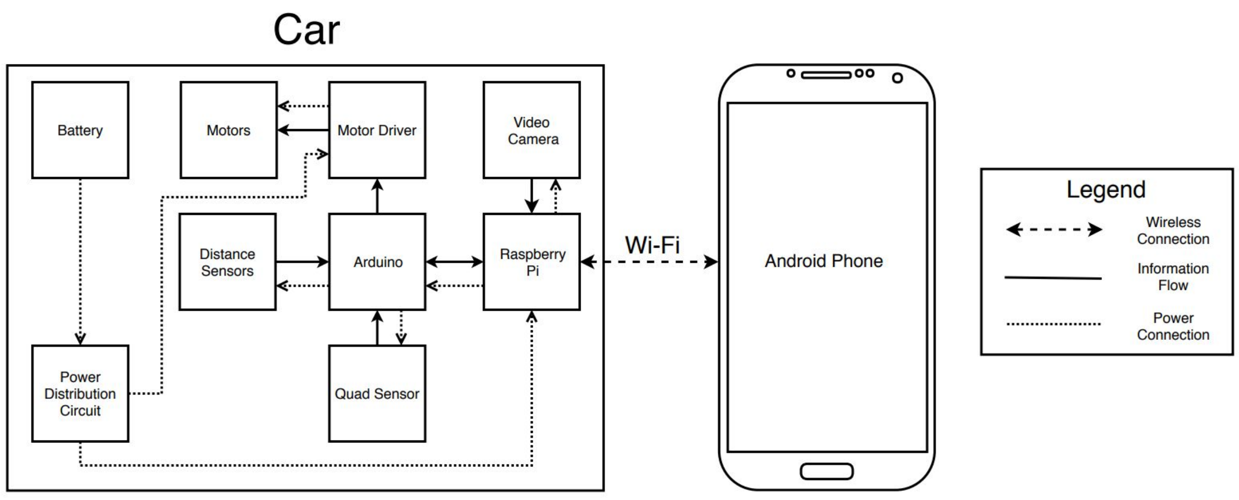


Figure 1.1 Hardware System Diagram

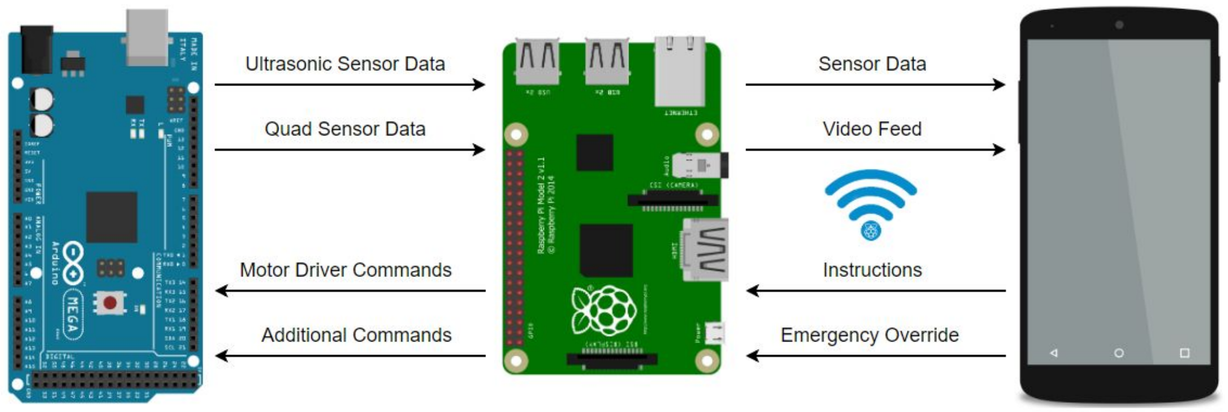


Figure 1.2 Communication Systems Diagram

This paper starts off by going over the design specifications that need to be met and explains why the specifications were chosen. Next, background information will be discussed about autonomous vehicles and how they can be applied to real life scenarios. The first development discussed is the power system. There, the design, need, and implementation is of the system's power system is discussed. After the power system, the Android app development is analyzed. In this section, the methods used to develop the app, the first prototype and the next version are discussed. The first prototype is where the system is now, and the next version of the app will be more robust and include more functionality. Next the software and system development for the Raspberry Pi is discussed. First the Raspberry Pi had to be configured a specific way to meet the system requirements. Then the software implementation for the Raspberry Pi is discussed. After the Raspberry Pi, the Arduino system is discussed. There are hardware and software requirements for the Arduino since it acts as the middleman between the system's hardware and software. The way these components are implemented and how the Arduino communicates with the different hardware is discussed as well as why it works the way it does. The paper finishes with what was accomplished this semester, what were some obstacles, how did the team overcome them and what will be done next semester.

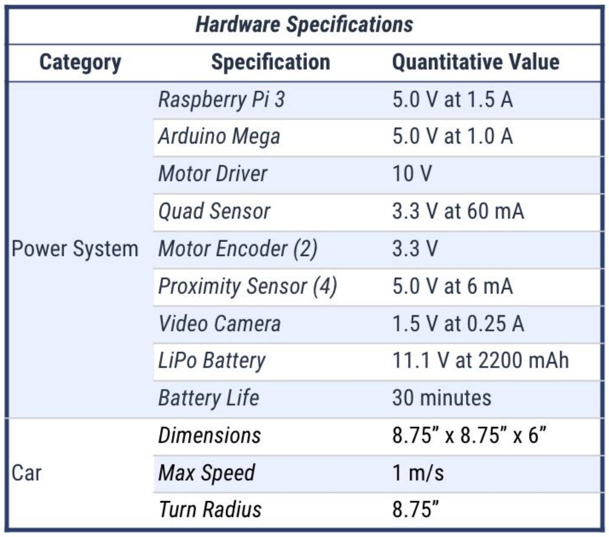
IV. Specifications

When defining the operations of the Autonomous Car, and number of system specifications were derived in order to help keep development of the car on track as well as having a set of concrete goals to abide by. The specifications are listed out for the different desired areas of operation below. What follows is a brief discussion of their details and importance.

Wi-Fi was chosen to be the form of communication between the Raspberry Pi and Android device. The Pi came with a Wi-Fi card that determined all the specifications besides the latency. Less than 1 second latency is desired, and the figure of 1 second was chosen as an upper bound not to cross. The reason it is so high is due to the desire to stream video to the phone in addition to the other data being sent.

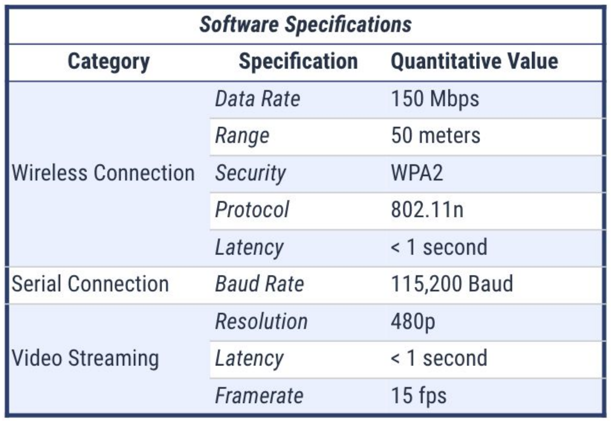
The specification for the video stream were determined so that it would have a good enough quality so that the user could distinguish details in the feed while it still seems like real time. The car dimensions were chosen so that all the required hardware could comfortably fit inside while not also being overly large. The power system specifications were derived from the different components that require power on the car. These specification are where decisions for the power system and batteries were derived from.

Table IV.1. Hardware Specifications for System



Hardware Specifications		
Category	**Specification**	**Quantitative Value**
Power System	*Raspberry Pi 3*	5.0 V at 1.5 A
	Arduino Mega	5.0 V at 1.0 A
	Motor Driver	10 V
	Quad Sensor	3.3 V at 60 mA
	Motor Encoder (2)	3.3 V
	Proximity Sensor (4)	5.0 V at 6 mA
	Video Camera	1.5 V at 0.25 A
	LiPo Battery	11.1 V at 2200 mAh
	Battery Life	30 minutes
Car	*Dimensions*	8.75” x 8.75” x 6”
	Max Speed	1 m/s
	Turn Radius	8.75”

Table IV.2. Software Specifications for System



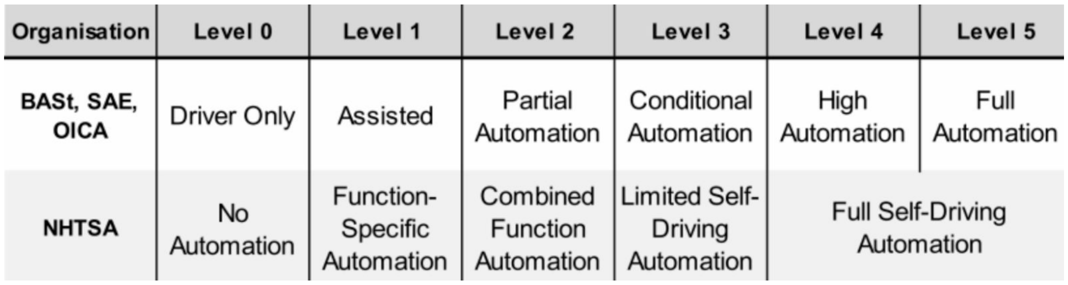
Software Specifications		
Category	**Specification**	**Quantitative Value**
Wireless Connection	*Data Rate*	150 Mbps
	Range	50 meters
	Security	WPA2
	Protocol	802.11n
	Latency	< 1 second
Serial Connection	*Baud Rate*	115,200 Baud
Video Streaming	*Resolution*	480p
	Latency	< 1 second
	Framerate	15 fps

V. Chapter 1: Background

A. Definitions

An autonomous car is a vehicle that has the ability to gain knowledge outside of its system and does not need human interference to function. Autonomous systems often use sensors and a processor to collect, store and analyze information about their environment in order to operate. There are various levels of autonomy, as classified by various organizations (BASt, SAE, OICA, and NHTSA). The table below shows the level variations, based on the four agencies listed.

Table 1.1. Levels of Autonomy for Vehicles.



Organisation	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
BASt, SAE, OICA	Driver Only	Assisted	Partial Automation	Conditional Automation	High Automation	Full Automation
NHTSA	No Automation	Function-Specific Automation	Combined Function Automation	Limited Self-Driving Automation	Full Self-Driving Automation	

The system discussed in this paper will be classified as a level 5 or "full automation," which is defined as a vehicle that can drive with no human driver and under all circumstances a human could. The behavior of an autonomous car could be defined in three sections: sensing, processing and reacting. The sensing is performed through sensors (Lidar, Ultrasonic, Radar) and video cameras. The processing is performed via an on-board processing computer, that utilizes the sensor data to make analyses on the environment. Next, the vehicle will adapt its systems to "react" to any changes in the environment.

The system being developed will utilize a Wi-Fi connection to control the vehicle from an IoT device. The on-board sensors are ultrasonic, magnetometer, accelerometer, gyroscope and video camera. Processing will be done by an Arduino Mega and Raspberry Pi, which will compute two control algorithms. Reaction to change will be done via motor drivers and front axle to change direction and slow the vehicle down.

B. Applications

There are a variety of applications for autonomous systems such as traversing unsafe or dangerous areas. This use allows for users to visually see an environment, move into areas due to small size of the vehicle and make decisions without physically being. Another application is a testing unit for ride-share corporations, so that safety and implemented tests can occur on a smaller vehicle before potentially damaging a large-scale system and mitigate safety concerns.

VI. Chapter 2: Supporting Systems (Madison)

A. Introduction

Power systems are an integral component for a functional design that includes various microprocessors and motors. Each has specific constraints which requires the power system to power both high current and precise voltage for sensitive electronics. The system consists of a buck converter and linear voltage regulator, their specific operations and characteristics will be discussed in a later section.

The chassis was an important component for the design since it serves as a housing unit and protects the major system components. In particular, the chassis creates the structural skeleton of the design and provides support for connections between the motor, motor driver, and software components.

The chapter is broken up into various sections such as, need, specifications of the circuit, components of the system, design, simulations and testing. The chassis will be discussed in terms of overall set-up and future recommendations. These sections will provide context and support for the design, while addressing ways to improve the system for the future.

1. Need

The overall system consists of multiple devices, ranging from microprocessors, motors and sensors. Each device has a designated operation voltage and current draw, which add complexities to the system operation, in terms of power distribution. Although various components can be powered by cascading devices via serial connection, system specifications are required due to constraints of particular components. Specific attributes and constraints of the system will be discussed in section 2.A.2 Require Specifications and Constraints.

The system required a chassis to serve as the frame of the car and integrate all the systems. Since mechanical engineering was not the focus of this senior project, a chassis was purchased with the smallest dimensions necessary to appropriately fit each major system component.

2. Required Specifications and Constraints

The system requires a standalone power system that is mountable, compact and produces a continuous and stable DC output to operate each component. These parameters are a result of the moving isolated system functionality. The system will have two outputs, 10.0 volts (V) to supply the motor driver and 5.0V at 3.0 amps (A), capable of supplying a peak draw of 2.803 A to supply the Raspberry Pi 3 (Pi). Since the Pi is used to power the Arduino Mega, camera module, quad sensor and proximity sensor, the amperage required will be above the designated 2.5A. Figure 2.1 shows the operating voltages and current draw for the overall configuration. Table 2.1 shows the power distribution of the system.

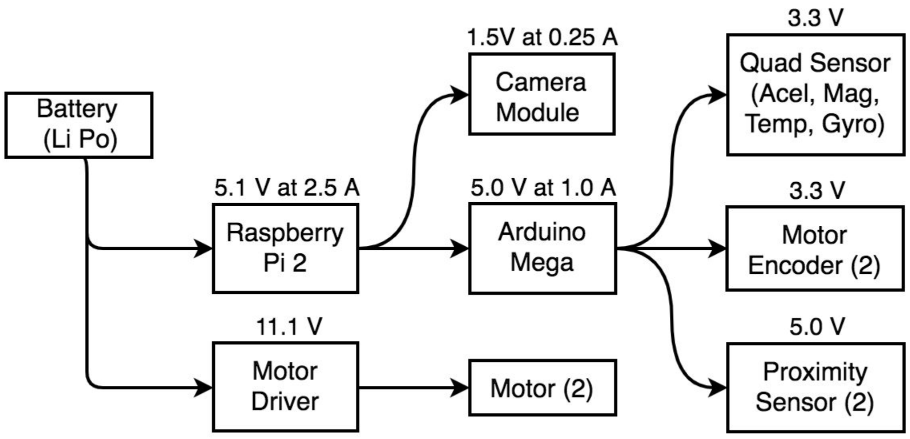


Figure 2.1 Operating voltages and current draw for power system with relational connections for distribution.

Table 2.1 Operating voltages and current draw for power system with relational connections for distribution.

Component	**Quantity**	**Voltage (V)**	**Current (A)**	**Power (W)**
Raspberry Pi 3	1	5.1	1.5	7.65
Arduino Mega	1	5.0	1.0	5.0
Camera	1	1.5	0.25	0.375
Quad Sensor	1	3.0	0.05	0.15
Proximity Sensor	4	5.0	0.024	0.12
Motor Encoder	2	3.3	Misc.	Misc.
Total	**10**	**22.9**	**2.824**	**13.295**

The system requires a chassis that can support the model application, which means small-scale size and easy-to-control using the Android device. Since the system will not be tested on rough terrain, the clearance of the model will be low. Overall, the components the model must house are the Raspberry Pi 3, the Arduino Mega, lithium polymer battery, motor driver, motor and sensors. However, the largest components (i.e. Raspberry Pi 3, Arduino Mega, and battery) are the determining factor to the size of the chassis. Table 2.2 shows the length and width of the components in inches.

Table 2.2. Dimensions of Components in System

Component	Length (in)	Width (in)
Raspberry Pi 3	3.38	2.20
Arduino Mega	4.33	2.08

Battery	4.17	1.30
Motor Driver	1.50	1.50
Power System	2.36	3.15
Motor	3.8	2.8

B. Power Subsystem

The power system is composed of 2 integrated circuits (ICs) and a battery for the major components. This section will discuss the hardware specifications, such as technical characteristics of the chosen components. In addition, the sub-circuits will be discussed in terms of their functionalities and schematics.

1. Hardware

i. Battery

A Lithium Polymer (LiPo) battery was chosen as the source due to its characteristics, as compared to other battery types. Weight is an important parameter to consider when building a vehicle, which resulted in choosing a LiPo battery over other available options. LiPo batteries are characteristically lighter weight, smaller in size, and have higher discharge rates [X]. Some disadvantages of using LiPo batteries are shorter lifespans (several hundred charging cycles), require careful handling and charging procedures. The disadvantages were mitigated by the overall project duration and a battery charger that safely charges LiPo batteries.

The battery chosen was an Ovonic battery with a voltage of 11.1V and 2200mAh capacity. The battery is 3S, which means there are three separate cells in series. The battery's discharge rate is 50C and results in a discharging rating of 110A, which is the maximum sustained load safely put on the battery. Figure 2.2 shows the battery, with a JST-XH and Deans plug for charging and discharging. To safely handle the battery, the battery may not go under 9V, which is the low voltage cutoff. In order to charge the battery, the charge rate must be set to 0.5C - 1.0C and have a 24.42W charging power.

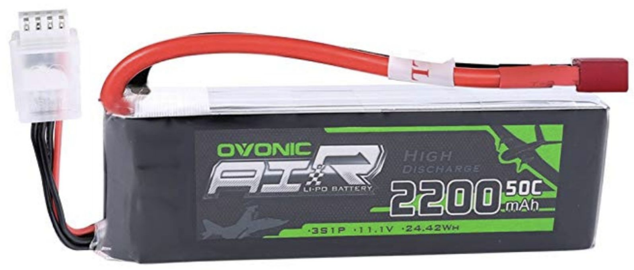


Figure 2.2 11.1V 2200mAh 3S 50C LiPo battery

One specification of the project was a 30 minute battery life. With the chosen battery those specifications were met using the following battery life equation:

$$Battery\ Life\ =\ \frac{C_{Bat}}{I_{Load}} * 0.7 \qquad (1)$$

This calculation determines the estimated hours that a battery can run given the battery capacity in mAh divided by the device consumption in mA. The quotient then is multiplied by a consumption rate, which is set to 0.7 for allowances of external battery factors. The estimated hours found for the battery life was 0.513 hours or 30.78 minutes, using a worst-case draw of 3.0A.

ii. Design Approaches

There are a variety of methods to construct a power system using buck/boost converters, and linear voltage regulators. Three designs were evaluated for the buck converter portion which includes utilizing various chips: LM317, AZ34036U, and LM2678T-ADJ. The LM317 is a linear voltage regulator that can be used to step down voltages. There are numerous downsides to this method, specifically heat dissipation and inefficiency. In particular, the heat dissipation for the LM317 for a 11.1V to 5V step-down is 18.3 Watts (W). The next viable option was the AZ34036U, which is a buck/boost converter. However, this IC does not contain a control system and therefore cannot adapt to changing current draw. Also, there is significant ripple on the output voltage signal, which requires more components (like capacitors) and a linear voltage regulator to create a stable output. The last choice was the LM2678T-ADJ, which contains an adapting duty-cycle controller, does not dissipate excessive heat and outputs ripples of less than 40 mV. As a result of these findings, the LM2678-ADJ was used for the buck converter, while the LM317 was used for the 11.1V to 10V step-down.

iii. DC-DC Buck Converter

A buck converter was used to step-down the input voltage from the multi-cell LiPo battery. The configuration in the most basic form consists of a switch, diode, and filter (capacitor and inductor). In the project's application, an IC, such as the LM2678T-ADJ will be used as the switching component.

a. Design

The design of the buck converter utilizes the LM2678T-ADJ chip from Texas Instruments. Figure 2.3 shows the pinout of the chip, in hardware the pins are alternated between two rows. Pins 1, 3, 5, 7 are in the front row, whereas pins 2, 4, 6 are in the back row. This configuration will result in some changes in the soldering layout.

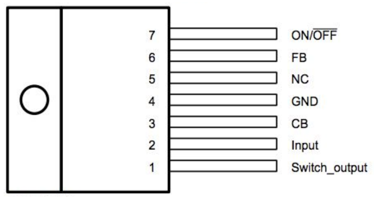


Figure 2.3 Pinout of LM2678T-ADJ Chip

In terms of functionality, the chip acts as a buck converter with an adaptable voltage input and duty cycle. Within the chip, there is a buck converter and a control system to continuously output the specified voltage value. The continuous output is generated by varying the duty cycle as the current draw changes. The chip works with various components, such as an inductor, schottky diode and capacitors, to act as the controller of the buck converter. Figure 2.4.a shows the schematic of the buck converter circuit with an input of 11.1V from a multi-cell LiPo battery and an output to the Raspberry Pi GPIO pin.

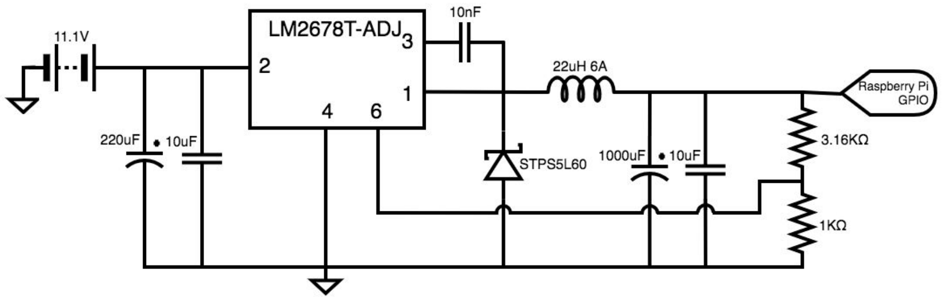


Figure 2.4.a Buck Converter Circuit with LM2678T-ADJ Chip

Input capacitors were utilized for the chip to operate, this component allows for the chip to switch current from the input source at high speed. If the capacitance is too low, the parasitic inductance of the input voltage wire will limit the amount of switchable current the chip can utilize each switching cycle. The value of the input capacitor was given by the datasheet [X]. The diode allows for current to flow in a complete path when there is switching in the circuit, which mitigates inductive spiking and voltage decreasing below 0V. Specifically, a schottky diode was chosen for the low forward voltage and minimal heat dissipation. A bootstrap capacitor was used to drive the switching transistor inside the chip, which connects output and switch pins. The value of this capacitor is designated by the datasheet [X], and should have a 50.0V rating. The output capacitors are used to smooth out the high frequency content of the switching waveform, so that there is a stable DC output. The feedback resistors determine the output of the chip, in this particular configuration the output was set to 5.0V. Since accuracy is an important factor, 1% tolerance resistors were used.

b. Construction

In order to create the circuit, there were particular specifications to ensure the efficiency of the DC step-down. Figure 2.4.b. shows the soldering layout designated by the datasheet.

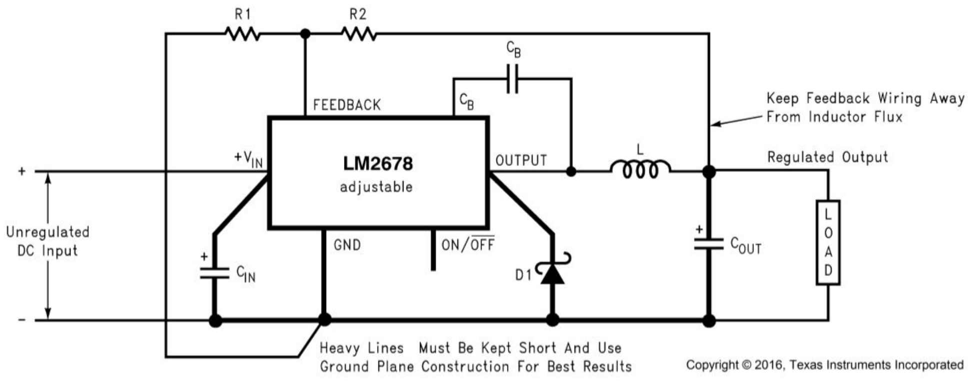

R1
R2
C_B
FEEDBACK
C_B
Keep Feedback Wiring Away From Inductor Flux
$+V_{IN}$
LM2678
adjustable
OUTPUT
L
Regulated Output
Unregulated DC Input
C_{IN}
GND
ON/$\overline{OFF}$
D1
C_{OUT}
LOAD
Heavy Lines Must Be Kept Short And Use Ground Plane Construction For Best Results
Copyright © 2016, Texas Instruments Incorporated

Figure 2.4.b. Buck Converter Circuit with Soldering Layout

Since the circuit operates at high voltage, the board was laid out to adhere to these requirements. The IC was placed in the center, ground line was designed to be one continuous thick solder line and components were placed to reduce the distance between one another. The last constraint was the feedback wire layout from the output resistors, which was placed furthest from the inductor to avoid coupling due to inductive flux. Ceramic capacitors were soldered underneath the PCB to ensure the shortest distance from other components. Figure 2.5. shows the solder construction of the buck converter.

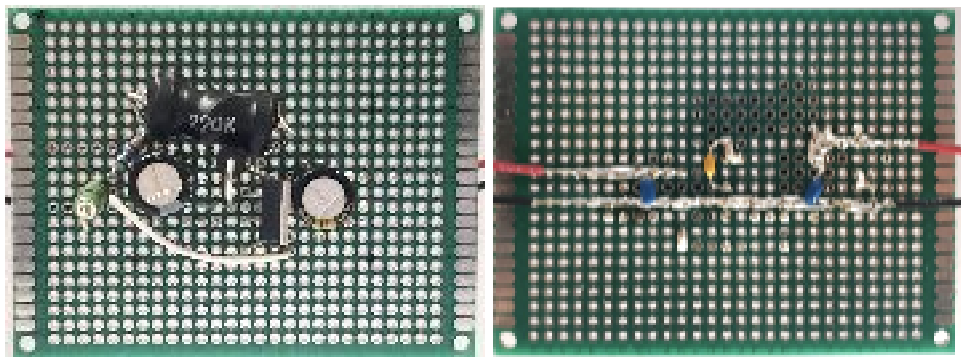


Figure 2.5. [Left] Front View of Buck Converter Circuit on PCB. [Right] Bottom View of Buck Converter Circuit on PCB.

iv. Linear Regulator

For the motor driver, the operational voltage was set at 10.0 V. Since this difference between the input and output is 1.1V, a linear voltage regulator was used. The LM317 is a low-cost, readily available, and efficient IC that will be used for the circuit. The LM317 works by connecting the supply voltage to the input pin, then using a voltage divider to set the output voltage. Figure 2.6. shows the pinout diagram of the chip.

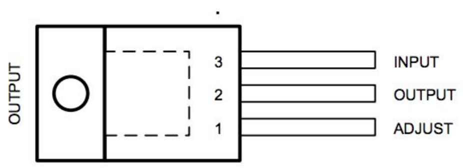


Figure 2.6. LM317 Pinout Diagram

The values for the resistors were calculated using the following equation:

$$V_o = V_{REF}(1 + \frac{R_2}{R_1}) + (I_{ADJ} * R_2) \quad (2)$$

Figure 2.7 shows the circuit for the linear voltage regulator with a resistor ratio of 7:1 (R2:R1).

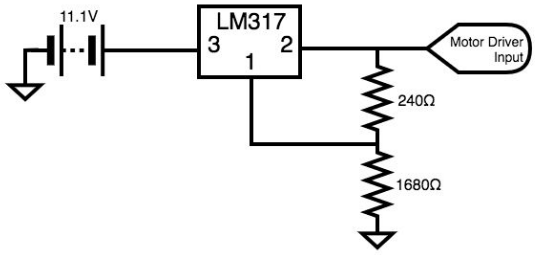


Figure 2.7. Linear Voltage Regulator with 10.0 V Output Using LM317

2. Simulation

For the purposes of circuit verification from a theoretical standpoint, simulations were performed utilizing LTSpice and PSpice of each individual component comprising the power system.

i. LTSpice Simulation Buck Converter

There were some issues simulating the chip on both LTSpice and PSpice, which resulted in an approximated buck converter. In other words, two simple configurations of the buck converter were simulated to illustrate the behavior of the LM2678T-ADJ chip. One simulation is a simple buck converter consisting of a diode, capacitor, inductor and pulse source. One other simulation was a darlington pair of transistors, which mimics the switching mechanism and is prevalent in many buck converter designs. Figure 2.8.a and 2.9.a. show the schematics for the two variations of buck converters. In each case, the simulation shows some oscillations in the initial start. However, due to the control circuit in the LM2678T-ADJ this overshoot should not be evident in the physical circuit, since the duty cycle will adapt to only output 5.0 V.

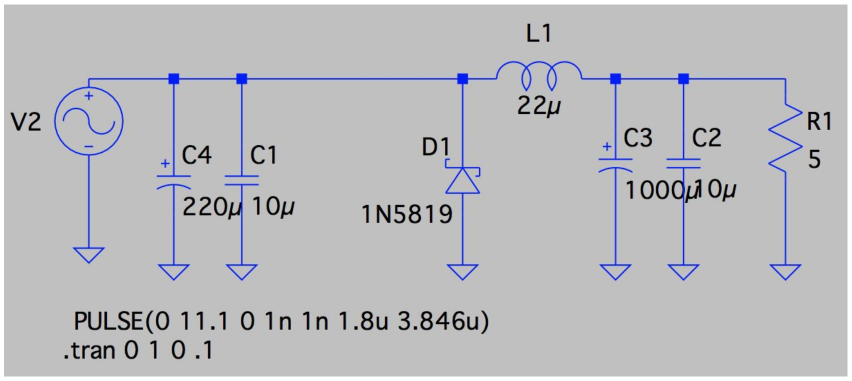


Figure 2.8.a. Simple Buck Converter Using Component Values and Load of 5 Ohms in LTSpice

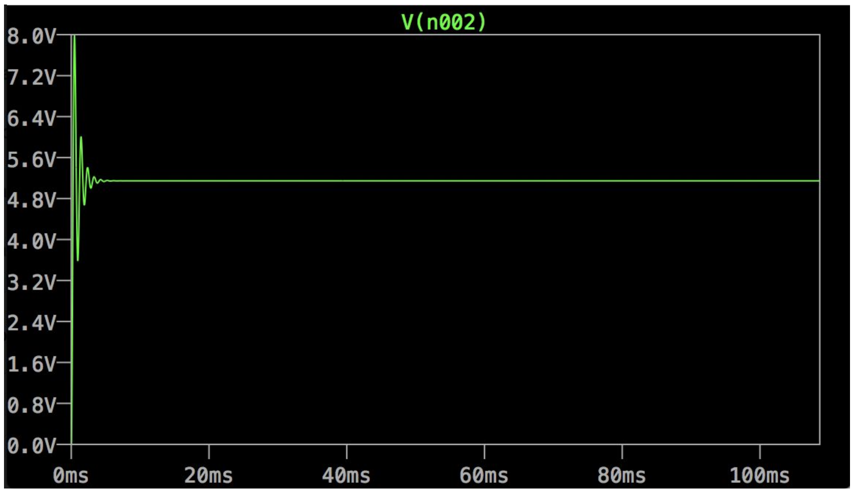


Figure 2.8.b. Simple Buck Converter Supplying 5.031 V from 11.1 V Source

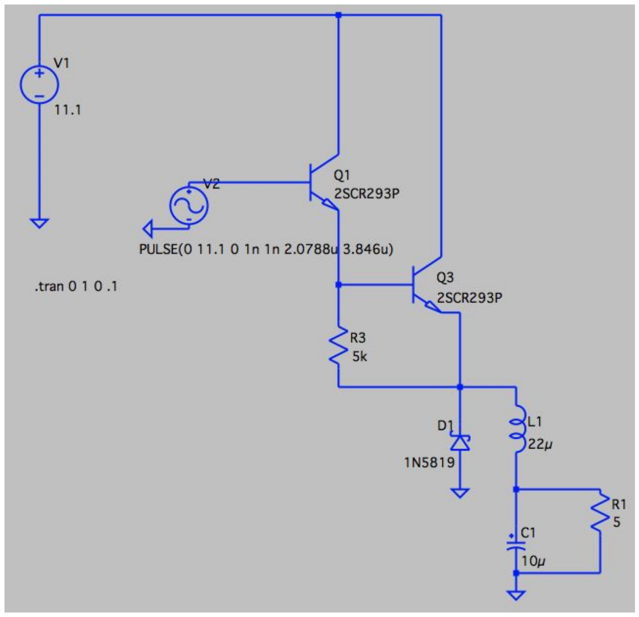


Figure 2.9.a. Darlington Buck Converter Using Component Values and Load of 5 Ohms in LTSpice

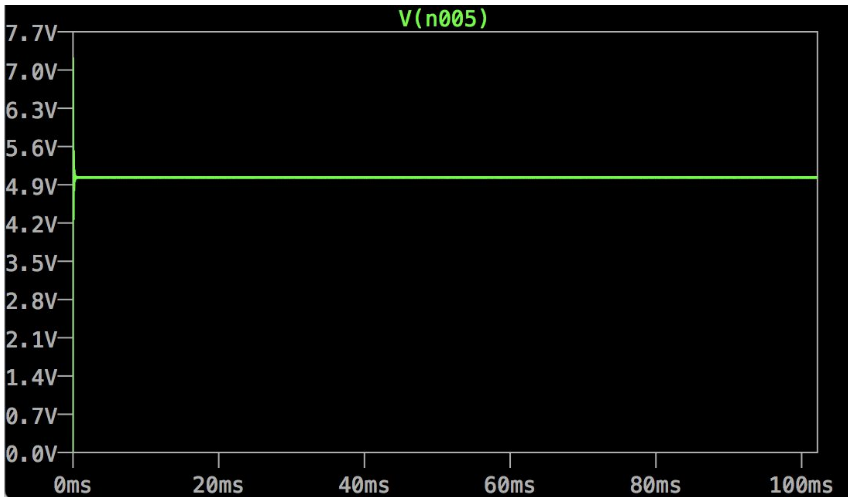


Figure 2.9.b. Darlington Buck Converter Supplying 5.031 V from 11.1 V Source

ii. PSpice Simulation Linear Voltage Regulator

Figure 2.10.a shows the schematics for the linear voltage regulator, which features the calculated resistors values and ratio. Figure 2.10.b shows the 10.0 V supply from the LM317.

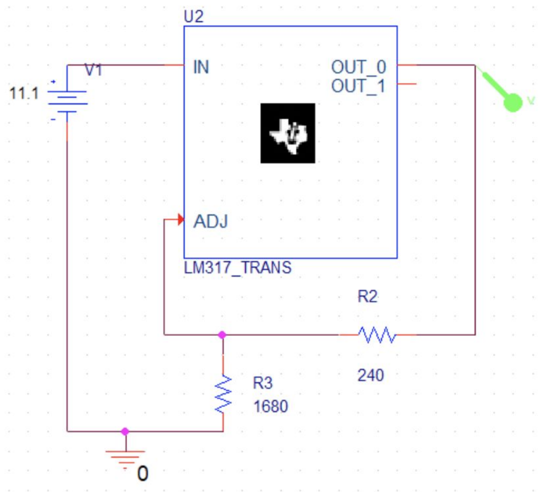


Figure 2.10.a. Linear Regulator Using Component Values in PSpice

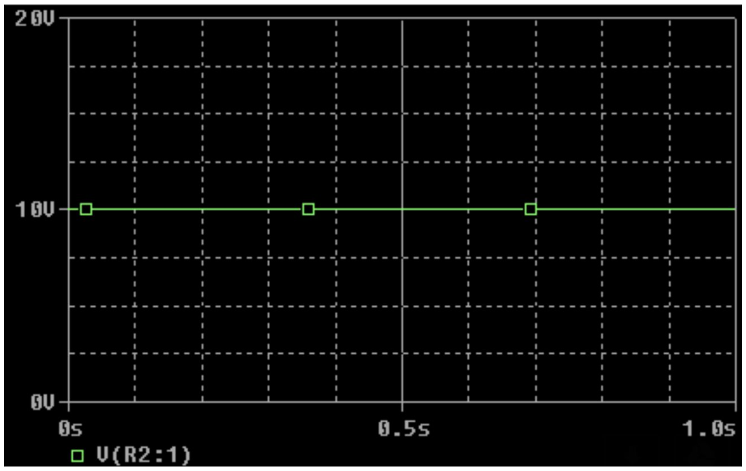


Figure 2.10.b. Linear Regulator Supplying 10.0 V from 11.1 V Source

3. Conclusion

Overall, the power system was design, constructed and simulated with relationally similar buck converters. This estimation was done to mitigate issues on PSpice with the LM2678T-ADJ chip model and convergence to due simulation timing issues. Both configurations tested supplied 5.0 V, but did have some visible overshoot. Upon testing with a higher load, the hardware buck converter did not reach the output voltage necessary. Therefore, a linear voltage regulator was placed on board. Upon testing, however, the chips began to dissipate heat due to the high current draw and hitting the chip maximum source current. Since the heat dissipation was high, a heat sink was installed, however, due to concerns of reliability the circuit was chosen not to be used. Therefore, a secondary battery is used to power the Raspberry Pi, which then powers the other modules.

4. Future Recommendations

Utilizing a PCB with a buck converter, either pre-built or designed, is the most optimum method to step-down the voltage, since the critical efficiency due to length of connections will be more optimal. It is also more optimal to directly power the Arduino Mega from the battery using the power pins, since this reduces the load on the Raspberry Pi. Even though the Arduino is powered through the serial cable, the This will then reduce the necessary amperage to power the Raspberry Pi.

C. Chassis

1. Construction

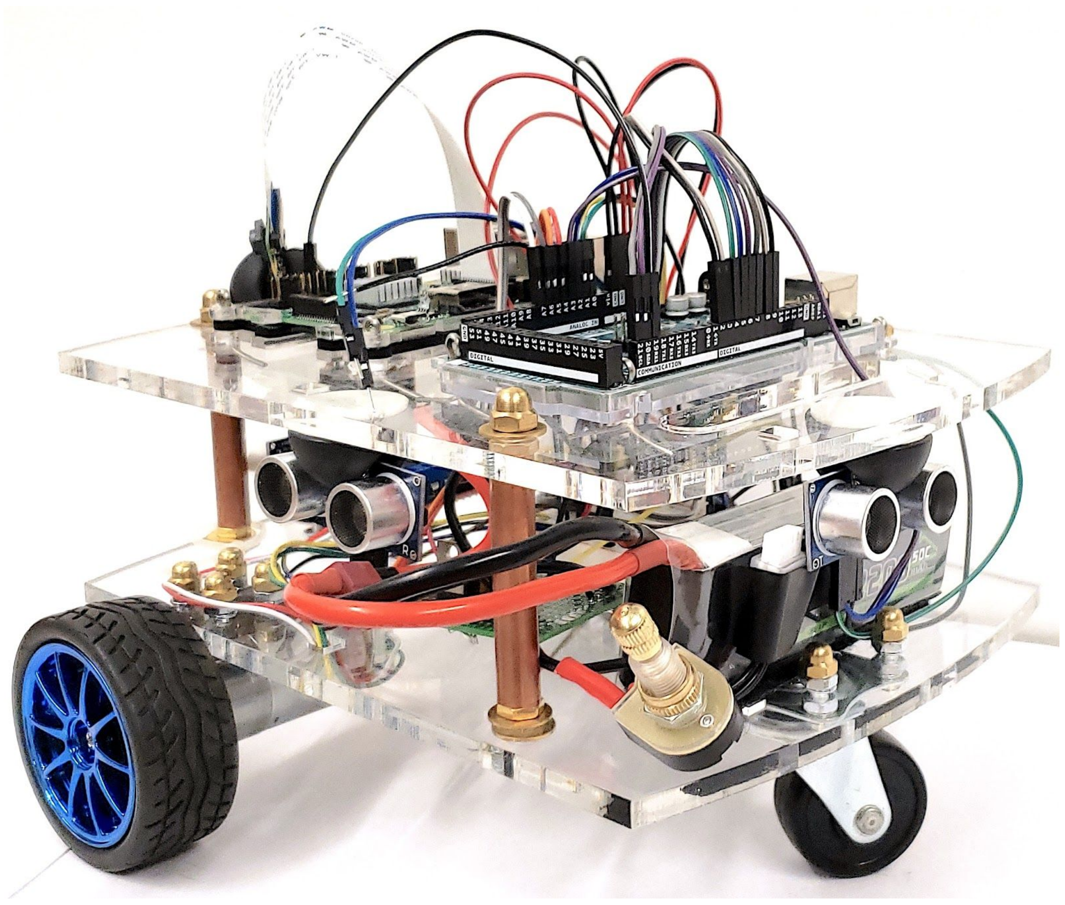


Figure 2.11.a. Constructed Car

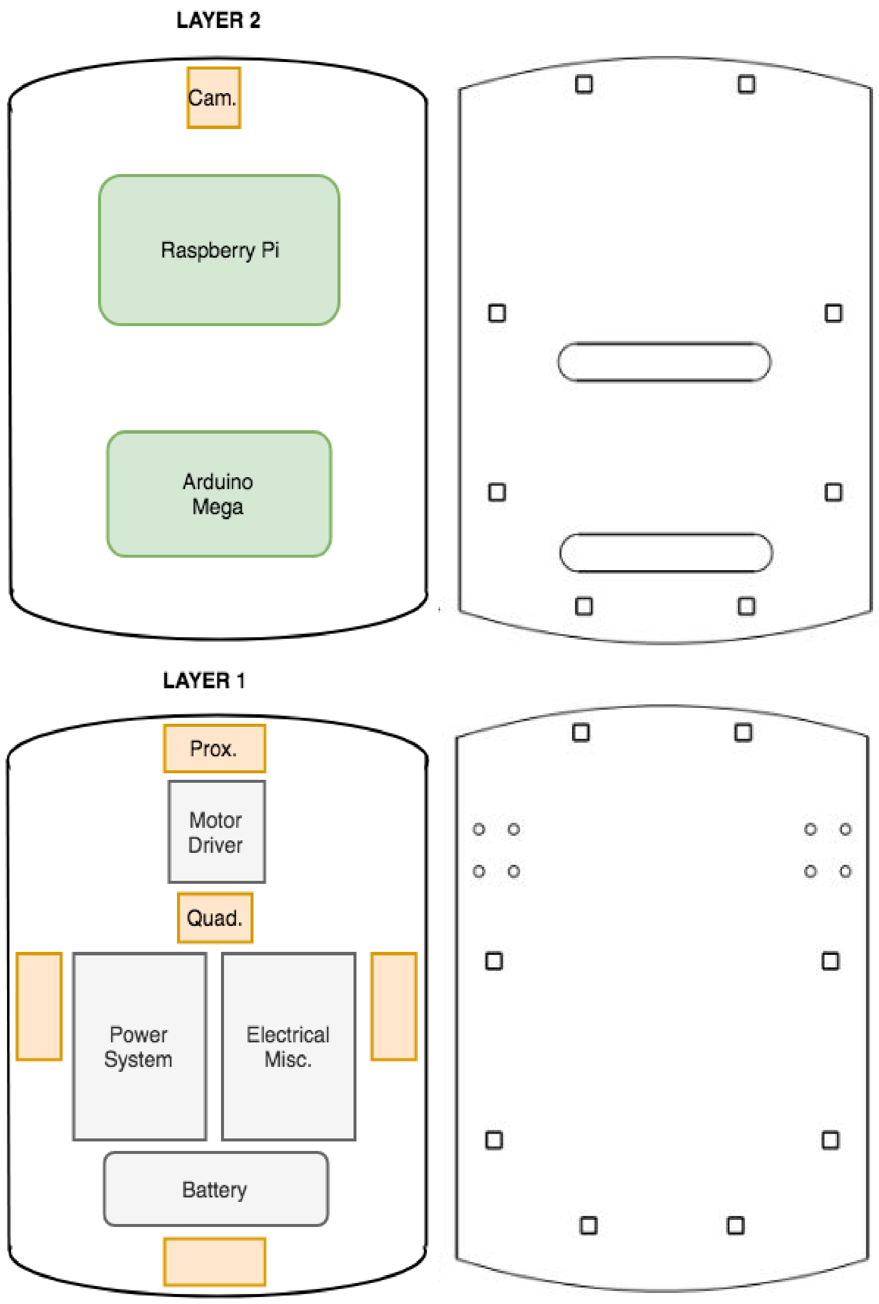


Figure 2.11.b. Layout of Car Layers

Figure 2.11 shows the plastic chassis mounts, the dimensions are 8.75 inches in length and 6 inches in height, with an 2.5 inch gap between the two layers. The car is constructed in various layers, in order to reduce the size dimensions of the car. The lowest layer contains the wheel mounts, motors and wheels. Two fixed wheels were used, which connect to two 12 V DC 600 RPM gear motors with encoders. The wheel mounts are two plastic L-shaped brackets that allow the motors to screw in. The third wheel is a swivel wheel that allows for turning and can rotate 360 degrees.

Layer 1, shown in Figure 2.11.b, includes the sensors and power distribution. The proximity sensors (4) are mounted on each side of the car, held by rubber grippers. The gyro is mounted on a breadboard in the center of the car to provide accurate measurements. The battery is placed towards the back of the car, held down by velcro. The leads of the battery are in a circuit with a twistable switch, which allows for simple on-off operation. The motor driver was mounted with nuts and bolts, to ensure the piece does not move. The electrical components were held down with cotter pins and holes drilled into the acrylic.

Layer 2, shown in Figure 2.11.b, includes the microprocessors (Arduino and Raspberry Pi) and all wiring from sensors. The boards are held in cases made of plastic and acrylic and then mounted with cotter pins. The wires between the components and sensors are feed in large holes that feed to the lower level. The camera module is mounted at the front of the car with a rubber gripper.

The chassis is made of cast acrylic sheets that were laser cut. The holes were designed to accommodate the sizes of the components. Creo 2D was the design software used to generate the cut layouts for the laser cutter. The acrylic layers were held together with screw rods that were fastened on both sides. The rods were then covered in copper pipes and brass caps for aesthetic and safety features.

2. Future Recommendations

Due to the swivel wheel and fixed wheel choice there was unintended directional drift and skidding resulting in errors for the control system. As per the industry panel, there was a recommendation to use a modern steering approach with shifting front wheels and fixed back motor wheels. It is also imperative to add a small fan to deal with cooling issues for the Raspberry Pi to decrease the possibility of shutdown due to heat.

VII. Chapter 3: Android Application (Rich)

A. Introduction

The Android application serves as a front-facing user interface to the system that provides a variety of functions and control to the user.

1. Need

The autonomous car contains many motors, sensors, and a camera that can give the user insight to the status of the system. If any hardware has a malfunction, the user needs a quick and easy-to-use application to interface with the autonomous car system and override its controls in the event of an emergency. This manual override function within the Android app is not the sole function of the Android Application.

After successful connection to the Raspberry Pi, users are given options such as: remotely controlling the car, distance and angle calibration, creating a set of instructions to save to a local SQLite database, view, edit, or insert previous runs to the SQLite database, disconnect from the Raspberry Pi, and view the Senior Project Team's website through the *More* button on the Secondary activity. These use cases provide a lot of functionality and control to the user for the remote car. In theory, this makes the autonomous vehicle more safe by constantly alerting the user in the event of obstacle detection that prevents execution of the desired task set by the user.

B. Android Studio

To design and develop an Android application the Android Studio integrated development environment (IDE) version 3.0 was used. Android Studio is the official IDE for Google's Android operating system. It is mainly built on JetBrains' IntelliJ IDEA software and is specifically designed for Android development. This software came with many useful Android Package Kits (APKs) that are used by the Android operating system for the design, development, distribution, and installation of mobile applications. The APK serves as an executable file that tells the Android operating system how to install the Android application software and what version of APK is needed.

C. Raspberry Pi Client Application

The Raspberry Pi Client Application is the final version of Android based mobile application used to validate communication between the Android mobile phone and the Raspberry Pi. It's main purpose was to provide a neat and easy to use mobile application that gave the user a multitude of functionality such as: connecting to the raspberry pi via socket or http server request, remote control, distance and angle calibration, local storing of sets of user specified instructions to a SQLite database, viewing of stored sets of instructions within the database, running of these stored sets on the remote car, manual override of the remote car with live stream video from the car's camera.

1. Main Activity Layout

The layout of different Android activities are developed through *.xml* files that are used within the Android Studio IDE. These files are deciphered by Android Studio to display a design and blueprint for a unique activity. Figure 3.1 shows the display and blueprint for the *MainActivity.xml* activity for the *Raspberry Pi Client* application*.*

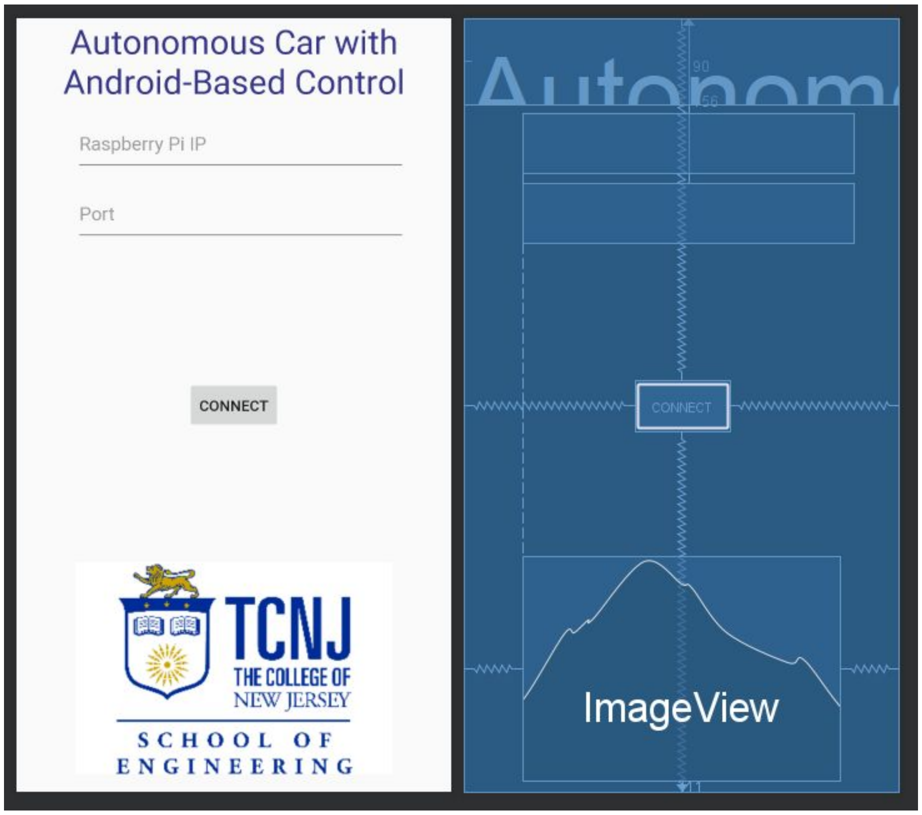


Figure 3.1 *activity_main.xml*

The above layout utilizes EditText fields for the user to input the desired IP address and port number of the remote Raspberry Pi. Additionally, a *connect* Button is conveniently placed below these EditText fields to allow the user to attempt to connect to the remote Raspberry Pi utilizing the text set within the EditText fields.

1.1. Main Activity Backend

The backend to the *activity_main.xml* activity is defined within *MainActivity.java,* shown in Appendix G.1.i. The java file tells the compiler what .xml file to load to the phone's display. Once the *connect* button is clicked, the IP address' format is validated and a valid integer port number. After the parameters are checked, a Client object is created from the given IP address and port number. A Thread object is then made from the Client object and started immediately. This calls the run Override method from within the *Client.java* file. Since the Client class extends the Thread library, a Thread object may be made from a Client object. This Runnable method creates a Socket object and establishes a connection from the Client object's constructed attributes of *ip* and *port*. The Socket object *clientSocket* is then used to generate OutputStreamWriter and InputStreamReader objects of the *clientSocket's* OutputStream and InputStream. These objects are told to utilize a "UTF-8" standard of encoder and decoder. If this is done successfully, and Intent object is created to start the java file *SecondaryActivity.java*.

2. Secondary Activity

The blueprint for the Secondary activity is defined by the *activity_secondary.xml* file which is shown below in Figure 3.2.

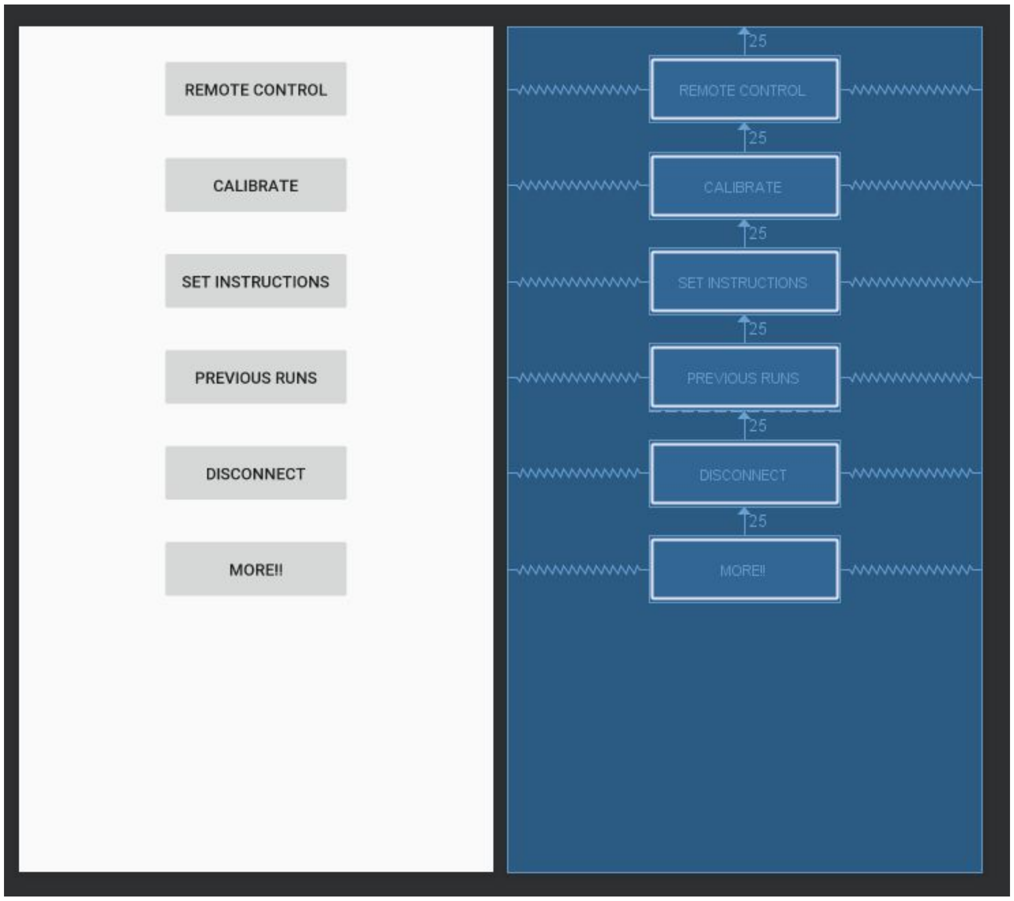


Figure 3.2 *activity_secondary.xml*

The secondary activity gives the user a menu of buttons to allow the user to navigate the different functionality of the application. The buttons for navigation include: remote control, calibration, set instructions, previous runs, disconnect, or view more. The program used to achieve this layout is shown in Appendix G.1.ii. The *SecondaryActivity.java* file is executed and sets the content view to the layout described in *activity_secondary.xml.* This backend java file can be found in Appendix G.1.XI. Once the content view is set, *OnClickListeners* are set for each of the buttons. For all of these *OnClickListeners* different activities are started based off the *Intent* which is dependant on which button was pressed. If the *disconnect* button is pressed, the socket connection is closed and the user is returned to the Main activity.

3. Manual Override Activity

The blueprint for the Manual Override activity is shown in Figure 3.3 below. This layout is positioned as such by the *activity_manual_override.xml* file, which can be found in Appendix G.1.iii.

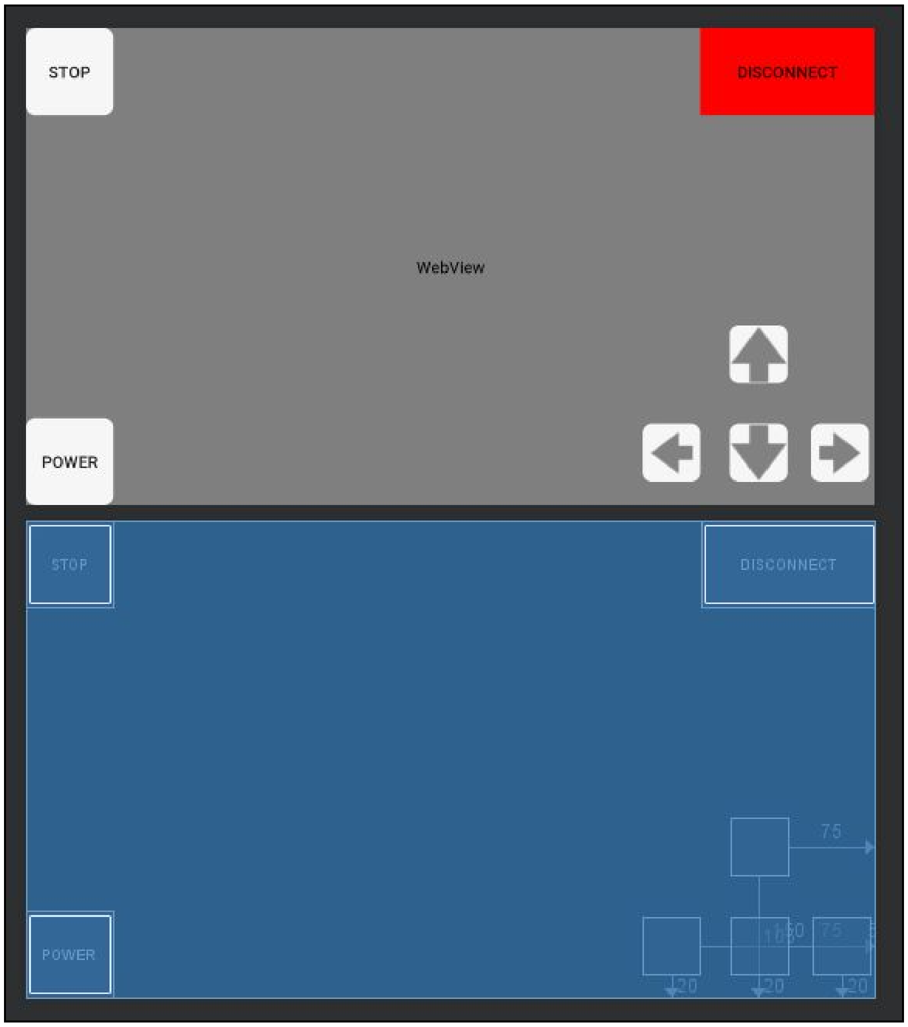


Figure 3.3 *Design and blueprint for activity_manual_override.xml*

The backend for the Manual Override activity is the *ManualOverride.java* file, shown in Appendix G.1.Xii. This java class is created when the *remote* button is pressed the Intent of the ManualOverride class. The *startActivity()* function is utilized with the argument of the Intent object. This runs the *onCreate()* method within the ManualOverride class. This *onCreate()*

method sets the content view to *activity_manual_override.* The first task this class has is to check if the Intent object that was used to start the Manual Override activity. This activity can be started from the *remote* button within the Secondary activity, the Calibrate activity, or the Previous Runs activity after selection of a saved run. If the ManualOverride class is accessed through the *remote* button, then the Intent object will use the function *putExtra()* so that the Manual Override Activity functions accordingly. The *putExtra()* function is used to send long instruction String data types. If the ManualOverride *onCreate()* method is given a String from the *Intent.getExtras()* function call, then it knows it must send that String through the *ObjectOutputStream.* Otherwise, the activity is opened normally and does not attempt to send a set of instructions.

The activity allowed the user to override the controls to the car in the event of an unwanted action.

The arrow buttons allowed the user to override control and send the corresponding instruction based on the pressed arrow button. In the event that the user wants to disconnect they can press the red *disconnect* Button that will send the “dc” instruction to the Raspberry Pi. If the user wishes to stop all car movement, the *stop* Button will be pressed to send the Raspberry Pi the “stop” instruction. The user can also remotely shutdown the Raspberry Pi with the *shutdown* Button. The shutdown button will cause the Raspberry Pi to execute the *sudo shutdown now* command.

3. SQLite Database

SQLite is a relational database management system, which will be used to save the name of a run as well as their associated set of instructions. The setup of this table is shown in Table 3.1.

Table 3.1 SQLite mylist_data Table

mylist_data		
ID	NAME	INSTRUCTIONS
ID INTEGER PRIMARY KEY AUTOINCREMENT	NAME TEXT	INSTRUCTIONS TEXT

The *DatabaseHelper.java* file allowed for the creation of a SQLite database using the *onCreate()* method. This method creates the table shown above. Additionally, this helper class allowed the user to add data, retrieve data, find a specific name of a saved run, replace or update a run, or delete a run using the following methods respectively: *addData(), getData(), findName(), nameReplace(),* and *deleteData().*

4. Calibration Activity

The calibration activity was used to allow for more niche scenarios in which the car may be sent to travel. Through the calibration activity, distance and angle measurements could be preset to allow for the car to travel as desired. The blueprint and layout for the Calibration activity can be found below or in the *activity_calibrate.xml* file.

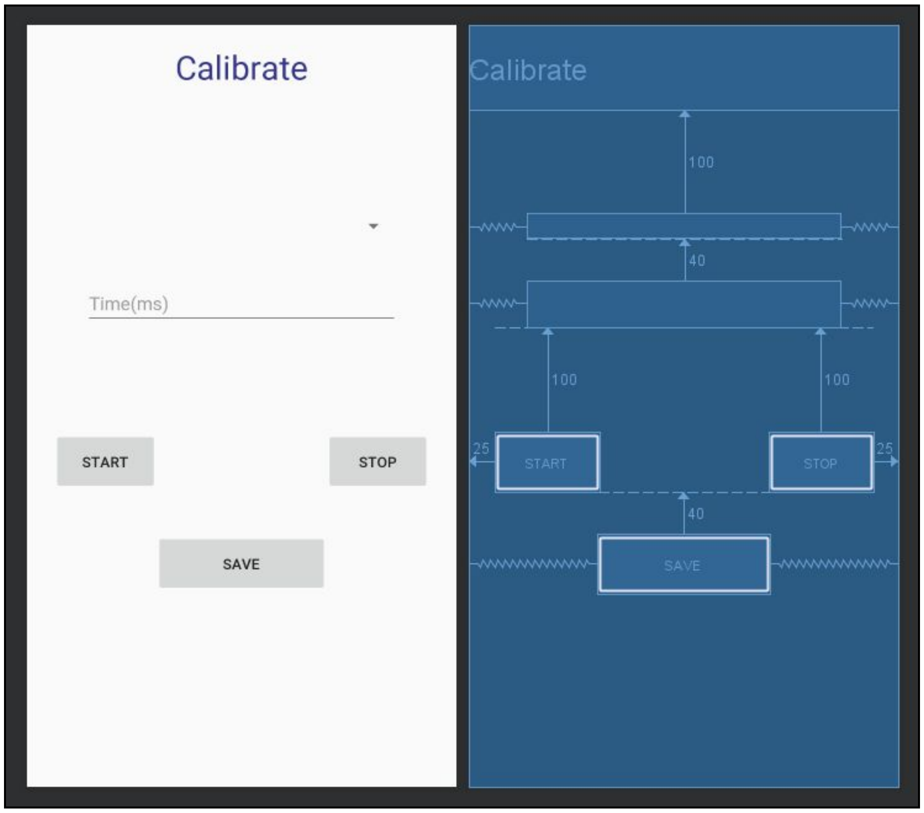


Figure 3.4 *Blueprint and layout for the Calibrate Activity*

The backend for this activity can be found in the *Calibrate.java* file within Appendix G.1.XIV. The user was allowed to select whether to calibrate the angle or the distance using a spinner. From there the user was prompted to enter an integer value to the EditText field positioned next to the spinner. The user could then test their calibration and stop it. Once the desired angle and distance are calibrated, the user must then save them using the *Save* Button. Upon saving, the data for calibration will be saved within the SQLite database with the name *calibration* with the corresponding attribute (distance or angle).

5. Set Instructions Activity

The set instructions activity allowed the user to specify multipliers to the preset calibration values to allow for the user to specify distances and or angles. The spinner consisted of a list of options such as: *stop, forward, backward, right,* and *left.* The blueprint and layout for this activity is shown below in Figure 3.6.

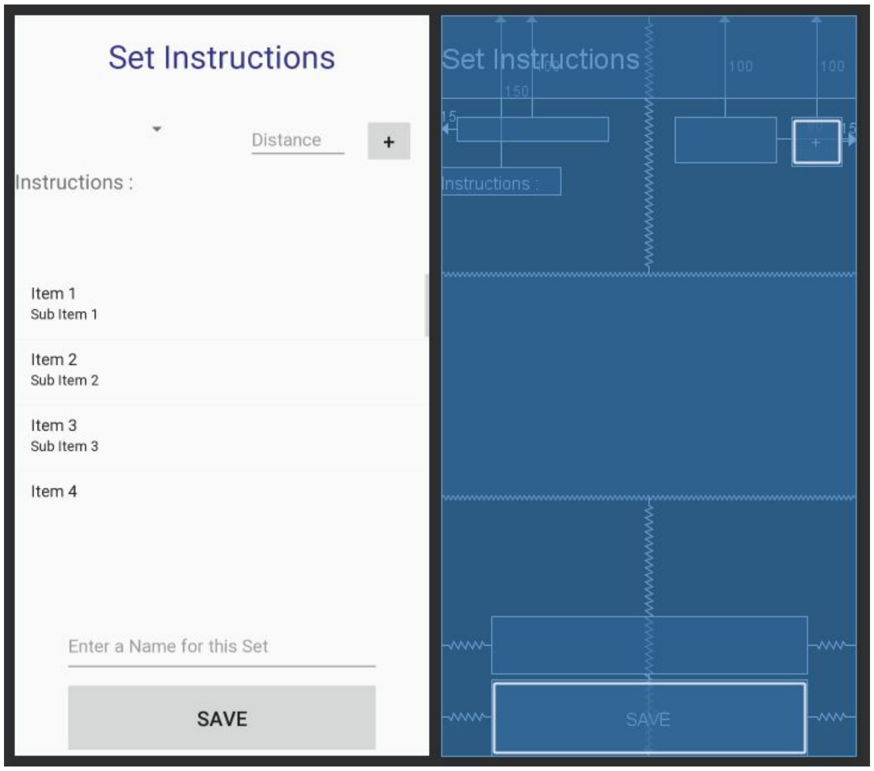


Figure 3.5 *blueprint and layout for activity_set_instructions.xml*

The entered selection would then be presented within a RecyclerAdapter view of a List, to allow for the list to update after entries and show the user what instructions they have already added. Once they have finished adding all their instructions, the user then must enter a name for the entry and press the *save* Button. These runs were then stored within the SQLite database with the corresponding parameters.

6. Previous Runs Activity

The previous runs activity allowed the user to view all saved runs within the database and select one to execute. The user was also able to delete any of these saved runs by pressing and holding one of the run names shown within the list. Once a run has been selected, the user will be sent to the Manual Override activity and will be able to monitor their run execution and override control at any point in time. Below, in Figure 3.6 is the blueprint and layout for the Previous Runs activity.

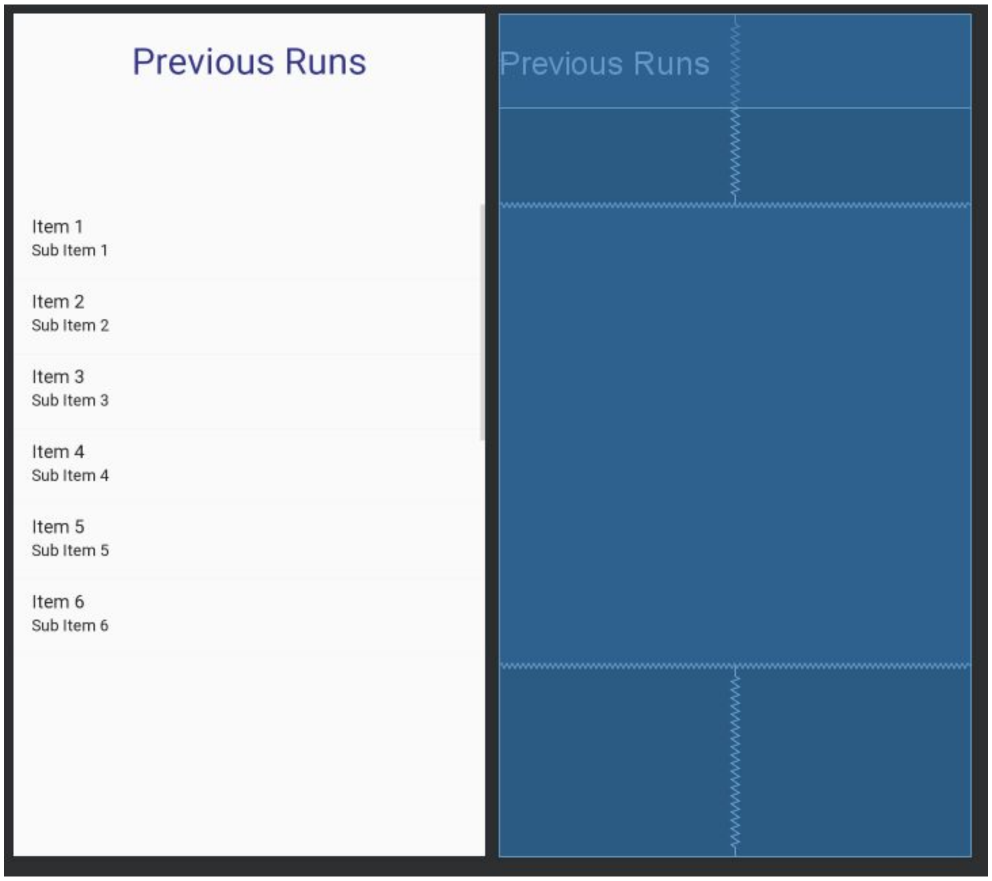


Figure 3.6 *blueprint and layout for activity_previous_runs.xml*

7. More Activity

The More activity would open up the user's default browser on their mobile device and bring the user to the team's website where all project information can be found.

VIII. Chapter 4: The Raspberry Pi (Brendan)

A. Introduction

The Raspberry Pi is the control computer on the car. Using a python script, the raspberry pi will perform the PID control and obstacle avoidance algorithms and send instructions to the Arduino Mega. The raspberry pi also acts as the bridge for a user controlled Android app and the car.

1. Need

The project goals are to create a car that has autonomous capabilities which require control and obstacle avoidance algorithms. The team also wanted to include a live video stream so a user can watch the car on their Android device. The team needed to research different combinations of system on a chips (SoCs) to find the best one to meet the system requirements.

The first board considered were the arduino boards. These boards provide great input and output capabilities and could be programmed with a control algorithm. However, Arduino boards are not powerful enough to perform video streams while controlling the car autonomously.

Another idea was using the Android device's processor and an Arduino, with the Android containing the control algorithms and the Arduino hosting the video stream. The main issue found was that if the Android disconnects unexpectedly this could be dangerous and the car should not be able to roam around without any control. This meant the Android cannot be used to process the control algorithms which means another device must be used in addition to the Arduino.

The Raspberry Pi is a SoC that is powerful enough to host a video stream and control algorithms efficiently while staying under $40. Users can send the Raspberry Pi instructions and then the Raspberry Pi can process the best path to get from its current location to the destination. The Raspberry Pi has built in support for a live stream, giving us the ability to stream to a user. Finally, if the Android device disconnects, the Raspberry Pi could stop the car so it does not harm anyone.

B. Wireless Access Point Setup

It was decided to use a wifi connection to connect the Android and the Raspberry Pi. This provides a secure connection that is faster than bluetooth and be configured to our needs. Using the school's wifi network is an option, however, setting up the Raspberry Pi as the access point is the ideal approach. Using the Raspberry Pi as the wifi host allows us to connect directly to the Raspberry Pi and not use a 3rd party to establish the connection. To do this the HOSTAPD and DNSMASQ libraries were installed.

1. HOSTAPD

HOSTAPD allows a user to configure their device as wireless access point and perform authentication for devices that attempt to connect to the host. HOSTAPD comes with IEEE security protocols and allows the user to configure the network to their liking. For this project, most of the default values that the raspberry pi website recommends were used, and stored that in our HOSTAPD config file. There are a few possible changes. One might be changing the hw_mode from “g” to “a” which will change the speed of the wifi network, this will require additional testing to determine the system tradeoffs. WPA2 is currently the security being used because it has been the best security protocol. WPA3 just recently came out and it is another option as long as it is compatible with the Raspberry Pi. The two components that needed to be setup were the SSIP and the wpa_passphrass. The last thing needed to be done was editing the default config file and add a line that points to our config file.

2. DNSMASQ

DNSMASQ provides a lightweight Domain Naming System (DNS) for small servers like our Raspberry Pi. A DNS is needed because as devices connect to our network the network needs a way to be able to identify them. In our case, DNSMASQ comes with a Dynamic Host Configuration Protocol (DHCP) as well which allows us to statically determine a our Raspberry Pi’s IP address. The last step setting up the network was editing the DHCP config file and define the Raspberry Pi as static with the IP address 192.168.4.1.

C. Python Script

Python is a powerful language and is a the leading edge of software development. Python libraries come with Raspbian, the operating system used on the Raspberry Pi, and is the the language used for this project because it is powerful and works well with Arduino communication. It also comes with socket support which enables communication with an Android device

1. Independent Functionality

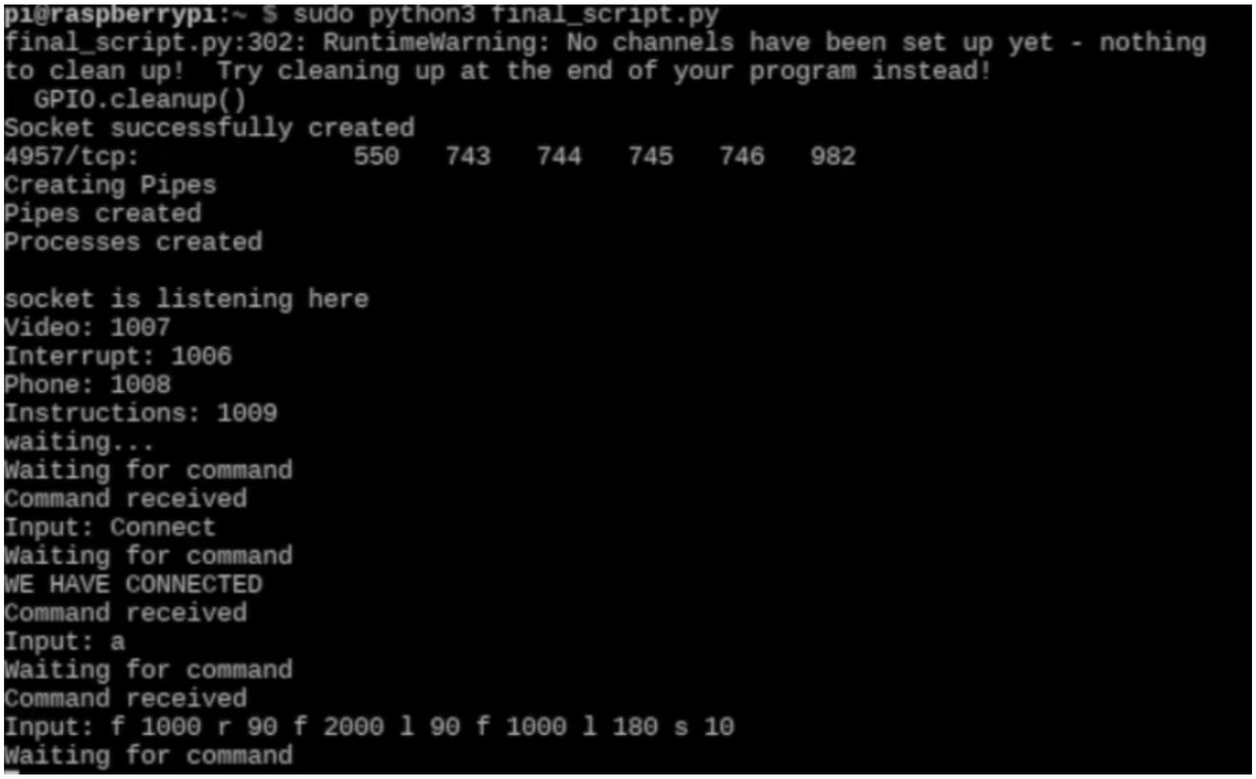
The Python script needed to be validated for its independent functionality before the Arduino and Android devices were connected. The required workflow was a user inputs a command and the script determines if the user wanted to turn the car on, off, left, or right. The user was stuck in an infinite loop that allowed them to input left, right, start or stop and each mapped to their own function. The function then printed out the command that it was told. This verified that the script was able to take inputs and identify which method to use for the commands.

2. Communicating with Arduino

Next the team established a connection between the Arduino and the Raspberry Pi. For this project, the Raspberry Pi needed encode a byte which the Arduino would interpret to do one of four things: Turn both motor drivers on, drive the left motor driver, drive the right motor driver, or turn them off. The python script used python's serial library. The Raspberry Pi's USB port needed to be defined so it can be used to write to the commands sent via the serial connection. The last step in interfacing the two was defining the byte encodings for the Arduino. "L" meant turn left, "R" meant turn right, "F" meant move forward, "S" meant stop. Similar to the independent functionality test, a user was in an infinite loop telling the Raspberry Pi what to do. This time the Raspberry Pi was able to write to the Arduino which would perform the correct motor driver operations.

3. Communicating with Android

To communicate with the Android device the socket library was needed. This allows us to create a socket and read or write to it. For the first part of this project, the Raspberry Pi only needed to be able to read from the socket. Once the data is read from the socket, the python script decodes it from UTF-8 to interpret the message. This was done by sending a string containing either the characters f, b, r, l, or s to indicate forward, back right, left or stop. As soon as a button is released end is sent. Once these interpretations are read in from the socket a conditional is then used to determine which function to use. To test this, the user would select the option from the Android and the script would print to the terminal what was read from the socket and the function it used. An example of the test can be seen in Figure 4.1 below. Here, the mode was set to autonomous and the string sent was f, 1000, r, 90, f, 2000, l, 90, f 1000, l 180, s, 10 meaning go forward for 1 second, turn 90 degrees right, go forward for 2 more seconds, turn left 90 degrees, go forward for 1 second, turn left 180 degrees then stop for a hundredth of a second. Figure 4.2.a shows a user in autonomous mode setting instructions to be sent over to the raspberry pi. Figure 4.2.b shows a user in manual override controlling the car with the arrows.



```
pi@raspberrypi:~ $ sudo python3 final_script.py
final_script.py:302: RuntimeWarning: No channels have been set up yet - nothing
to clean up!  Try cleaning up at the end of your program instead!
  GPIO.cleanup()
Socket successfully created
4957/tcp:              550   743   744   745   746   982
Creating Pipes
Pipes created
Processes created

socket is listening here
Video: 1007
Interrupt: 1006
Phone: 1008
Instructions: 1009
waiting...
Waiting for command
Command received
Input: Connect
Waiting for command
WE HAVE CONNECTED
Command received
Input: a
Waiting for command
Command received
Input: f 1000 r 90 f 2000 l 90 f 1000 l 180 s 10
Waiting for command
```

Figure 4.1: Autonomous Communication Between the Raspberry Pi and Android Device

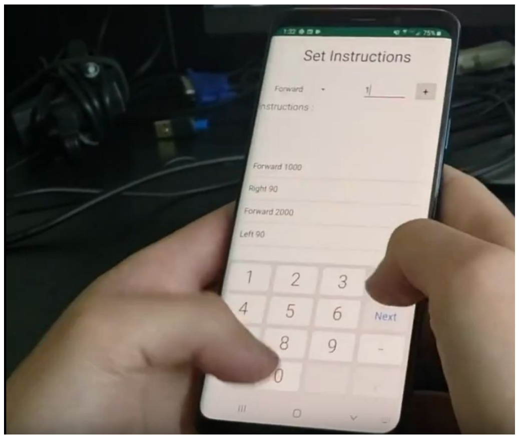


Figure 4.2.a: Autonomous Settings

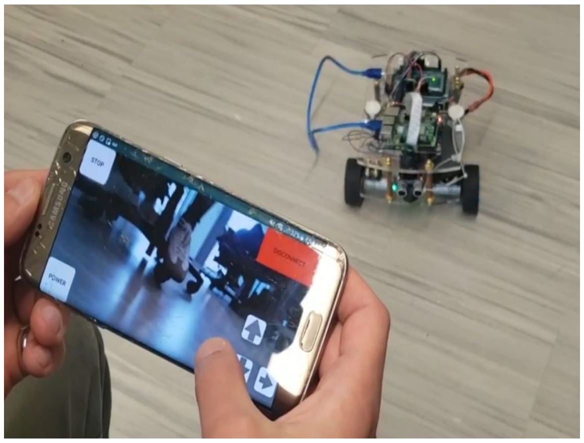


Figure 4.2.b: Manual Override

4. Bridging the Android and Arduino

The last part done for the python script was combining both the Android to Raspberry Pi and Raspberry Pi to Arduino software. This script first waits for a socket connection. Once established it waits to read from the socket. The Android device writes to the socket and then the python script reads this, decodes and interprets the command as turn left, turn right start or stop. It then calls the respective function which translates to telling the Arduino what motor drivers to activate. In this loop if the Android device ever disconnect from the Raspberry Pi, the script goes to a stop the car state and then the script stops execution.

D. Multi Processing

The team quickly found that there was a need to run many tasks in parallel. With the Android device and Raspberry Pi using a socket for communication, the python script stops execution and waits for a message from the Android. Since the second semester tasked the team with adding control and obstacle avoidance algorithms as well as a video stream, we needed the Raspberry Pi to be able to run the other tasks. So in addition to the main process that is run when calling the script, there are 4 other processes created and running: a phone communication process, an interrupt process, an instruction processing process and a video stream process. When the main script is about to finish, it terminates the other processes and cleans them up so they do not become zombie processes in the operating system.

E. Phone communication Process

The phone communication as seen in part 3 above was moved to its own process so that the rest of the script can run on its own. In this process, the message is received from the phone then echoed back so the phone knows the Raspberry Pi received the message. From there, the process passes the message to a conditional and goes to one of three main tests. Either the script passes on the message to the main script to perform operations somewhere else, the script tells the Arduino which drive instructions it wants to use, or the message tells the pi to stop listening and power off. To communicate back to the main process, a pipe communication is used. To prevent the main script from waiting to receive the message from the phone communication process, a GPIO was used so the Raspberry Pi could poll when data is ready to be read from the phone communication process.

F. Interrupt Process

An interrupt process was added to the python script so that the arduino would not have to use a sleep or delay function in its main loop. Since python supports multiprocessing, it makes sense to use that to send an interrupt to the Arduino.   The interrupt was needed because the pulling sensor data needed to be done every tenth of a second. Once a phone connection has been established, the process tells the Raspberry Pi to write “HIGH” to a GPIO pin for 5 hundreths of a second then write “LOW” to the same GPIO pin. This pin is connected to the external interrupt pin on the Arduino.

G. Instruction Process

The instruction process takes a string of commands and breaks the string into a list. The first item in the list is the direction, and the second is the time it is running for. While there are items in the list, this process will tell the Arduino what to do. This needs to be in a separate process so that if the Android sends a stop signal or anything else from manual override, the car stops executing the rest of the instructions and listens to the user. This process runs forever so at any point during the cars lifetime an instruction string can be sent and executed.

H. Video Stream

To implement a video stream, the team purchased a camera dongle which was built for the raspberry pi. The stream used Pythons HTTP Server which started a web server that the phone could access. In that separate script, a server object was used to send data to the web server and in this the camera module was called to collect mJPEGs at 10 frames a second. The main script creates another process and when a connection is established that process runs the stream script.

I. Changing Run Time

Once establishing that the script works to our needs, the team wanted to make the system more user friendly. The first step was making it always run since the script stopped after the Android device disconnected. A user should not have to start the script every time they go to use it. So when a phone disconnects, instead of stopping the script, it goes back into a listening state where it waits for a phone connection. The last step was making the script run as soon as the Raspberry Pi was turned on. At first the team tried to create a service to start the script. This was done unsuccessfully, so the next idea was making it run when the terminal is opened. This worked however a user would still need to open the terminal to run the script. This meant finding a way to run the terminal as soon as the Raspberry Pi turned on. To do this a shortcut for the terminal was created and stored it in an autoStart folder in the Raspberry Pi's .config folder. When the Raspberry Pi was turned on, the script broke. The issue was that the socket requires the Raspberry Pi's IP address and port number. The IP address was either in use or not defined as soon as the socket tried to use them, so there needed to be a check to see if they were in use and if so, wait until they could be reused. Once adding that in, the script was able to run when the Raspberry Pi booted up and connect to the Android device. The last thing added was a log file that time stamped each operation performed so after several runs the team can debug any issues that occured. The state diagram can be seen below in Figure 4.3.

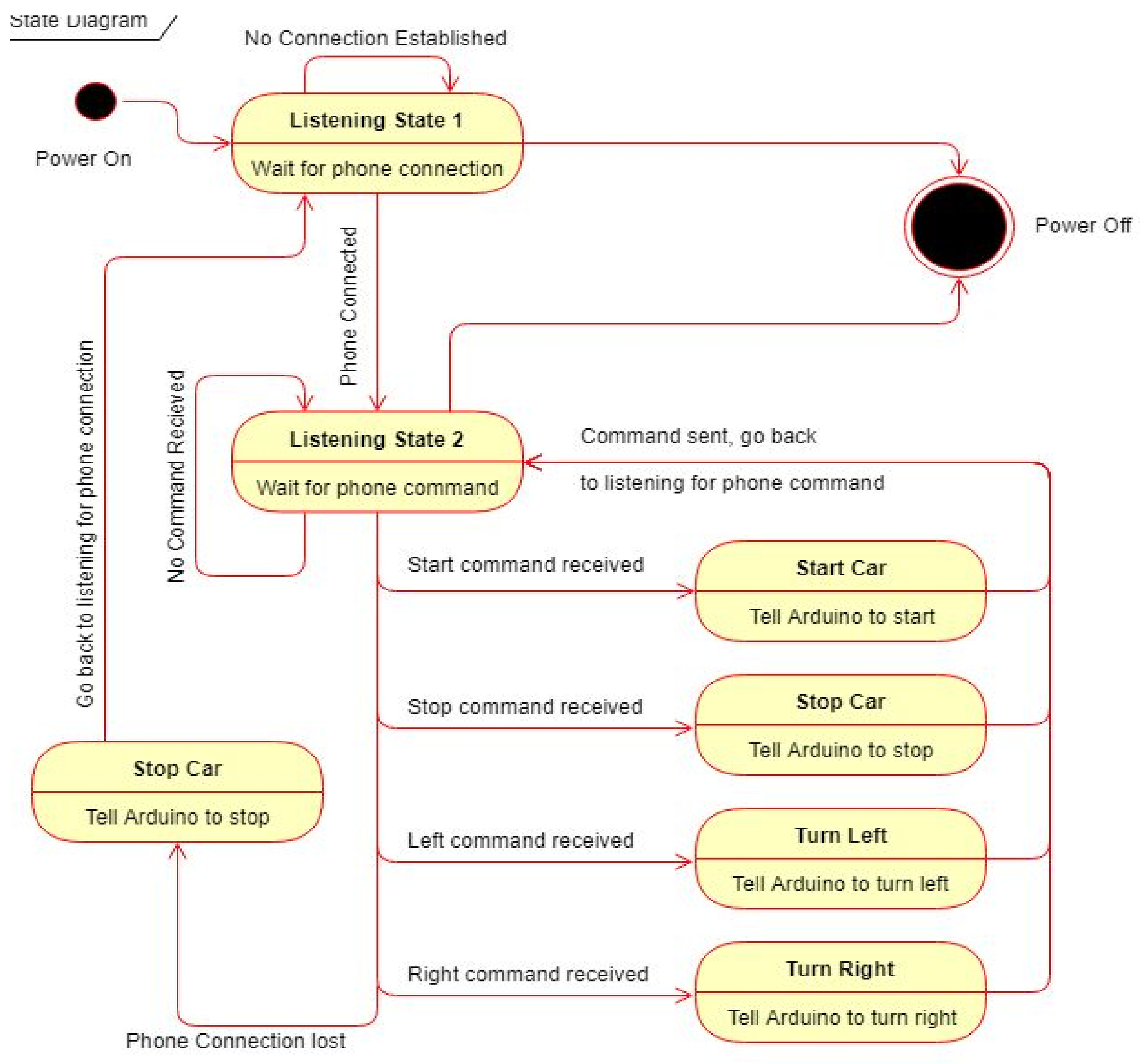


Figure 4.3: State Diagram of Raspberry Pi Python Script

IX. Chapter 5: Arduino Subsystem (Terry)

A. Introduction

An important part in making the car autonomous is the suite of sensors that read in the necessary information in order for the car to make decisions. In parallel to this, a way to control the motors and turning for the car was also required. This subsystem sits at the opposite end of the chain of connections from the phone and acts as the bride between the hardware and software systems.

1. Need

A decision was made to use the Arduino Mega 2560 as the microprocessor that controls the sensor subsystem. There were a number of differing factors that went into this decision, but overall it makes the entire system more robust. The car required a number of different sensors in order to sufficiently control the car. In addition, control signals to the motor driver were required in order to drive and steer the car. Although the control and obstacle avoidance algorithm will both operate on the Raspberry Pi, it was decided that the arduino would still be a necessary addition to the system.

The first major reason to include the arduino was the desire to incorporate a camera that could stream video of the car's current position to the user on their phone. This necessitated the introduction of the Raspberry Pi into the system because Arduinos don't have the capability to stream video over Wi-Fi to a cell phone. In addition the GPIO pins on the Pi are not electrically protected and can be harder to use than those on the Arduino. Also this would limit the use of a case over on the Pi and the number of GPIO pins may have been insufficient. The final deciding factor was that there are already a number of different APIs written to interface with the different sensors required for the car. While it would have been interesting to manually write them, a decision was made that it would be an unnecessary addition to the project, and that the Arduino would be sufficient.

B. Hardware and Sensors

In order to properly control the car, the control system needs some sort of inputs in order to decide how much power to give to the different motors. The control system also requires these sensor readings in order to make the small adjustments in driving direction, speed and relative angle of the car. In addition the obstacle avoidance algorithm also requires some set of sensors in order to determine where an obstacle is, and where free paths exist around it. Research was done on the different types of sensors and how to implement their readings within the control system.

1. Ultrasonic Sensors

In order to properly make the car autonomous, a distance sensor was required to alert the algorithm about any objects nearby. Although distance measurements won't directly be used in the control system, they are needed for the obstacle avoidance system. The Adafruit HC-SR04 Ultrasonic Sonar Distance Sensor was chosen for this task. Four of these sensors will be placed around the car in order to detect obstacles and report the cars relative distance to any walls nearby. Each has a total of four different pins that need to be connected to the Arduino in order to be powered and send the appropriate signals. An image of the sensor can be seen below in Figure 5.1.

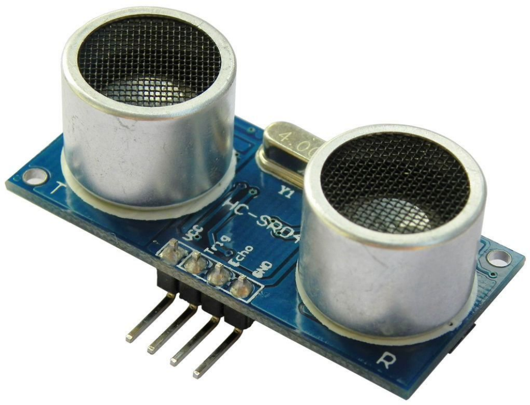


Figure 5.1 Adafruit Ultrasonic Sensor

The sensors operate by sending out a signal from the Arduino to the Trig pin on the sensor. They then read the back the echo on the echo pin and a duration measurement is recorded. This value can then be converted to how far away the object was that reflected the signal. This was all accomplished in code and tested in an Arduino script. The sensors also on require a 5 volt supply and only draw 15mA apiece.

2. Quad Sensor

The other important sensors that were required to automate the car were a gyroscope and accelerometer in order to record both relative rotation and acceleration of the car. The main operation of the control system will be based off of these sensor readings for adjusting the speed and angle of the car. When looking through the different options the Adafruit 9-DOF Breakout Board - LSM9DS1, pictured below, was chosen and will be referred to as the quad sensor. The quad sensor consists of four different sensors, an accelerometer, gyroscope, magnetometer, and temperature sensor. The gyroscope will allow the car to set a reference direction and adjust for any small changes in direction. The accelerometer can measure changes in its acceleration. The magnetometer can be used to figure out true direction by using the earth’s magnetic poles. All these sensors will be used in tandem in order to send information to both the control system and the obstacle avoidance algorithm

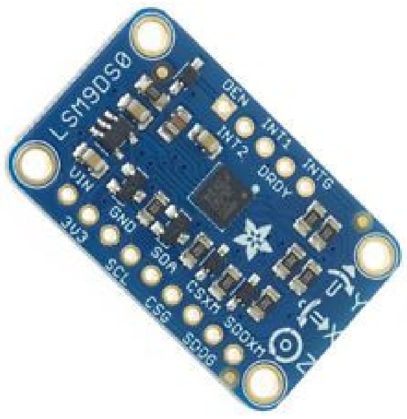


Figure 5.2 Adafruit Quad Sensor

3. Motor Encoder

An additional sensor was implemented later on into the build. Each of the two front wheels had a motor encoder attached to them which used hall effect sensors in order to determine the speed of each of the wheels. This data was important for use in the control system for the car which will be discussed later. The way the hall sensors work is by reading in the magnetic field produced by the motor in two different places. This produces two out of phase square waves which can be then used to determine the exact position of the wheel in the motor. Each of these individual rotations can be counted up and compared with the amount of time passed in order to determine the speed of the wheels.

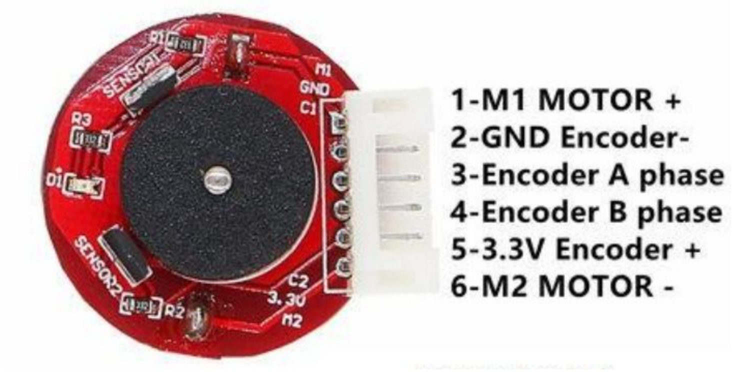


Figure 5.3 Motor Encoder

4. Motor Driver

An L298N motor driver was used in order to control the speeds of the motors on the car. Motor drivers work by using an H-bridge circuit to control the direction of current across the motor. The L298N motor drive is a dual H-bridge board and can power up to two motors. The board can handle up to 46 volts, so the 11.1 volt battery is perfectly usable with the board. Motor drivers function by taking in a PWM signal in order to control the speed at which they run the motors at. This means, on the Arduino’s side, the PWM output pins are required and the higher the duty cycle, the faster the motors will turn. The Arduino is capable of outputting 256 different PWM sizes, and is capable of driving the motors at necessary speeds.

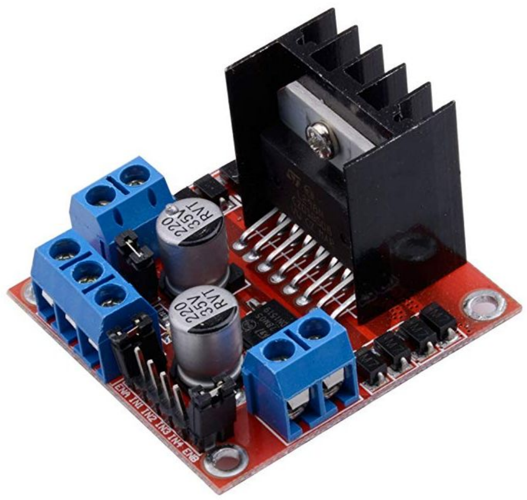


Figure 5.4 Motor Driver

C. Software

In order to interface the Arduino with the different sensors, code had to be and uploaded to the arduino using Arduino sketches. The Arduino software was download on the Raspberry Pi in order to upload the code from the Pi straight to the arduino using the serial to USB cable between them. This cable also conveniently provides power to the Arduino and is what is used for the serial transmission from the Pi to Arduino. Once the code is written and verified, it can be uploaded onto the Arduino flash memory and will then run on startup.

When writing code for the Arduino microprocessors a number of things should be kept in mind. Arduino code blocks are broken up into three main sections each with its own purpose. In the beginning of the code block, variables and Pins can be defined to be used later. It is convenient to define the different used pins as their own variable names in order to keep code readable. The second section is the setup portion of the code where the serial data transmission is initialized at a specified baud rate so that the arduino can print out data to the console. This same serial transmission is what is used in order to the Arduino to communicate with the Pi. In addition other important tasks like setting the pin mode for the Arduino’s GPIO pins can be set up. The last block consists of an infinite loop which will continue to run the code while the arduino is turned on. All of its reading and writing to GPIO pins is done in this section, as well as the logic required to send data serialy.

D. Communication

An important detail of the car operation is how the different sensor data is read and sent around to the appropriate devices for computation. The Arduino its in a position where it has to read in the sensor data from multiple sensor, send data to the Raspberry Pi, receive data from the Pi and send signals to the motor drivers and any additional hardware placed on the car. Most of the means of communication is through serial data transmission, and a number of different protocols are used in different areas of the car. The arduino uses the I2C serial protocol to communicate with the quad sensor which allows it to read in a large number of different data streams from the same sensor. The Arduino then uses a USB connection to the Raspberry Pi in order to send and receive data and instructions to it. The general flow of information between the Arduino and Pi is shown below in Figure 5.3. A more comprehensive overview of these connections will be discussed in the following sections.

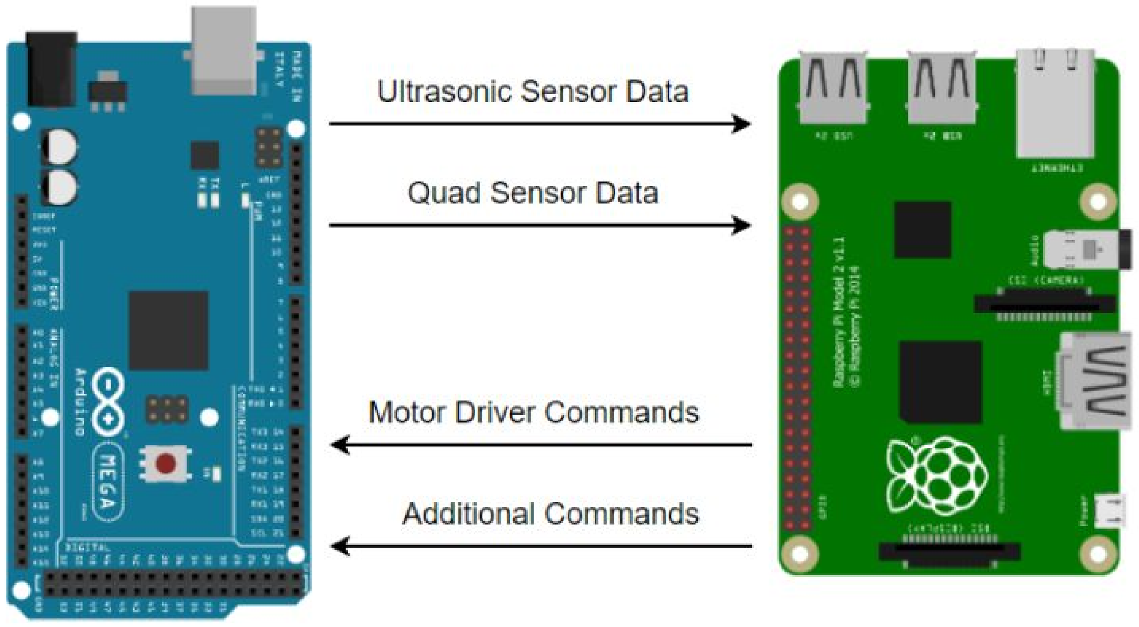


Figure 5.5 Communications between Raspberry Pi and Arduino

1. Sensors to Arduino

This section will go over how the different sensors were implemented and connected to the Arduino. The currently configuration is a preliminary design that was set up for the Senior Project Presentation demo, but the concepts will remain the same for the functioning car.

i. Ultrasonic Sensors

The first sensors connected to the Arduino were the Ultrasonic sensors discussed earlier. Their implementation was mostly straight forward as they don't require any specific transmission protocol. Rather, analog signals are sent and received by the Arduino in order to get the appropriate readings. First, the test configuration shown in Figure 5.4 below was set up in order to test and calibrate the sensors. Next, two analog connections are made from the Arduino to the sensors. The trig pin is sent a 10 microsecond pulse in order to tell the sensor to start making a measurement. Eight pulses are then sent out from the sensor and the value at the echo pin is read in. The value received is a time measurement in the order of 100s of microsections. This duration measurement can be converted to a centimeter length by dividing it

58. Now that the measurement is in an understandable unit, its value can be used to make decisions about what the sensors are sensing. The code for this test can be found in appendix G.3.i. Since these sensors operate directly with the analog GPIO pins, any number of them can be configured to run at once. A total of four will be used in the final design of the car.

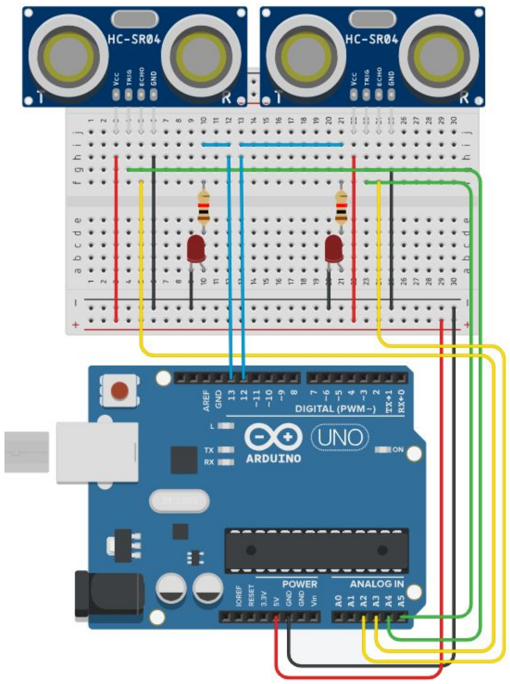


Figure 5.6 Ultrasonic Sensor Test Configuration

ii. Quad Sensor

The other sensor that had to be interfaced with the Arduino was the quad Sensor. Unlike the ultrasonic sensor, the quad sensor is an entire chip that can communicate over I2C. Although this protocol can be somewhat tricky to use, the sensor came with a library that had already configured the protocol. This meant setting up the sensor was a relatively easy task. Only the power, ground, SCL and SDA pins had to be connected. From here the example code based off of the library was modified in order to test the different operations of the quad sensor. The code used to test can be seen in in appendix G.3.ii. After testing to make sure the functionality was as expected, the code was removed from the Arduino so that other testing could be done.

iii. Motor Encoders

The third set of sensors that were used for the car were motor encoders attached to each one of the wheels. In order to read these sensors, each of the phase pins had to be attached to an interrupt pin on the arduino in order to read them in real time. In addition a timer was used on the Arduino to count up in milliseconds from when it was initiated. Then, using an Encoder library, the two phase pins from each of the seniors were read in every 100ms and converted into speed by taking the difference in encoder value. The motor encoders were tested to make sure the accurately read in clockwise and counterclockwise rotations of the wheel.

2. Raspberry Pi to Arduino

The other important aspect of communication was between the Arduino and the Raspberry Pi. This was carried out by making a USB serial connection between the two devices. A setting to allow serial communication had to be altered on the Pi and then additional code on both ends had to be written to send information. First, on the Pi's end, a basic python script had to be set up in order to initialize serial communication. The port to communicate with the arduino was either ttyACM0 or ttyACM1 and had to be adjusted depending on where the serial connection was recognized. There are future plans to automate this selection process when the Pi boots up. After the connection was established, data could be sent serially using the ser.write() command. The implementation for sending strings of instructions can be seen in appendix G.2.i.

On the Arduino side, a script had to be written in order to read incoming serial data. A new function was written to test if there was any new serial data being sent to the Arduino. In the case there was, the function would then start reading in the data and save it to a string in order to use it later in the code. The implementation of this for the Senior Project 1 demo can be seen in appendix G.2.iii.

E. Obstacle Avoidance

Although a number of different obstacle avoidance algorithms were explored, none were officially implemented on the car due to the lack of time. In order to account for this lack of obstacle avoidance, a simple version was written in order to make sure the car would not run into obstacles. This was done by reading in all the front, back, left, and right ultrasonic sensors every 100 ms. From here the Arduino would check if the distance reading was below the threshold. If it was, the car would stop driving, and control was handed over to the operator on the phone. From there, they could navigate the car away from the obstacle and then turn back on autonomous mode if desired.

F. Control System

A basic discrete proportional integral (PI) control system was implemented on the arduino in order to control the functions of the wheels. Since both motors weren't identical, they each drove a different speed with the same PWM input into the motor driver. This meant that a control system was required in order to drive both motors are the same speed. PI control is fairly simple to implement in code. As shown in Figure 5.5, a set speed was chosen, this was the desired speed for the motor encoder to read in. The current speed of the motor was subtracted in order to calculate the error which got fed into the controller. The integral part for a discrete PI controller consisted as a running sum of all the errors from each iteration. Both a value for Kp and Ki were chosen in order to tune the controller. Finally the controller would send a specific

PWM value to the motor driver to update the speed. Overall the control system worked as intended, but closing the loop around the gyro reading in addition would have benefited steering.

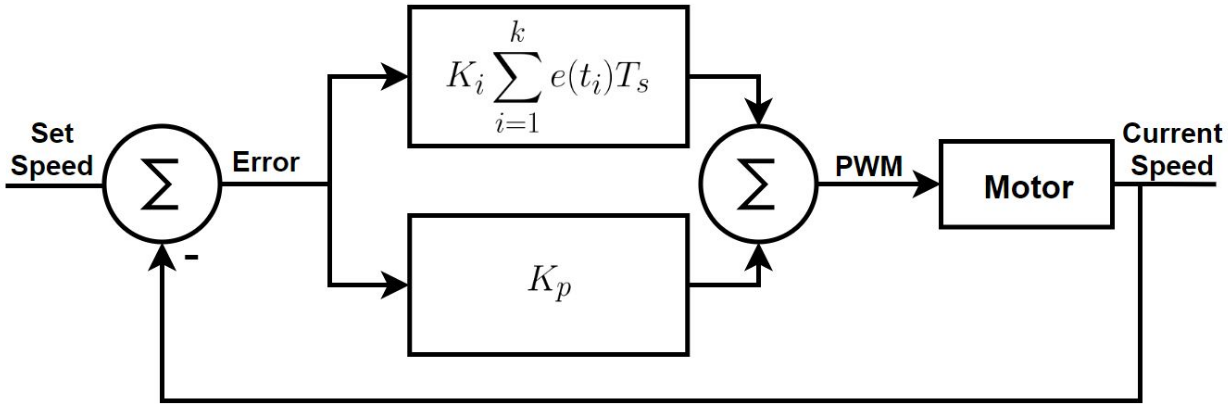


Figure 5.7 PI Control System

G. Conclusion

Overall, the Arduino acts as the middleman between the hardware and software on the car. It is an efficient way to deal with a large number of GPIO peripherals, and already has prebuilt libraries for some of the sensors being used. At the same time it is able to efficiently communicate with the Raspberry Pi in order to send and receive information regarding the control of the car.

X. Chapter 6: Integration, Testing and Validation (Madison)

A. Integration and Testing

The system was integrated via a series of workflows and areas; hardware and software. Simultaneously, the hardware chassis was constructed and components were placed and mounted, while the software applications were integrated. Once the hardware portion was completed, the hardware and microprocessors were wired together, which includes the serial connection between the Raspberry Pi and Arduino Mega. The system was then able to interface hardware and software. The wiring schema and power system was tested by supplying simple code to move the motors via Arduino, which using a NI box. Next, the software workflow was tested virtually to detect listening between the Raspberry Pi and application, which includes the initial set modes autonomous or manual (i.e. ‘a’ or ‘m’), list of instructions, or singular instructions with breaks to indicate the sent instruction is completed on the software workflow.

Once both hardware and software were separately implemented, they were integrated together by testing each step in communication and adding additional modules (other components) until the full workflow was completed. The entire programming and hardware system was testing by relaying instructions to the phone which were used to power the motors and drive the car. This workflow test acted as the backbone process behind manual override, which utilized interrupts to break the queue of pre-set instructions used in “Autonomous” mode.

B. Validation

In order to validate the system, functionality was checked for various phone support and data management of set instructions. Various adaptation were tested, such as implementing situation-based interrupts. For example, a threshold value was set for the ultrasonic sensors, which resulted in an interrupt sent to the Raspberry Pi and phone. This resulted in the car overriding the user with a forced stop for 5 seconds upon obstacle detection.

In terms of specifications, the categories were verified individually. The power system was testing by verify each component for voltage and current using the NI box. All specifications were met including a 30-minute battery life, which was far surpassed. A thorough depletion test was not done, however, after over six hours of use the voltage of the LiPo battery dropped from 12.1 V to 11.3 V. In terms of the car category, the dimensions of the car were smaller than originally set, which was a desired outcome. The max speed of 1 meter per second surpassed the previous set value is 1 foot per second. In terms of the communication latencies, all specifications were met, the latency of phone to Raspberry Pi was 0.5 seconds, which satisfies the less than 1 second delay. The baud rate was set due to the gyroscope. In terms of video streaming, 480p satisfies the initial set resolution of greater than or equal to 480p. The frame rate was previously set to 25, however, the current setting is 15 frames per second. This can be change but will result in potentially higher latency. The frame rate was the only specification not met, but has the ability to be changed.

XI. Chapter 7: Conclusion

A. Complications

The team faced many problems this semester with the setup of the Raspberry Pi. One of the Team’s Wi-Fi chips was not functioning properly and there was a period of time where the team struggled with the setup of the Raspberry Pi as a WAP. Additionally, towards the end of Senior Project I the team was set behind on the soldering of components for the power system due to not all of the parts arriving on time. Within the Android application, there were difficulties with the setup of the *Socket* and the *ObjectOutputStream* object.

When there was struggle with the setup of the WAP, another team member was assigned to the task to aid in it’s development to prevent further setback. In the case of the power system components, luckily they arrived prior to the end of Senior Project I and the soldering of components was still completed as planned. The issues within the Android application were solved after extensive debugging methods done within the *Logcat* debugger view in Android Studio.

During the Serial connection, the communication relied on actual values to be sent over. A complication arose when sensors could not pick up any data and would attempt to write 0 to the serial. This would send nothing over to the Raspberry Pi and would result in an unreliable message from the Arduino. A step taken to try to overcome this was set the minimum to 5 so that the Arduino would always write something to the serial. However, the Arduino will still skip some values therefore the Raspberry Pi could not reliably read from the serial connection. Another issue that arose during the Raspberry Pi and Arduino communication was timing. The Raspberry Pi would attempt to interrupt the Arduino every tenth of a second, however the Arduino would occasionally be in a system sleep to read data from the sensors. This is a potential cause for the serial write issues stated above.

An issue with our video stream is that it ate up processing power due to the way it streamed the video feed. This took away some other processing capabilities and introduced latency to the rest of our system. This caused some responsive issues during testing.

Tuning the Raspberry Pi in different environments also proved to cause issues since the friction coefficients for the different surfaces on which the car could drive on would have slight effects on the control system, causing it to overcompensate in some environments. Multiprocessing on the Android Application also caused issues since the latency in writing to serial would cause the Raspberry pi to not read the sensors as quickly as we would have hoped.

Cooling the Raspberry Pi also caused issues with testing. Since the Raspberry Pi only has heat syncs, upon testing for hours we would find the Pi would slow down significantly due to the high temperatures of the controller. This would cause the latency to fluctuate unexpectedly and the car would vary in latencies.

B. Future Recommendations

The recommendations for future development include: creating an axel for the car, adding more weight and traction, a cooling system, an fully implemented obstacle avoidance algorithm, changing how the sensor data is read, changing the video stream, adding server capabilities. Adding an axel and weight will allow the car to be tuned more accurately and will lead to better steering. A cooling system should also be added to ensure neither the Raspberry Pi nor the Arduino will overheat in a heated environment. As of now the sensors are connected to the arduino which needs to send the information back to the Raspberry Pi. Looking into a method that allows the Raspberry Pi to read the sensor data will take out a step in the process of reading in the sensor data. The Raspberry Pi also writes mJPEG images to a web server which is very inefficient. There is a method of streaming bits to a socket which can be interpreted on the Android as a BitMap. This should be further investigated to see if performance is improved. Adding server socket capabilities will allow multiple devices to connect to the python script. As of now only one phone has the ability to do so, and making the script the ability to listen to multiple devices will make it more robust.

C. Senior Project Accomplishments

The team was able to successfully design and develop a power system, Android application, Raspberry Pi wireless access point, and Arduino hub for sensor data analysis as well as establish and verify communication across the three platforms. The team was able to create a video stream that recorded video from the Raspberry Pi and was able to broadcast to an Android device. The Android device was able to save data to a SQLite database enabling a user to re-run instructions from previous runs. With strong communication, thorough planning, and positive mentality the team was able to stay on track with the project schedule and overcome the many obstacles that were encountered.

XII. List of References

Malinen, J. (2013). *hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. [online] W1.fi. Available at: https://w1.fi/hostapd/ [Accessed 6 Dec. 2018].

https://www.eia.gov/analysis/studies/transportation/automated/pdf/automated_vehicles.pdf

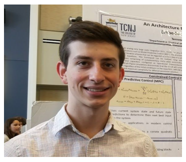
XIII. Appendices

A. Team Biography

Brendan Behrens is a senior Computer Engineering major at The College of New Jersey. His current interests are in data science and security as well as software development and embedded systems. After graduation he plans to work on the STEALTHbits Technologies team as a Junior Developer.

Richard Bustamante is a senior Computer Engineering major at The College of New Jersey. He is the current IEEE student chapter treasurer. His current interests include mobile app development, data science, and software development. After graduation he plans to work in app development.

Madison Mastroberte is a senior Electrical Engineering major at The College of New Jersey. She is the current IEEE student chapter vice-president. Her current interests are data science and machine learning, and she plans to obtain a Master's degree. After graduation, she will be joining Bloomberg L.P. as a technical data analyst on the Alternative Data team.



Terrence Skibik is a senior Electrical Engineering major at The College of New Jersey. He currently does control systems research with Dr. Adegbege in the LECO lab. He is the current IEEE student chapter president and enjoys being involved in the department. His current interests include control systems, optimization, and machine learning. After graduation he will be attending the University of Colorado Boulder to pursue a Ph.D. in control and dynamical systems.

B. Gantt Chart

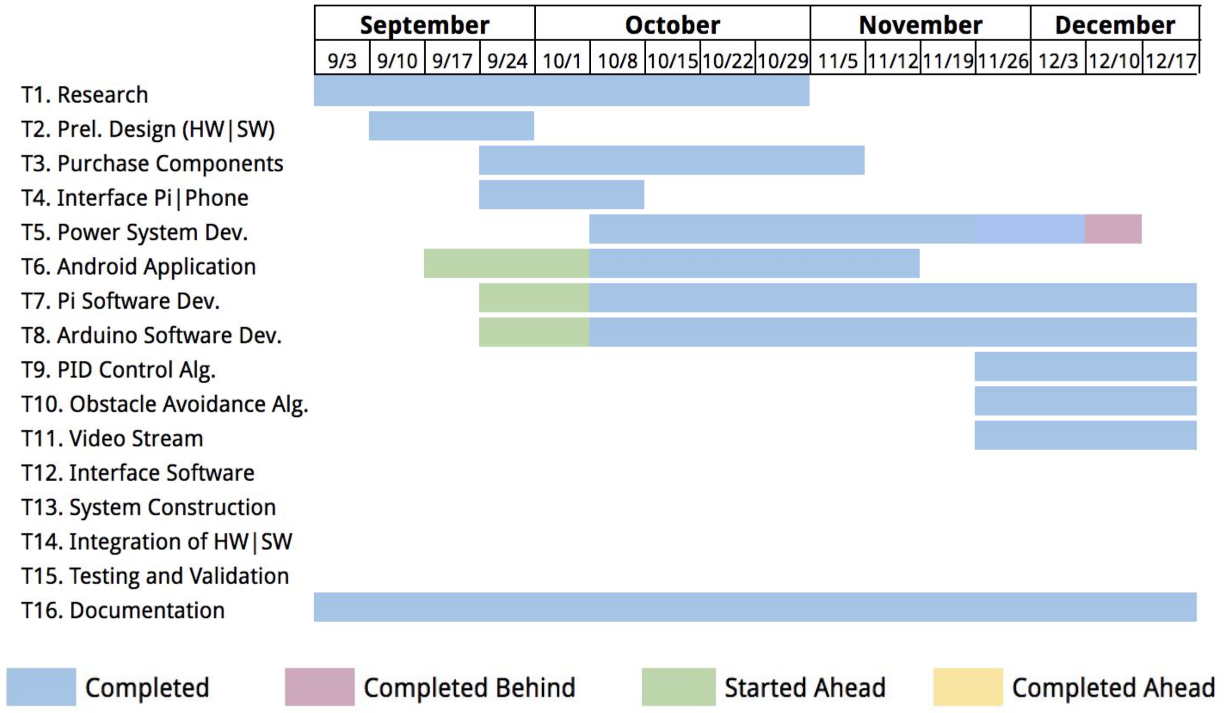


Figure B.1 Gantt Chart for Senior Project 1

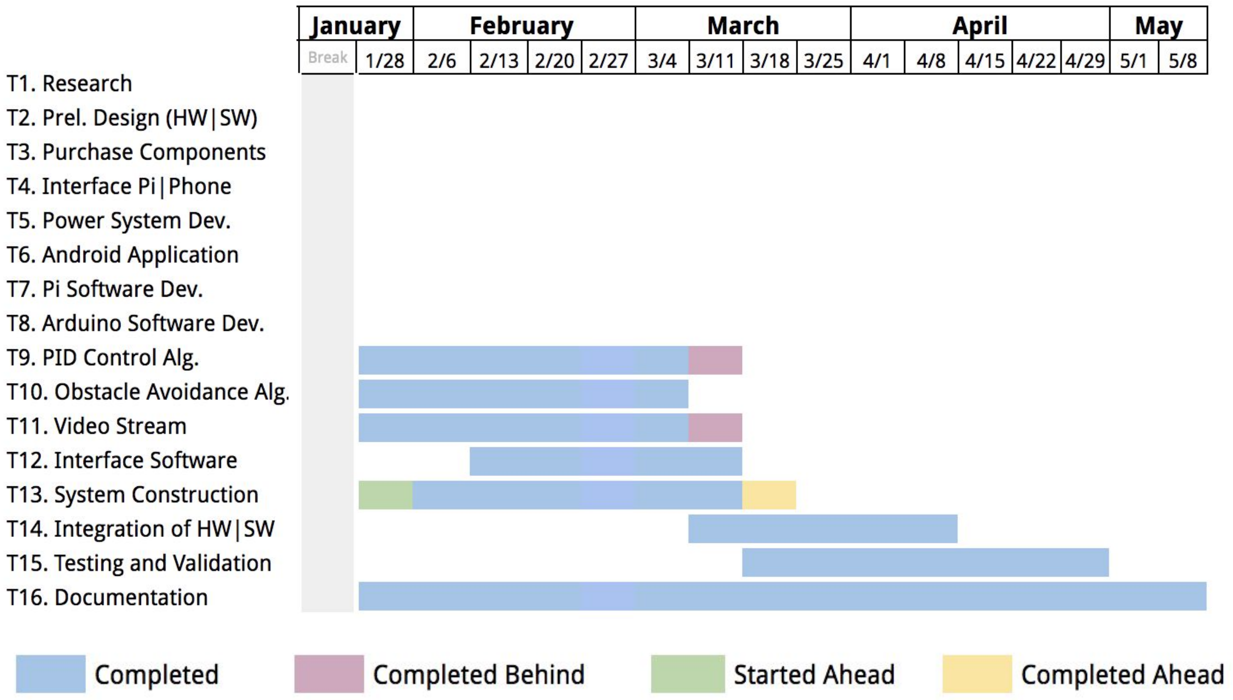


Figure B.2 Gantt Chart for Senior Project 2

C. Financial Budget

Purpose	**Component**	**Units**	**Cost**
Car Construction	Chassis	1	$40.00
	Motor (2)	2	$32.90
	Motor Driver (2)	2	$15.00
	Wheels (4)	4	$26.00
Power System	Battery (12V)	1	$29.99
	IC Chips	1	$11.64
	Resistors	1	$3.70
Software Components	Samsung S9+	1	$917.00
	Raspberry Pi	1	$39.95
	WiFi Card (Pi)	1	$8.01
	Arduino Uno	1	$24.99
Misc.	Video Camera	1	$25.58
	Gyro Sensor	1	$14.95
	Proximity Sensor	3	$29.25
	Accelerometer	1	$14.95
Total	$1,233.91		
Saved	$964.96		
Final Total	**$276.96**		

D. Engineering Standards and Realistic Constraints

PROJECT: **Autonomous Car with Android-Based Control**
Team Leader: **Richard Bustamante**

Checklist Completed: [illegible] 04/22/2019
Technical Advisor: Dr. Ambrose Adegbege **Date:**

DIRECTIONS:
This checklist should be completed by the team leader and signed by the technical advisor.
In the self-check column, place a C if covered in body of report, a N if not covered in body of report but is covered in the remarks column, and a N/A if not applicable (in all cases, C, N, or N/a, include a justification in the remarks column).

	Self-Check				Remarks
	C	N	N/A		
Economic	X				Budget discussed in Section XII, Appendix C.
Environmental		X			The system has little environmental impact as it generates no external output, with the exception of ultrasonic waves from the four mounted sensors and the Lithium Polymer battery must be disposed in regard to local laws.
Social		X			The social impact of the autonomous systems focuses on privacy, as related to the streaming camera on board.
Political			X		This project has no political implications.
Ethical		X			The autonomous system can cause ethical concern regarding control and decision that impact the surrounding environment.
Health and Safety		X			Due to the nature of autonomous systems, the car was equipped with an emergency override to ensure safety in the surrounding environment.
Manufacturability			X		This project exists as a prototype.
Sustainability			X		Sustainability is not relevant for the project as it is a prototype.
Standards	X				Standards discussed when necessary throughout the report.

E. List of Contacts

Dr. Ambrose Adegbege
Senior Project Advisor, Assistant Professor
Director of LECO Lab
Department of Electrical and Computer Engineering
School of Engineering, The College of New Jersey
2000 Pennington Road, Ewing, NJ 08628-0718
E: adegbega@tcnj.edu
T: (609)771-2863

Mike Steeil
Engineering Laboratory Technician
School of Engineering, The College of New Jersey
2000 Pennington Road, Ewing, NJ 08628-0718
E: steeilm1@tcnj.edu
T: (609)771-2785

Dr. Allen Katz
Senior Project Coordinator
Professor of Electrical and Computer Engineering
New Jersey Institute of Technology (D. Sc. 1971)
E: a.katz@ieee.org

F. Car Operation

This section will give a brief overview of operating the car from start to finish. This will include turning the car on, downloading the application for the phone, and operating the application.

Setup Instructions

1. The phone user must set their Android phone into developer mode
   a. Go to Settings > About phone (About device) > Build number
   b. Tap Build number seven times
   c. Return to settings, Developer options is now listed at the bottom
   d. WIthin Developer options turn on USB Debugging

2. Download the Android .apk from https://github.com/sancher6/Android_App/tree/master/final_app
   a. Build app onto device through Android Studio (available for APK versions 15 and up)

3. Currently the hardware is configured so that the Raspberry Pi can take an input from a USB power brick to a micro-USB port for testing purposes. In this case, plug the Pi into the brick. If the buck converter is connected, continue to step 3a.
   a. The Raspberry Pi can typically be powered directly from the battery through a buck converter in order to step down the voltage to 5V. If in this configuration, turn the large switch clockwise.

4. Once the Raspberry Pi is powered wait 30 seconds for it to boot up and establish the WiFi server
   a. On the phone, connect to the raspnet WiFi with password: seniorproject

5. Now if step 3a. Was not used, turn the large switch clockwise to power the motor driver

6. Finally open the application on the phone and hit connect. If everything was done properly, the application menu will pop up and all the different tasks will operate properly.

Operation instructions

The application allows for a number of tasks to be completed. This will briefly give an outline of each option on the main menu of the application.

1. Remote Control - Allows the user to directly control the phone
   a. Up arrow - drives car forward
   b. Down arrow - drives car backward
   c. Left arrow - pivot left

    d. Right arrow - pivot right
    e. Stop - immediately stops regardless of instruction queue
    f. Disconnect - disconnects the app from the current session with the car
    g. Shutdown - powers down the Raspberry Pi so that it can be safely cut from power

2. Calibrate - Calibrates driving distances and turning angles. The full functionality is not currently implemented
    a. Distance - set a time in ms
    b. Angle - set turn amount in ms
    c. Save - save current calibration settings for autonomous mode
    d. Start - test the current set calibration
    e. Stop - stop test of current set calibration

3. Set Instructions - Allows users to queue up instructions for the car to execute. Must calibrate first
    a. Dropdown - select forward, backward, left, right, stop, for the next instruction
    b. Distance entry - Set how far the car should travel depending on calibrated setting
    c. "+" - Add the current instruction to the queue
    d. Name entry - Set a name for the current set of instructions
    e. Save - Add instruction set to database

4. Previous Runs - Contains a list of all set runs and allows users to start runs
    a. Select any saved run in the list and the car will start executing the instructions

5. Disconnect - Disconnects the phone from the current session with the car

6. More - Sends the user to the senior project webpage

G. Software

1. Android Application Code

i. activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:lines="2"
        android:text="@string/proj_title"
        android:gravity="center"
        android:textSize="32sp"
        android:textColor="@color/Blue" />

    <EditText
        android:id="@+id/port"
        android:layout_width="313dp"
        android:layout_height="57dp"
        android:layout_alignParentTop="true"
        android:layout_alignStart="@+id/tcnj"
        android:layout_alignLeft="@+id/tcnj"
        android:layout_marginTop="156dp"
        android:hint="@string/port"
        android:inputType="numberSigned" />

    <EditText
        android:id="@+id/ip"
        android:layout_width="313dp"
        android:layout_height="57dp"
        android:layout_alignParentTop="true"
        android:layout_alignStart="@+id/port"
        android:layout_alignLeft="@+id/port"
        android:layout_marginTop="90dp"
        android:hint="@string/ip"
        android:inputType="text" />

    <Button
        android:id="@+id/connect"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="@string/connect" />

    <ImageView
        android:id="@+id/tcnj"
        android:layout_width="wrap_content"
        android:layout_height="212dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="11dp"
        android:contentDescription="@string/TCNJ"
        app:srcCompat="@drawable/tcnj_engineering" />

</RelativeLayout>
```

ii. activity_secondary.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondaryActivity">


    <Button
        android:id="@+id/remote_control"
        android:layout_width="165dp"
        android:layout_height="58dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"
        android:text="@string/remote_control" />

    <Button
        android:id="@+id/calibrate"
        android:layout_width="165dp"
        android:layout_height="58dp"
        android:layout_below="@+id/remote_control"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"
        android:text="@string/calibrate" />

    <Button
        android:id="@+id/set_instructions"
        android:layout_width="165dp"
        android:layout_height="58dp"
        android:layout_below="@+id/calibrate"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"
        android:text="@string/set_instructions" />

    <Button
        android:id="@+id/previous_runs"
        android:layout_width="165dp"
        android:layout_height="58dp"
        android:layout_below="@+id/set_instructions"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"
        android:text="@string/previous_runs" />

    <Button
        android:id="@+id/disconnect"
        android:layout_width="165dp"
        android:layout_height="58dp"
        android:layout_below="@+id/previous_runs"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"
        android:text="@string/disconnect" />

    <Button
        android:id="@+id/more"
        android:layout_width="165dp"
        android:layout_height="58dp"
        android:layout_below="@+id/disconnect"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="25dp"
        android:text="@string/more" />

</RelativeLayout>
```

iii. activity_manual_override.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".ManualOverride">


<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

<Button
    android:id="@+id/stop"
    android:layout_width="75dp"
    android:layout_height="75dp"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:text="@string/OFF"
    android:background="@drawable/rounded" />

<Button
    android:id="@+id/off"
    android:layout_width="75dp"
    android:layout_height="75dp"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentBottom="true"
    android:text="@string/power"
    android:background="@drawable/rounded" />

<Button
    android:id="@+id/dc"
    android:layout_width="150dp"
    android:layout_height="75dp"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:text="@string/disconnect"
    android:background="@color/RED" />

<ImageButton
    android:id="@+id/forward"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="105dp"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_marginRight="75dp"
    android:layout_marginEnd="75dp"
```

```
        android:contentDescription="@string/forward"
        app:srcCompat="@drawable/up"
        android:background="@drawable/rounded"/>

    <ImageButton
        android:id="@+id/right"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="20dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="5dp"
        android:layout_marginEnd="5dp"
        android:contentDescription="@string/right"
        app:srcCompat="@drawable/right"
        android:background="@drawable/rounded"/>

    <ImageButton
        android:id="@+id/left"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="20dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="150dp"
        android:layout_marginEnd="150dp"
        android:contentDescription="@string/left"
        app:srcCompat="@drawable/left"
        android:background="@drawable/rounded"/>

    <ImageButton
        android:id="@+id/back"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="20dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_marginRight="75dp"
        android:layout_marginEnd="75dp"
        android:contentDescription="@string/back"
        app:srcCompat="@drawable/back"
        android:background="@drawable/rounded"/>

</RelativeLayout>
```

iv. activity_set_instructions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SetInstructions">

    <Spinner
        android:id="@+id/list"
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="15dp"
        android:layout_marginLeft="15dp"
        android:layout_marginTop="100dp" />

    <TextView
        android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:lines="2"
        android:maxLines="4"
        android:text="@string/set_instructions"
        android:gravity="center"
        android:textSize="32sp"
        android:textColor="@color/Blue" />

    <Button
        android:id="@+id/add"
        android:layout_width="50dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_marginTop="100dp"
        android:layout_marginEnd="15dp"
        android:layout_marginRight="15dp"
        android:textSize="20sp"
        android:text="@string/add" />
```

```
<ListView
    android:id="@+id/listV"
    android:layout_width="fill_parent"
    android:layout_height="220dp"
    android:layout_centerVertical="true"/>

<EditText
    android:id="@+id/dist"
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
    android:layout_marginTop="100dp"
    android:layout_marginEnd="80dp"
    android:layout_marginRight="80dp"
    android:hint="@string/dist"
    android:inputType="numberSigned" />

<EditText
    android:id="@+id/name"
    android:layout_width="313dp"
    android:layout_height="57dp"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="80dp"
    android:hint="@string/run_name"
    android:inputType="text" />


<TextView
    android:id="@+id/instr_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentLeft="true"
    android:layout_marginTop="150dp"
    android:text="@string/instruction"
    android:textSize="20sp" />

<Button
    android:id="@+id/save"
    android:layout_width="313dp"
    android:layout_height="75dp"
    android:layout_centerHorizontal="true"
    android:layout_alignParentBottom="true"
    android:textSize="20sp"
```

```
        android:text="@string/save" />

</RelativeLayout>
```

v. activity_previous_runs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PreviousRuns">

    <TextView
        android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:lines="2"
        android:maxLines="4"
        android:text="@string/previous_runs"
        android:gravity="center"
        android:textSize="32sp"
        android:textColor="@color/Blue" />

    <ListView
        android:id="@+id/list"
        android:layout_width="fill_parent"
        android:layout_height="400dp"
        android:layout_centerVertical="true"/>

</RelativeLayout>
```

vi. activity_calibrate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Calibrate">

    <TextView
        android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:lines="2"
        android:text="@string/calibrate"
        android:gravity="center"
        android:textSize="32sp"
        android:textColor="@color/Blue" />

    <Spinner
        android:id="@+id/options"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/title"
        android:layout_marginTop="100dp" />

    <EditText
        android:id="@+id/time"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/options"
        android:layout_marginTop="40dp"
        android:hint="@string/Time"
        android:inputType="numberSigned" />

    <Button
        android:id="@+id/Start"
        android:layout_width="100dp"
        android:layout_height="58dp"
        android:layout_below="@+id/time"
        android:layout_marginTop="100dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_marginStart="25dp"
        android:layout_marginLeft="25dp"
        android:text="@string/start" />

    <Button
        android:id="@+id/Stop"
        android:layout_width="100dp"
        android:layout_height="58dp"
        android:layout_below="@+id/time"
        android:layout_marginTop="100dp"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_marginEnd="25dp"
        android:layout_marginRight="25dp"
        android:text="@string/stop" />

    <Button
        android:id="@+id/Save"
        android:layout_width="165dp"
        android:layout_height="58dp"
        android:layout_below="@+id/Start"
        android:layout_marginTop="40dp"
        android:layout_centerHorizontal="true"
        android:text="@string/save" />

</RelativeLayout>
```

vii. Colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="Blue">#363789</color>
    <color name="connected">#00ff00</color>
    <color name="RED">#ff0000</color>
    <color name="white">#ffffff</color>
</resources>
```

Viii. strings.xml

```
<resources>
    <string name="app_name">RaspberryPiClient</string>
    <string name="proj_title">Autonomous Car with Android-Based
Control</string>
    <string name="port">Port</string>
    <string name="ip">Raspberry Pi IP</string>
    <string name="connect">Connect</string>
    <string name="TCNJ">TCNJ School of Engineering</string>

    <string name="power">POWER</string>

    <!--secondary activity-->
    <string name="remote_control">Remote Control</string>
    <string name="set_instructions">Set Instructions</string>
    <string name="previous_runs">Previous Runs</string>
    <string name="calibrate">Calibrate</string>
    <string name="disconnect">DISCONNECT</string>
    <string name="more">MORE!!</string>
    <string name="ip_num">192.168.4.1</string>
    <string name="port_num">4957</string>

    <string name="start">START</string>
    <string name="stop">STOP</string>
    <string name="forward">F</string>
    <string name="left">L</string>
    <string name="right">R</string>
    <string name="back">B</string>
    <string name="OFF">stop</string>

    <!--set instructions activity-->
    <string name="run_name">Enter a Name for this Set</string>
    <string name="action">Action</string>
    <string name="dist">Distance</string>
    <string name="save">Save</string>
    <string name="add">+</string>
    <string name="instruction">Instructions :</string>

    <string-array name="instructions">
        <item>Forward</item>
        <item>Backward</item>
        <item>Left</item>
        <item>Right</item>
        <item>stop</item>
    </string-array>

    <!--calibrate activity-->
    <string name="Time">Time(ms)</string>
    <string-array name="options">
        <item>Distance</item>
        <item>Angle(Degrees)</item>
    </string-array>

</resources>
```

IX. styles.xml

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
```

```
</resources>
```

X. MainActivity.java

```
package com.e.raspberrypiclient;

import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

import static com.e.raspberrypiclient.GlobalApplication.makeToast;

public class MainActivity extends AppCompatActivity {
    private Button Connect;
    private EditText ip,port;
    private String TAG = "MAIN";
    private String ipaddress;
    private int portnum;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Buttons and Edit Texts are all set
        Connect = (Button)findViewById(R.id.connect);

        //EditText Fields
        ip = (EditText)findViewById(R.id.ip);
        port = (EditText)findViewById(R.id.port);
        //connect to pi
        Connect.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d("Connect : ", "CONNECT BUTTON PRESSED");
}
                String check = ip.getText().toString();
                String check2 = port.getText().toString();
                if(!check.isEmpty() && !check2.isEmpty()){
                    ipaddress = check;
                    portnum = Integer.parseInt(check2);
                }else {
                    ipaddress = "192.168.4.1";
                    portnum = 4957;
                }
                makeToast("Connecting");
                Intent intent = new Intent(MainActivity.this,
SecondaryActivity.class);
                intent.putExtra("ip",ip.getText().toString());
                intent.putExtra("port",port.getText().toString());
                startActivity(intent);
            }
        });
    }
}
```

XI. SecondaryActivity.java

```
package com.e.raspberrypiclient;

import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import static com.e.raspberrypiclient.GlobalApplication.makeToast;

public class SecondaryActivity extends AppCompatActivity {
    private Button remote;
    private Button set_instructions;
    private Button previous_runs;
    private Button dc;
    private Button more;
    private Button calibrate;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_secondary);

        //Buttons are all set
        remote = (Button)findViewById(R.id.remote_control);
        calibrate = (Button)findViewById(R.id.calibrate);
        set_instructions = (Button)findViewById(R.id.set_instructions);
        previous_runs = (Button)findViewById(R.id.previous_runs);
        dc = (Button)findViewById(R.id.disconnect);
        more = (Button)findViewById(R.id.more);
        //onClickListeners for Buttons
        remote.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d("Remote Control : ", "REMOTE CONTROL BUTTON PRESSED");
                Intent intent = new Intent(SecondaryActivity.this,
ManualOverride.class);
                startActivity(intent);
            }
        });

        calibrate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d("Calibrate : ", "CALIBRATE BUTTON PRESSED");
                Intent intent = new Intent(SecondaryActivity.this,
Calibrate.class);
                startActivity(intent);
            }
        });

        set_instructions.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d(" : ", "SET INSTRUCTIONS BUTTON PRESSED");
                Intent intent = new Intent(SecondaryActivity.this,
SetInstructions.class);
                startActivity(intent);
            }
        });

        previous_runs.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d(" : ", "PREVIOUS RUNS BUTTON PRESSED");
                Intent intent = new Intent(SecondaryActivity.this,
PreviousRuns.class);
                startActivity(intent);
            }
        });

        dc.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d(" : ", "DISCONNECT BUTTON PRESSED");
                Intent intent = new Intent(SecondaryActivity.this,
MainActivity.class);
                makeToast("Disconnecting ...");
                startActivity(intent);
            }
        });

        more.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
////                Log.d(" : ", "MORE BUTTON PRESSED");
                Intent browserIntent = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://engprojects.tcnj.edu/rccar19/the-team/"));
                startActivity(browserIntent);
            }
        });
    }
    @Override
    public void onBackPressed(){
        Intent intent = new Intent(SecondaryActivity.this, MainActivity.class);
        startActivity(intent);
    }
}
```

Xii. ManualOverride.java

```
package com.e.raspberrypiclient;

import android.annotation.SuppressLint;
import android.content.DialogInterface;
import android.content.Intent;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.ImageButton;

import static com.e.raspberrypiclient.GlobalApplication.makeToast;

public class ManualOverride extends AppCompatActivity {
    private WebView webView;
    private ImageButton f;
    private ImageButton b;
    private ImageButton l;
    private ImageButton r;
    private Button stop;
    private Button dc;
    private Button off;
    private Client client;
    AlertDialog.Builder builder;
    private String TAG = "MANUAL OVERRIDE";

    @SuppressLint("ClickableViewAccessibility")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_manual_override);

        Intent in = getIntent();
        Bundle bundle = in.getExtras();

        webView = findViewById(R.id.webview);
        webView.getSettings().setCacheMode(WebSettings.LOAD_DEFAULT);
        webView.setWebViewClient(new WebViewClient());
        webView.loadUrl("http://192.168.4.1:8000/");

        client = new Client("192.168.4.1",4957);
//          Log.d(TAG, "Client Created");
        client.start();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        if(bundle!=null){
            client.setToReturn("a");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            String longinstr = (String)bundle.get("AUTO");
            client.setToReturn(longinstr);
//              client.setToReturn("end");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        client.setToReturn("m");


        //ImageButtons are all set
        f = (ImageButton)findViewById(R.id.forward);
        b = (ImageButton)findViewById(R.id.back);
        l = (ImageButton)findViewById(R.id.left);
        r = (ImageButton)findViewById(R.id.right);
        stop = (Button)findViewById(R.id.stop);
        dc = (Button)findViewById(R.id.dc);
        off = (Button)findViewById(R.id.off);
        builder = new AlertDialog.Builder(this);

        //forward Held
        f.setOnTouchListener(new View.OnTouchListener(){
            @Override
            public boolean onTouch(View v, MotionEvent event){
            if(event.getAction() == MotionEvent.ACTION_DOWN){
                //send forward as long as this is held
                client.setToReturn("f");
//                    Log.d("SENDING ", "FORWARD");
                return true;
            }else if(event.getAction() == MotionEvent.ACTION_UP) {
                client.setToReturn("end");
                return false;
            }
            return false;
            }
        });

        b.setOnTouchListener(new View.OnTouchListener(){
            @Override
            public boolean onTouch(View v, MotionEvent event){
            if(event.getAction() == MotionEvent.ACTION_DOWN){
                //send forward as long as this is held
                client.setToReturn("b");
//                    Log.d("SENDING ", "BACKWARD");
                return true;
            }else if(event.getAction() == MotionEvent.ACTION_UP){
                client.setToReturn("end");
                return false;
            }
            return false;
            }
        });
        l.setOnTouchListener(new View.OnTouchListener(){
            @Override
            public boolean onTouch(View v, MotionEvent event){
            if(event.getAction() == MotionEvent.ACTION_DOWN){
                //send forward as long as this is held
                client.setToReturn("l");
                return true;
            }else if(event.getAction() == MotionEvent.ACTION_UP){
                client.setToReturn("end");
                return false;
            }
            return false;
            }
        });
        r.setOnTouchListener(new View.OnTouchListener(){
            @Override
            public boolean onTouch(View v, MotionEvent event){
            if(event.getAction() == MotionEvent.ACTION_DOWN){
                //send forward as long as this is held
                client.setToReturn("r");
                return true;
            }else if(event.getAction() == MotionEvent.ACTION_UP){
                client.setToReturn("end");
                return false;
            }
            return false;
            }
        });
        //stop
        stop.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d("Command : ", "STOP BUTTON PRESSED");
            client.setToReturn("stop");
            }
        });

        //disconnect
        dc.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
//                Log.d("Command : ", "DISCONNECT  BUTTON PRESSED");
            client.setToReturn("Disconnect");
            finish();
            }
        });

        //POWER OFF
        off.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
            //Setting message manually and performing action on button click
            builder.setMessage("Do you want to Shutdown?")
                    .setCancelable(false)
                    .setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            //  Action for 'YES' Button
                            client.setToReturn("Off");
                        }
                    })
                    .setNegativeButton("No", new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            //  Action for 'NO' Button
                            dialog.cancel();
                        }
                    });
            //Creating dialog box
            AlertDialog alert = builder.create();
            //Setting the title manually
            alert.setTitle("ALERT");
            alert.show();
            }
        });
    }

    @Override
    public void onBackPressed(){
//        Log.d("BACK BUTTON PRESSED ", "DISCONNECTING");
        client.setToReturn("Disconnect");
        finish();
    }

    @Override
    public void onStop(){
        super.onStop();
        client.disconnect();
        finish();
    }
}
```

Xiii. Client.java

```
package com.e.raspberrypiclient;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import android.util.Log;

public class Client extends Thread{
    private String ip;
    private int port;
    private OutputStreamWriter osw;
    private InputStreamReader isr;
    private String TAG = "CLIENT";
    private String toReturn;
    private String received;
    private String previous;
    public Socket clientSocket;
    public boolean connect;
    public boolean still;

    public Client(String ip, int port) {
        this.ip = ip;
        this.port = port;
    }

    @Override
    public void run(){
        Log.d(TAG, "Running");
        try {
            clientSocket = new Socket(ip,port);
            osw = new
OutputStreamWriter(clientSocket.getOutputStream(),"UTF-8");
            isr = new InputStreamReader(clientSocket.getInputStream(),"UTF-8");
            connect = false;
            still= true;
            previous = "";
        } catch (IOException e) {
            disconnect();
        }
        PrintWriter out = new PrintWriter(osw, true);
        BufferedReader in = new BufferedReader(isr);
//      Send Connect Instruction
        setToReturn("Connect");
        while(true){
            //ensure toReturn is set to the correct String
            if((connect && toReturn.equalsIgnoreCase("Connect"))){
                //nothing
            }else{
                if(toReturn.equalsIgnoreCase(previous)) {
                    //nothing
                }else{
                    previous = toReturn;
                    out.println(getToReturn());
                    try {
                        received = in.readLine();
                    } catch (Exception e) {
                        received = "Connect";
                    }
                    //            Log.d(TAG,"SERVER SAYS: " + received);
                    if (received.equalsIgnoreCase("Disconnecting")) {
//                        Log.d(TAG, "Closing Connection");
                        try {
                            out.close();
                            in.close();
                            clientSocket.close();
                            connect = false;
                        } catch (IOException e) {
//                            Log.d("ERROR ", e.getMessage());
                            disconnect();
                        }
                        break;
                    } else if (received.equalsIgnoreCase("connected")) {
                        connect = true;
                    } else if (received.equalsIgnoreCase("off")) {
                        connect = false;
                        break;
                    }
                }

            }
        }
    }

    public void setToReturn(String message){
//        Log.d(TAG,message);
        toReturn = message;
    }
    public String getToReturn(){
        return toReturn;
    }

    public void disconnect(){
        try {
            osw.close();
            isr.close();
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        connect = false;
    }
```

```
}
```

XIV. Calibrate.java

```
package com.e.raspberrypiclient;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

import static com.e.raspberrypiclient.GlobalApplication.makeToast;

public class Calibrate extends AppCompatActivity {
    private DatabaseHelper myDB;
    private Spinner opt_spinner;
    private EditText time;
    private Button start;
    private Button stop;
    private Button save;
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;
    private Client client;
    private String TAG = "MANUAL OVERRIDE";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_calibrate);

        //initialize variables
        start = (Button)findViewById(R.id.Start);
        stop = (Button)findViewById(R.id.Stop);
        save = (Button)findViewById(R.id.Save);
        opt_spinner = (Spinner)findViewById(R.id.options);
        time = (EditText)findViewById(R.id.time);
        //initialize database
        myDB = new DatabaseHelper(this);

        //      THIS IS FOR THE
SPINNER/////////////////////////////////////////////////////////////////////////
        ArrayAdapter<String> myAdapter = new ArrayAdapter<>(Calibrate.this,
                android.R.layout.simple_list_item_1,
getResources().getStringArray(R.array.options));


myAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);
        opt_spinner.setAdapter(myAdapter);

        //to get spinner selected item
        // String text = opt_spinner.getSelectedItem().toString();

        start.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                if(time.getText().toString().length() < 1){
                    makeToast("Must enter a valid Time!!");
                }else{
//
client.setToReturn(opt_spinner.getSelectedItem().toString() + " " +
time.getText().toString());
                }
            }
        });

//        stop.setOnClickListener(new View.OnClickListener(){
//            @Override
//            public void onClick(View v) {
//                client.setToReturn("stop");
//            }
//        });

        save.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                if(time.getText().toString().length() < 1) {
                    makeToast("Must enter a valid Time!!");
                }else {
//                      distance calibration

if(opt_spinner.getSelectedItem().toString().equalsIgnoreCase("Distance")) {
```

```
                        boolean namecheck =
myDB.findName("Calibration(distance)");
                        if (namecheck) {
                            boolean insertData =
myDB.addData("Calibration(distance)", "Forward " + time.getText().toString());
                            if (insertData) {
                                makeToast("Successfully Saved Distance
Calibration!");
                            } else {
                                makeToast("Error Saving Run");
                            }
                        } else {
                            String temp = myDB.getId("Calibration(distance)");
                            boolean replace = myDB.nameReplace(temp,
"Calibration(distance)", "Forward " + time.getText().toString());
                            makeToast("OVERWRITING PREVIOUS DISTANCE
CALIBRATION");
                        }
                    }
//                    angle calibration
                    else{
                        boolean namecheck =
myDB.findName("Calibration(angle)");
                        if (namecheck) {
                            boolean insertData =
myDB.addData("Calibration(angle)", "Left " + time.getText().toString());
                            if (insertData) {
                                makeToast("Successfully Saved Angle
Calibration!");
                            } else {
                                makeToast("Error Saving Run");
                            }
                        } else {
                            String temp = myDB.getId("Calibration(angle)");
                            boolean replace = myDB.nameReplace(temp,
"Calibration(angle)", "Left " + time.getText().toString());
                            makeToast("OVERWRITING PREVIOUS ANGLE
CALIBRATION");
                        }
                    }
                }
            }
        });
    }
}
```

XV. PreviousRuns.java

```
package com.e.raspberrypiclient;
```

```
import android.content.Intent;
import android.database.Cursor;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;

import static com.e.raspberrypiclient.GlobalApplication.makeToast;

public class PreviousRuns extends AppCompatActivity {
    private static final String TAG = "Previous Runs Activity";
    DatabaseHelper mDatabaseHelper;
    private ListView mListView;
    private ArrayList<String> stringArrayList;
    private ArrayAdapter<String> stringArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_previous_runs);

        mListView = (ListView)findViewById(R.id.list);

        mListView.setAdapter(stringArrayAdapter);

        mDatabaseHelper = new DatabaseHelper(this);
        final String[] instr =
getResources().getStringArray(R.array.instructions);

        populateListView();

        mListView.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                boolean calCheck =
mDatabaseHelper.findName("Calibration(distance)") ||
mDatabaseHelper.findName("Calibration(angle)");
                //calibration found
                if(!calCheck){
                    String longInstr =
mDatabaseHelper.getInstr(mListView.getAdapter().getItem(position).toString());
//                    makeToast("Selected: " +
mListView.getAdapter().getItem(position).toString());
                    Intent i = new Intent(PreviousRuns.this,
ManualOverride.class);
                    Log.d(TAG,longInstr);
                    i.putExtra("AUTO", longInstr);
                    startActivity(i);
                } else{makeToast("MUST CALIBRATE FIRST!");}
            }
        });
        mListView.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> parent, View view,
int position, long id) {
                //delete item

mDatabaseHelper.deleteData(mDatabaseHelper.getId(mListView.getAdapter().getItem
(position).toString()));

stringArrayAdapter.remove(mListView.getAdapter().getItem(position).toString());
                stringArrayAdapter.notifyDataSetChanged();
                makeToast("Deleted");
                return false;
            }
        });
    }

    private void populateListView(){
        Log.d(TAG, "populateListView: Displaying Data within the ListView");
        //get the data and append to a list
        Cursor data = mDatabaseHelper.getData();

        stringArrayList = new ArrayList<>();
        if(data.getCount() == 0){
            makeToast("Database is Empty!");
        } else{
            while(data.moveToNext()){
                //get data from column 1 added to list
                stringArrayList.add(data.getString(1));

                //create the list adapter and set the adapter
                stringArrayAdapter = new
ArrayAdapter<String>(getApplicationContext(),
                        android.R.layout.simple_list_item_1, stringArrayList);
                mListView.setAdapter(stringArrayAdapter);
            }
        }
    }
    @Override
    public void onBackPressed(){
//        Log.d("BACK BUTTON PRESSED ", "DISCONNECTING");
        Intent intent = new Intent(PreviousRuns.this, SecondaryActivity.class);
        startActivity(intent);
    }
}
```

XVI. SetInstructions.java

```
package com.e.raspberrypiclient;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Spinner;

import java.util.ArrayList;

import static com.e.raspberrypiclient.GlobalApplication.makeToast;
import static java.lang.Character.toLowerCase;

public class SetInstructions extends AppCompatActivity {
    private DatabaseHelper myDB;
    private Button save;
    private Button add_instr;
    private Spinner instr_spinner;
    private ListView instrlist;
    private ArrayList<String> stringArrayList;
    private ArrayAdapter<String> stringArrayAdapter;
    private EditText distance;
    private EditText name;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_set_instructions);

        add_instr = (Button)findViewById(R.id.add);
        save = (Button)findViewById(R.id.save);
        distance = (EditText)findViewById(R.id.dist);
        name = (EditText)findViewById(R.id.name);
        instr_spinner = (Spinner)findViewById(R.id.list);
        instrlist = (ListView)findViewById(R.id.listV);
        stringArrayList = new ArrayList<String>();
        myDB = new DatabaseHelper(this);


//      THIS IS FOR THE
LIST//////////////////////////////////////////////////////////////////////
        stringArrayAdapter = new ArrayAdapter<String>(getApplicationContext(),
                android.R.layout.simple_list_item_1, stringArrayList);
        instrlist.setAdapter(stringArrayAdapter);

//      THIS IS FOR THE
SPINNER///////////////////////////////////////////////////////////////////////
        final ArrayAdapter<String> myAdapter = new
ArrayAdapter<String>(SetInstructions.this,
                android.R.layout.simple_list_item_1,
getResources().getStringArray(R.array.instructions));


myAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);
        instr_spinner.setAdapter(myAdapter);

        //to get spinner selected item
        // String text = instr_spinner.getSelectedItem().toString();

        save.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){
                if(stringArrayList.isEmpty()){
                    makeToast("MUST FIRST ADD INSTRUCTIONS!");
                } else{
                    if(name.getText().toString().length() < 1) {
                        makeToast("MUST ENTER A RUN NAME IN ORDER TO SAVE!");
                    } else{
                        boolean namecheck =
myDB.findName(name.getText().toString());

                        //name is not in database
                        if(namecheck){
                            boolean insertData =
myDB.addData(name.getText().toString(), getAll(stringArrayList));

                            if(insertData){
                                makeToast("Successfully Saved Run!");
                            } else{
                                makeToast("Error Saving Run");
                            }
                        //name is in database
                        } else{
                            String temp =
myDB.getId(name.getText().toString());
                            boolean replace =
myDB.nameReplace(temp,name.getText().toString(), getAll(stringArrayList));
                            makeToast("Name already exists, overwriting");
                        }
                    }
                }
            }
        });

        add_instr.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                boolean calCheck = myDB.findName("Calibration(distance)") ||
myDB.findName("Calibration(angle)");

                //calibration found
                if(!calCheck){
                    if(distance.getText().toString().length() < 1){
                        makeToast("Must enter a valid Distance!!");
                    }else{
                        makeToast("Calibrating Instruction");
                        String textInt = distance.getText().toString();
                        int distInt = Integer.parseInt(textInt);
                        String instr =
instr_spinner.getSelectedItem().toString();
                        if(instr.equalsIgnoreCase("forward") ||
instr.equalsIgnoreCase("backward")){
                            //get distance calibration
                            textInt = myDB.getInstr("Calibration(distance)");
                            textInt = textInt.replace("Forward ","");
                            int temp = Integer.parseInt(textInt);
                            textInt = String.valueOf(distInt*temp);
                        }else{
                            textInt = myDB.getInstr("Calibration(angle)");
                            textInt = textInt.replace("Left ","");
                            int temp = Integer.parseInt(textInt);
                            textInt = String.valueOf(distInt*temp);
                        }
                        stringArrayList.add(instr + " " + textInt);
                        stringArrayAdapter.notifyDataSetChanged();
                        distance.setText("");
                    }
                }else{makeToast("MUST CALIBRATE FIRST!");}
            }
        });
    }
    public String getAll(ArrayList<String> instructions){
        String temp = "";
        for(int i = 0; i < instructions.size(); i++){
            String[] splited = instructions.get(i).split("\\s+");
            temp = temp + toLowerCase(splited[0].charAt(0))+ " " + splited[1]+"
";
        }
        return temp;
    }
}
```

XVII. DatabaseHelper.java

```
package com.e.raspberrypiclient;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper extends SQLiteOpenHelper {
    public static final String DATABASE_NAME = "mylist.db";
    public static final String TABLE_NAME = "mylist_data";
    public static final String COL1 = "ID";
    public static final String COL2 = "NAME";
    public static final String COL3 = "INSTRUCTIONS";

    public DatabaseHelper(Context
context){super(context,DATABASE_NAME,null,1);}


    @Override
    public void onCreate(SQLiteDatabase db) {
        String createTable = "CREATE TABLE " + TABLE_NAME + " (ID INTEGER
PRIMARY KEY AUTOINCREMENT, "
                + "NAME TEXT, INSTRUCTIONS TEXT)";
        db.execSQL(createTable);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }

    public boolean addData(String name, String instructions){
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put(COL2,name);
        contentValues.put(COL3,instructions);

        long result = db.insert(TABLE_NAME, null, contentValues);

        if(result == -1){
            return false;
        }else{
            return true;
        }
    }

    public Cursor getData(){
        SQLiteDatabase db = this.getWritableDatabase();
        Cursor data = db.rawQuery("SELECT * FROM " + TABLE_NAME,null);

        return data;
    }

    public boolean findName(String name){
        SQLiteDatabase db = this.getWritableDatabase();
        Cursor data = db.rawQuery("SELECT * FROM " + TABLE_NAME,null);

        while(data.moveToNext()){
            //name is in database return false
            if(name.equals(data.getString(1))){
                data.close();
                return false;
            }
        }
        //name not in database, return true;
        data.close();
        return true;
    }

    public String getId(String name){
        SQLiteDatabase db = this.getWritableDatabase();
        Cursor data = db.rawQuery("SELECT * FROM " + TABLE_NAME,null);
        String temp = "";

        while(data.moveToNext()){
            if(name.equals(data.getString(1))){
                temp = data.getString(0);
                data.close();
                return temp;
            }
        }
        return temp;
    }

    public boolean nameReplace(String id, String name, String instructions){
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();

        contentValues.put(COL1,id);
        contentValues.put(COL2,name);
        contentValues.put(COL3,instructions);

        long result = db.update(TABLE_NAME, contentValues, "ID = ?", new
String[] {id});

        if(result == -1){
            return false;
        }else{
            return true;
        }

    }

    public String getInstr(String name){
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor data = db.rawQuery("SELECT * FROM " + TABLE_NAME,null);

        String temp = "";

        while(data.moveToNext()){
            if(name.equals(data.getString(1))){
                temp = data.getString(2);
                data.close();
                return temp;
            }
        }
        return temp;
    }
    public Integer deleteData(String id){
        SQLiteDatabase db = this.getWritableDatabase();
        return db.delete(TABLE_NAME, "ID = ?", new String[] {id});
    }
}
```

XVIII. GlobalApplication.java

```
package com.e.raspberrypiclient;

import android.app.Application;
import android.content.Context;
import android.widget.Toast;

public class GlobalApplication extends Application {

    private static Context appContext;

    @Override
    public void onCreate() {
        super.onCreate();
        appContext = getApplicationContext();

        /* If you has other classes that need context object to initialize when
application is created,
         you can use the appContext here to process. */
    }

    public static Context getAppContext() {
        return appContext;
    }

    public static void makeToast(String Butter){
        Toast.makeText(getAppContext(), Butter,
                Toast.LENGTH_SHORT).show();
    }
}
```

XIX. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.e.raspberrypiclient">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.BLUETOOTH" />

    <application
        android:name=".GlobalApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="true"
        tools:ignore="GoogleAppIndexingWarning">
        <activity android:name=".Calibrate" />
        <activity
            android:name=".PreviousRuns"
            android:screenOrientation="portrait" />
        <activity
            android:name=".SetInstructions"
            android:screenOrientation="portrait"
            android:windowSoftInputMode="stateHidden|adjustNothing" />
        <activity
            android:name=".ManualOverride"
            android:screenOrientation="landscape" />
        <activity
            android:name=".SecondaryActivity"
            android:screenOrientation="portrait" />
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

2. Raspberry Pi Code

i. Code for Sending Strings over the Serial Connection

```
import serial

ser = serial.Serial('/dev/ttyACM0',9600)

def send_data(message):
   N = len(str(message))
   if N < 10:
       print(N)
       ser.write('0'.encode('utf-8'))
       ser.write(str(N).encode('utf-8'))

   else:
       ser.write(str(N).encode('utf-8'))

   ser.write(message.encode('utf-8'))
```

```
msg = input('Please type a message to send: ')

send_data(msg)
```

3. Arduino Code

```
//#define ENCODER_OPTIMIZE_INTERRUPTS

#include <Wire.h>
#include <SPI.h>
#include <Encoder.h>
#include <Adafruit_LSM9DS1.h>
#include <Adafruit_Sensor.h>


#define trigPinFront A0
#define echoPinFront A1

#define trigPinBack A2
#define echoPinBack A3


#define trigPinLeft A5
#define echoPinLeft A4

#define trigPinRight A6
#define echoPinRight A7

#define sensor_ISR 19
#define raspberry_Counter 31

//Hall Sensors
#define hallA 2
#define hallB 3

#define hallC 20
#define hallD 21

//Motor A
#define motorPin1 8  // Pin 14 of L293
#define motorPin2 7  // Pin 10 of L293

#define enB 4

#define motorPin3 6  // Pin 14 of L293
#define motorPin4 5  // Pin 10 of L293
```

```
#define enA 9


Encoder rightEnc(hallA, hallB);
Encoder leftEnc(hallC, hallD);

unsigned long timer = 0;
unsigned long auto_time = 0;


Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();


volatile int isPressed = 0;
volatile int sendInfo = 0;

volatile byte vals[6];
volatile byte prev[6];

volatile int f_sense;
volatile int l_sense;
volatile int r_sense;
volatile int b_sense;

//Incoming Serial Comm--------------------

char receivedChar;
boolean newData = false;

//PI Globals------------------------------------
char moving = 'n';
char man = 'y';


double previous_error_r = 0;
double integral_r = 0;
double deriv_r = 0;
int setpoint_speed_r = 300;
double k_pr = .25;
double k_ir = 2;
double k_dr = 0;

double previous_error_l = 0;
double integral_l = 0;
double deriv_l = 0;
int setpoint_speed_l = 300;
double k_pl = .5;
double k_il = 2;
double k_dl = 0;

boolean pid_test = true;

//-------------------------------------------

void setup() {

  Serial.begin(9600);

  pinMode(sensor_ISR, INPUT);

  pinMode(raspberry_Counter,OUTPUT);

  attachInterrupt(digitalPinToInterrupt(sensor_ISR), read_sensors_ISR, RISING);

  pinMode(trigPinFront,OUTPUT);
  pinMode(trigPinBack,OUTPUT);
  pinMode(trigPinLeft,OUTPUT);
  pinMode(trigPinRight,OUTPUT);

  pinMode(echoPinFront,INPUT);
  pinMode(echoPinBack,INPUT);
  pinMode(echoPinLeft,INPUT);
  pinMode(echoPinRight,INPUT);

  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(enA, OUTPUT);

  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
  pinMode(enB, OUTPUT);

  timer = millis();
  auto_time = millis();

}

long right_pos = -999;
long left_pos = -999;


void loop() {

 if(sendInfo == 1) {
   Serial.write((const char *)vals, sizeof(vals));
   sendInfo = 0;
 }


  int t_step = 50;
  int auto_timelimit = 5000;

  long new_right_pos, new_left_pos;
  long right_speed, left_speed;

  double error_r;
  double pid_speed_r;
  int error_l;
  int pid_speed_l;
  double output_r;
  int output_l;

  int enc_r;
  int enc_l;

 if (f_sense < 5){
   man = 'n';
   auto_time = millis();
 }

 if(millis() - auto_time >= auto_timelimit)
     man = 'y';

 if (millis() - timer >= t_step){
   Serial.println(f_sense);
   timer = millis();
   new_right_pos = rightEnc.read();
   new_left_pos = leftEnc.read();

   enc_r = new_right_pos - right_pos;
   vals[4] = new_right_pos - right_pos;
   right_pos = new_right_pos;

   enc_l = new_left_pos - left_pos;
   vals[5] = new_left_pos - left_pos;
   left_pos = new_left_pos;



   if (moving == 'y') {

     error_r = setpoint_speed_r - enc_r;
```

```
      integral_r = integral_r + (error_r*t_step*.001);
      //deriv_r = (error_r - previous_error_r) / (t_step*.001);
      output_r = (k_pr * error_r) + (k_ir* integral_r);// + (k_dr * deriv_r);
      previous_error_r = error_r;

      error_l = setpoint_speed_l - enc_l;
      integral_l = integral_l + (error_l*t_step * .001);
      //deriv_l = (error_l - previous_error_l) / t_step;
      output_l = (k_pl * error_l) + (k_il* integral_l);// + (k_dl * deriv_l);
      previous_error_l = error_l;

      forwardRight((int)output_r);
      forwardLeft((int)output_l);

    }

    else if (moving == 'n') {
      output_r = 0;
      output_l = 0;
      previous_error_r = 0;
      integral_r = 0;
      deriv_r = 0;
      previous_error_l = 0;
      integral_l = 0;
      deriv_l = 0;

//      stopLeft();
//      stopRight();
    }

  }



  recvInfo();
 if (man == 'y')
    drive();

 else if (man == 'n'){
   stopLeft();
   stopRight();
 }

}
void read_sensors_ISR(){
  digitalWrite(trigPinFront,LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinFront,HIGH);
  delayMicroseconds(10);
  f_sense = (pulseIn(echoPinFront,HIGH)/2)/29.1;

  digitalWrite(trigPinBack,LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinBack,HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinBack,LOW);
  b_sense = (pulseIn(echoPinBack,HIGH)/2)/29.1;

  digitalWrite(trigPinLeft,LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinLeft,HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinLeft,LOW);
  l_sense = (pulseIn(echoPinLeft,HIGH)/2)/29.1;

  digitalWrite(trigPinRight,LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinRight,HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinRight,LOW);
  r_sense = (pulseIn(echoPinRight,HIGH)/2)/29.1;

//  for(int i = 0; i < 6; i++)
//  {
//    if(vals[i] > 200)
//      vals[i] = 200;
//    else if(vals[i] == 0)
//    {
//      vals[i] = 240;
//    }
//
//    prev[i] = vals[i];
//  }


  sendInfo = 1;
}

//------------------------------------Forward
Left---------------------------------

void forwardLeft(int speed){
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);

  analogWrite(enB, speed);
```

```
}

//----------------------------------Backward
Left--------------------------------

void backwardLeft(int speed){
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);

  analogWrite(enB, speed);
}

//----------------------------------Stop Left--------------------------------

void stopLeft(){
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);

  analogWrite(enB, 0);
}

//----------------------------------Forward
Right--------------------------------

void forwardRight(int speed){
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);

  analogWrite(enA, speed);

}

//----------------------------------Backward
Right--------------------------------

void backwardRight(int speed){
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);

  analogWrite(enA, speed);
}

//----------------------------------Stop Right--------------------------------

void stopRight(){
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  analogWrite(enA, 0);
}

//--------------------------------RecvINFO----------------------------------

void recvInfo() {


  if (Serial.available() > 0) {

    receivedChar = Serial.read();
    newData = true;

  }
}

void drive(){
  char howToMove = receivedChar;

  int car_speed = 100; //change between 80-150

  while(newData == true){
    if (howToMove == 'f'){
      forwardLeft(car_speed+10);
      forwardRight(car_speed);
    }

    else if (howToMove == 's'){
      moving = 'n';
      stopLeft();
      stopRight();
    }

    else if (howToMove == 'l'){
      moving = 'n';
      stopLeft();
      forwardRight(car_speed);
    }

    else if (howToMove == 'r'){
      moving = 'n';
      stopRight();
      forwardLeft(car_speed+10);
    }

    else if (howToMove == 'b'){
      moving = 'n';
      backwardRight(car_speed);
      backwardLeft(car_speed+10);
    }

    else if (howToMove == 'e'){
      moving = 'n';
      stopLeft();
      stopRight();
    }


    newData = false;
  }
}
```