

AUTONOMOUS LAWNMOWER

Max Lesser

Vincent McMasters

Jonathan Poulouse

Josh Samaniego

SUBSYSTEM REPORTS

REVISION – 3
29 April 2021

Table of Contents

Power Supply Subsystem Report	98
MCU Subsystem Report	119
Motors Subsystem Report	152
User Interface Subsystem Report	163

Autonomous Lawnmower

Vincent McMasters

POWER SUPPLY SUBSYSTEM REPORT

REVISION – 3
24 April 2021

Table of Contents

1 Subsystem Introduction	101
2 Simulation	103
3 Printed Circuit Board (PCB)	106
3.1 PCB Evaluation	107
4 Load Testing	109
5 Final Subsystem at the end of 403	111
5.1 Subsystem Weight at the end of 403	111
5.2 Subsystem Dimensions at the end of 403	111
5.3 Subsystem Image at the end of 403	112
6 Component Mounting	113
6.1 Internal Mounting	113
6.2 Wiring	114
6.3 Final Assembled Component Box	115
7 Unexpected Late Changes	116
8 Reflection	118
8.1 What I Learned	118
8.2 What I Would Change If I Did It Again	118

List of Tables

Table 1: Autonomous Lawnmower Weight Specifications	111
Table 2: Autonomous Lawnmower Dimension Specifications	111

List of Figures

Figure 1: Flow of Signal and Power within Power Supply and to Other Subsystems	102
Figure 2: Printed Circuit Board Schematic	103
Figure 3: DC Sweep of the Solar Path Signal	104
Figure 4: DC Sweep of the Wall Outlet Path Signal	104
Figure 5: DC Sweep of the Voltage Output that will be sent to the Charge Controller	105
Figure 6: Altium Generated PCB Top View	106
Figure 7: Altium Generated PCB Bottom View	107
Figure 8: Received PCB Bottom View	108
Figure 9: Received PCB Bottom View	108
Figure 10: 12V 100W Light Bulb	109
Figure 11: 12V 10W Light Bulb in 5V Circuit	110
Figure 12: 12V 5W Light Bulb in 3.3V Circuit	110
Figure 13: Power Supply Subsystem Overview Image	112
Figure 14: PCB Mounting Image	113
Figure 15: Solar Charge Controller Mounting Image	114
Figure 16: Power Supply Subsystem Overview Image from the side	115
Figure 17: Power Supply Subsystem Overview Image from the top	115
Figure 18: Voltage Output of New LiPo Batteries (5Ah - left, 8Ah - right)	116
Figure 19: New LiPo Batteries Connected	117

1 Subsystem Introduction

This subsystem consists of all the components that are used to power the Autonomous Lawnmower. This includes the PCB, in which there are two paths, one for signal and one for voltage. The signal path takes the input voltages from both the solar panel and the wall power plug and verifies if the solar panel voltage is between 6 and 15 volts using a window comparator. The output from this comparator is a signal that is used as the MOSFET gate, in order to effectively open or close the MOSFET as a switch. The voltage path is the input to the signal path, but are also to be input to the MOSFET source and is passed through in the according path dependent upon the decision made by the signal path. The voltage then flows through the corresponding MOSFET and diode, which is used to prevent backwards voltage flow, and to the PCB output. The PCB output is connected to the input of the solar charge controller, in order to ensure the battery is safely charged. The connection between the PCB and the solar charge controller is disconnectable in order to ensure that the lawnmower can disconnect from the system when it needs to operate freely. On the mower unit itself, is where the solar charge controller, battery, and connections to other systems will be. The charge controller is wired directly into the battery and the loads to the battery are also wired directly to the battery as some of our loads require very high currents, such as our motor driver which will require 7 Amps.

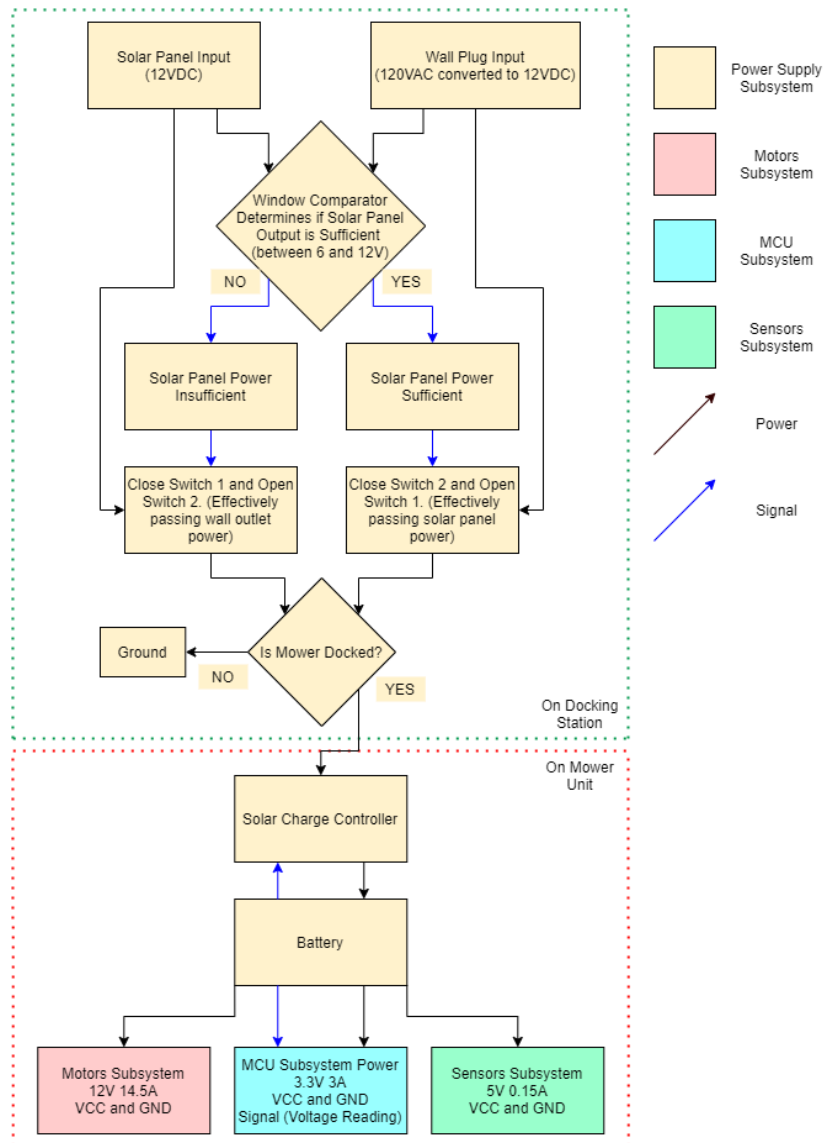


Figure 1: Flow of Signal and Power within Power Supply and to Other Subsystems

The above block diagram represents the major components of the power supply. The top box, represented by green dotted lines, is the portion of the power supply that will reside on the docking station. The bottom box, represented by red dotted lines, is the portion of the power supply that will be on the mower unit. Components on the PCB such as resistors are not included above. In order to simulate the loads I have acquired different wattage landscaping and car lights, which will be replaced by the other subsystems in our application moving forward.

2 Simulation

The first major step in the development of my power supply was the development of my power supply subsystem was simulating the circuit for the determination of which input to take.

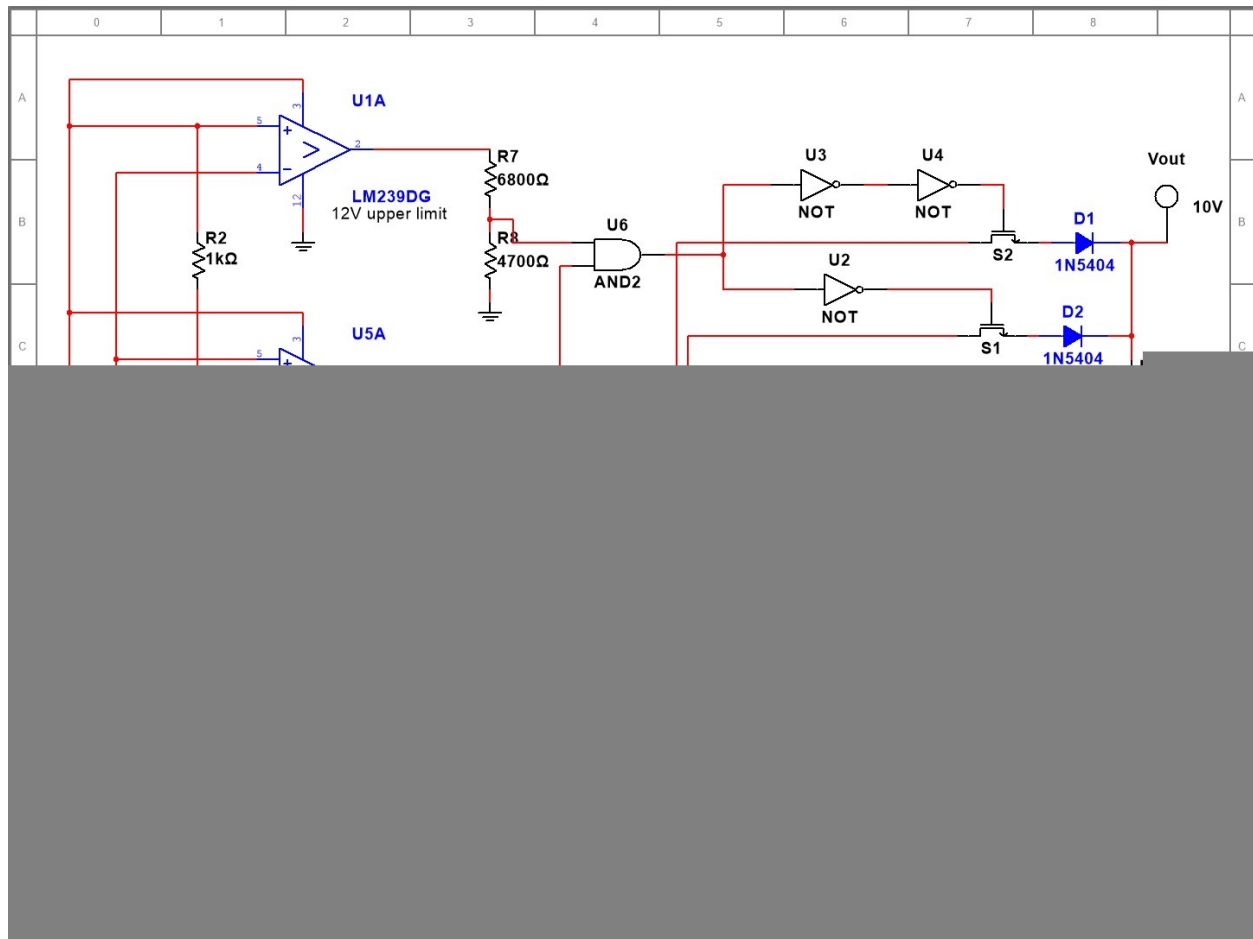


Figure 2: Printed Circuit Board Schematic

In theory, the window comparator should output a high signal for any input voltages in the range of 6V and 12V, exclusive. The comparator can be validated by running a DC sweep, in which the Solar Panel Voltage is varied from 0 to 15V, which is the standard range of a 12V solar panel. Figure 3 below confirms this by showing the solar power path has a high signal for any voltage ranging from 6 to 12V and a low signal for all other voltages. It correspondingly shows in figure 4 that the wall outlet signal is high for any voltage outside the 6 to 12V range and low signal for all other voltages

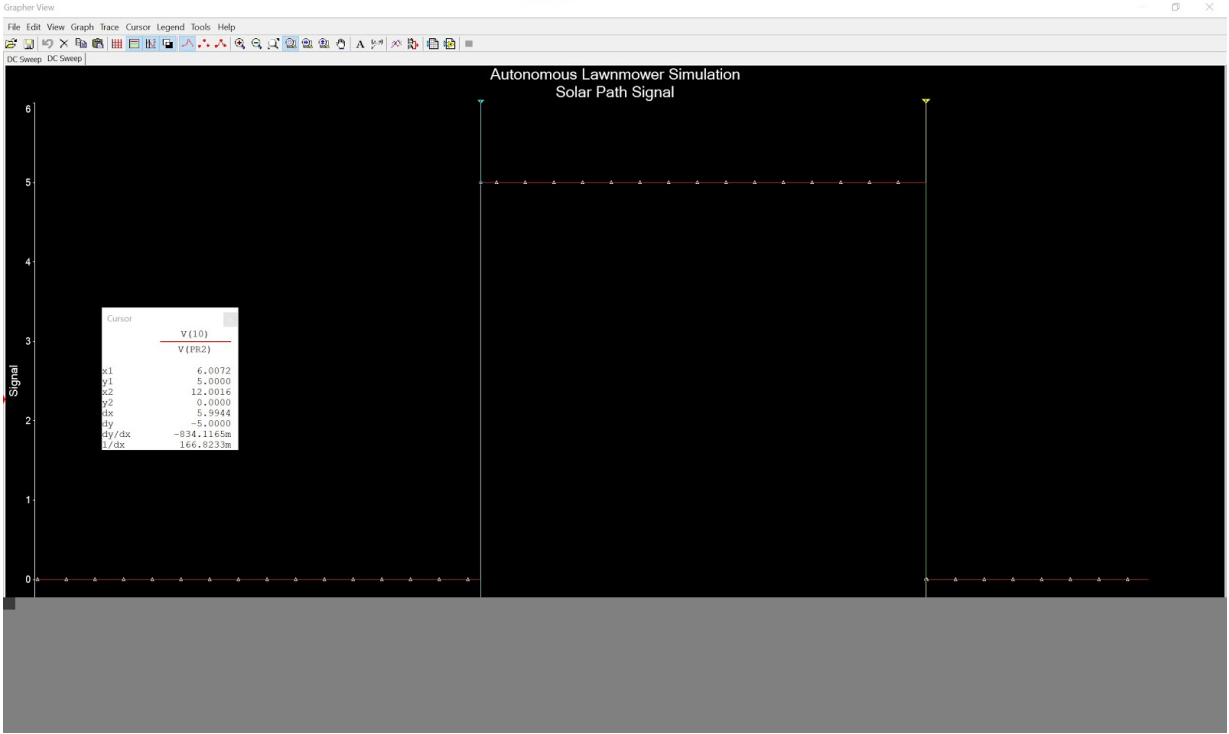


Figure 3: DC Sweep of the Solar Path Signal

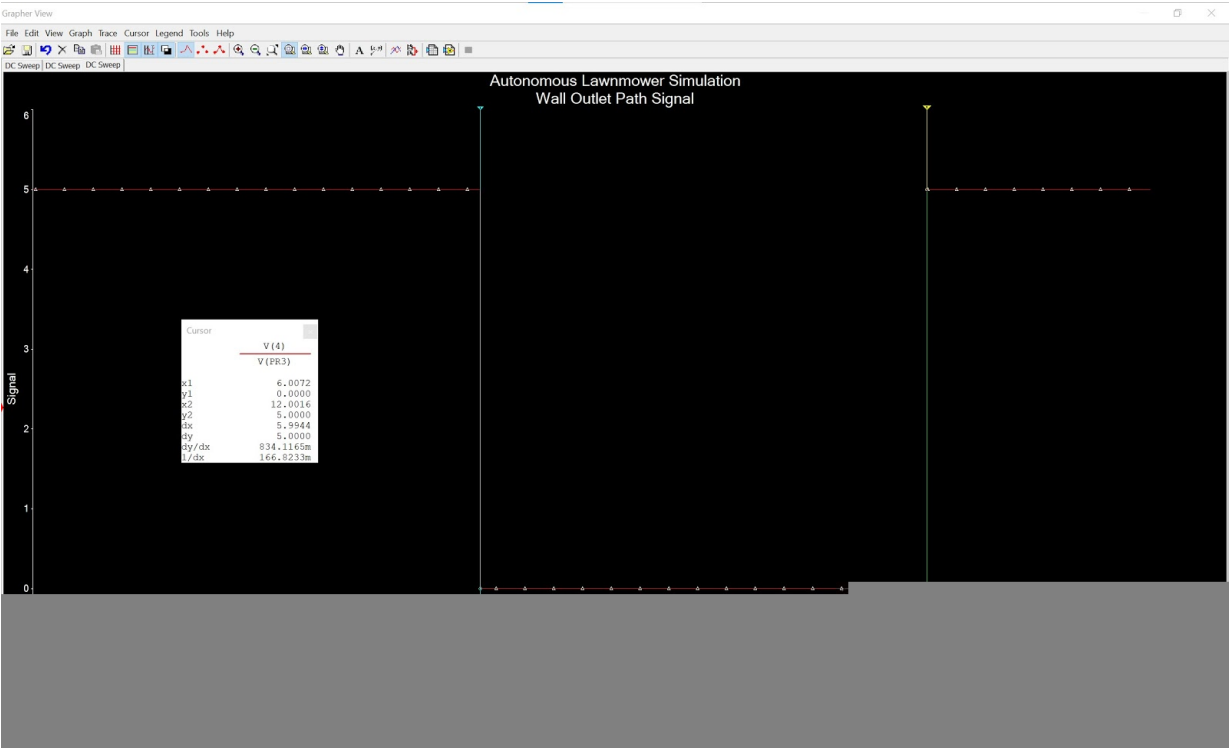


Figure 4: DC Sweep of the Wall Outlet Path Signal

These signals combined tell us which path will be supplying the power to charge the battery for a given voltage of the solar panel. The figure below shows the combination of these two signals, in which we can see the voltage output to the solar charge controller. The

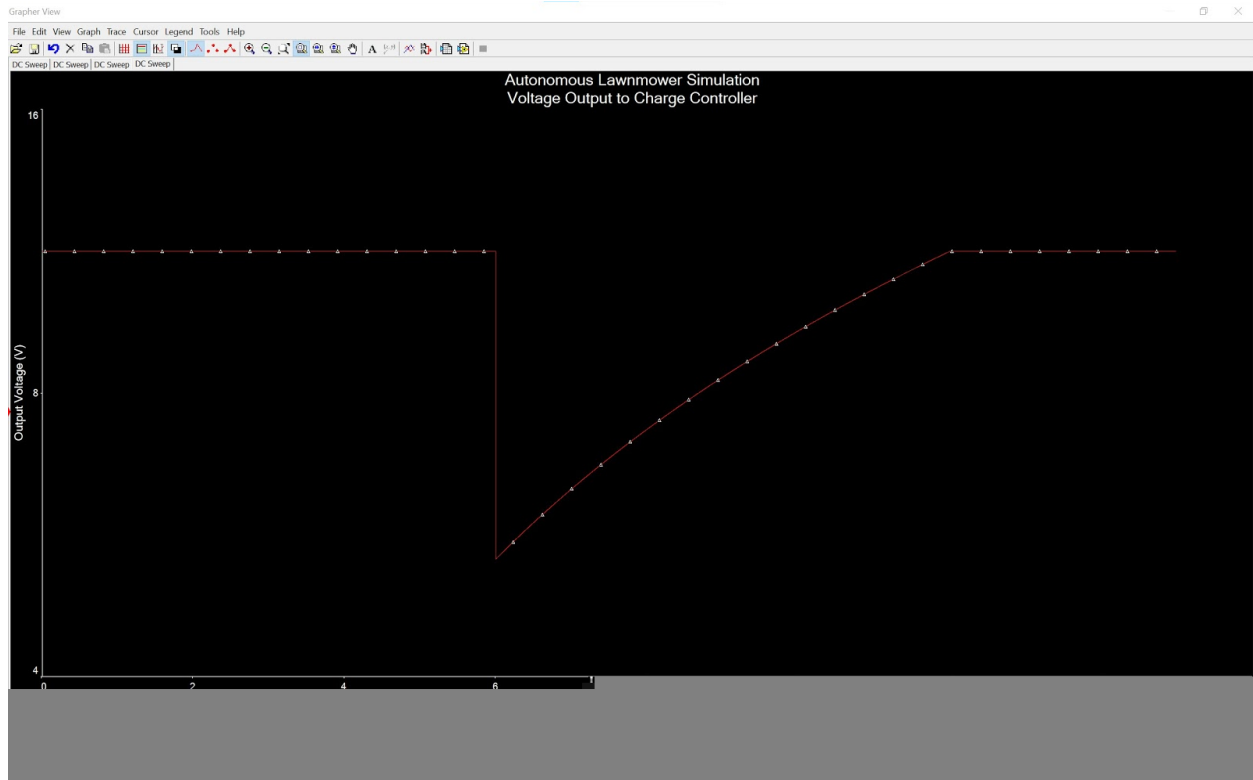


Figure 5: DC Sweep of the Voltage Output that will be sent to the Charge Controller

3 Printed Circuit Board (PCB)

The Power Supply PCB was designed using Altium Designer 20.2.5 as a 2-layer board, in order to reduce overall manufacturing costs. The PCB includes the two comparators that function as a window comparator, screw terminals for input and output ports, resistors, power MOSFETs, diodes, as well as AND and NOT gates used for signal flow. The following two figures show the top and bottom view of the PCB in black and white including the solder pads, traces, through holes, and silkscreen used for part labeling. The original PCB came from Advanced Circuits, but after initial evaluation, it was reordered from JLC PCB as they could produce it for far cheaper.

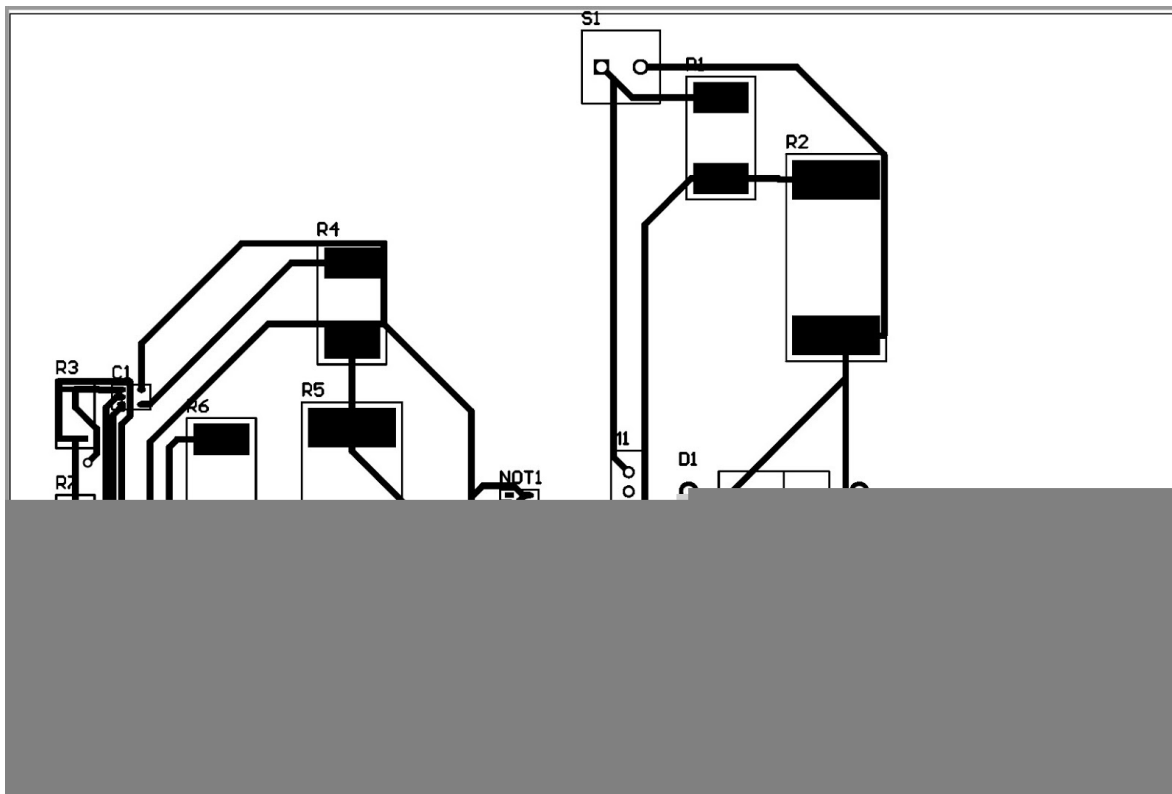


Figure 6: Altium Generated PCB Top View

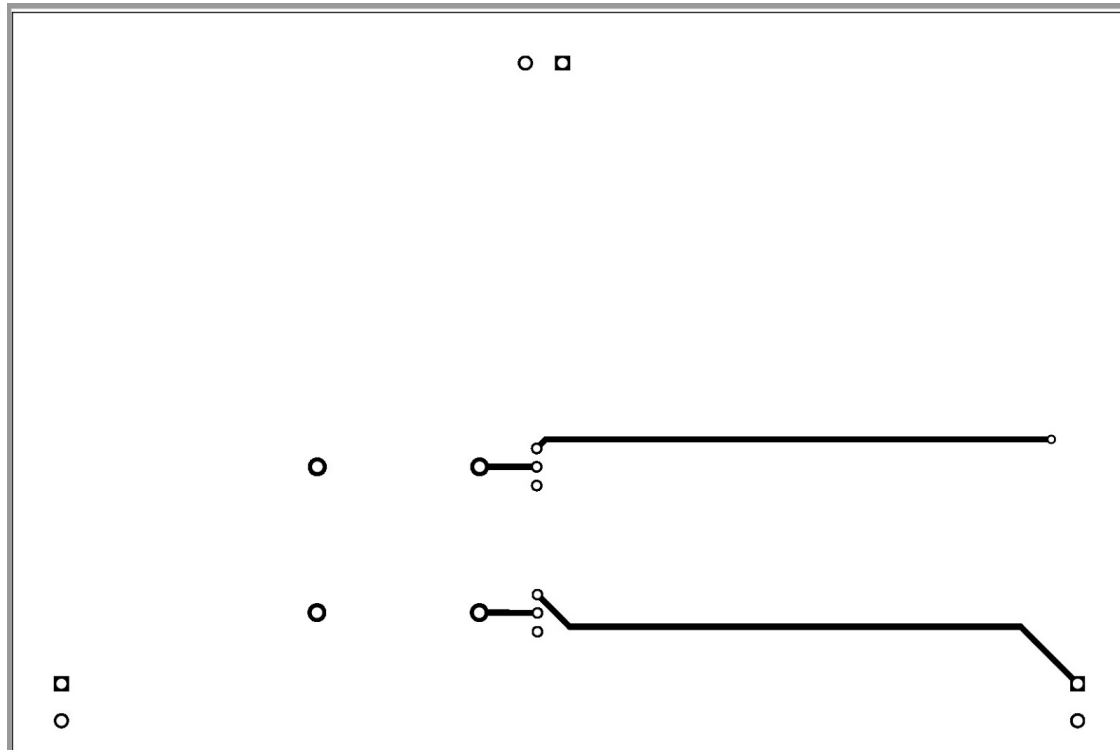


Figure 7: Altium Generated PCB Bottom View

3.1 PCB Evaluation

After receiving the initial PCB and evaluating it some issues became apparent. First, the hole sizes for both the through hole MOSFETs and diodes were too small as I had generated the hole size using the max hole diameter given in the datasheets, but they had been slightly larger than that, within the given percent error. The MOSFET holes also had been corrected to round holes when the pins are rectangular, and in doing so the hole took the diameter from the smaller dimension. Another thing that is worth mentioning but did not create any issues is the fact that some of the traces were right next to each other.

In order to fix these issues I reordered my PCB from a new vendor, JLC PCB, as they could produce the board in about the same time frame, shipping included, but for a much lower cost. They also had a minimum quantity of 5 boards per order, so I was able to have extras if need be as well as to practice soldering on. I enlarged the holes by taking the maximum hole size and adding 3 mil to the diameter for both the MOSFET and diode holes. I also changed my footprints for the MOSFETs to include round holes in which I made sure the diameter of the hole represented that larger dimension, in my case the pin width. Below is the top and bottom view of my final assembled PCB

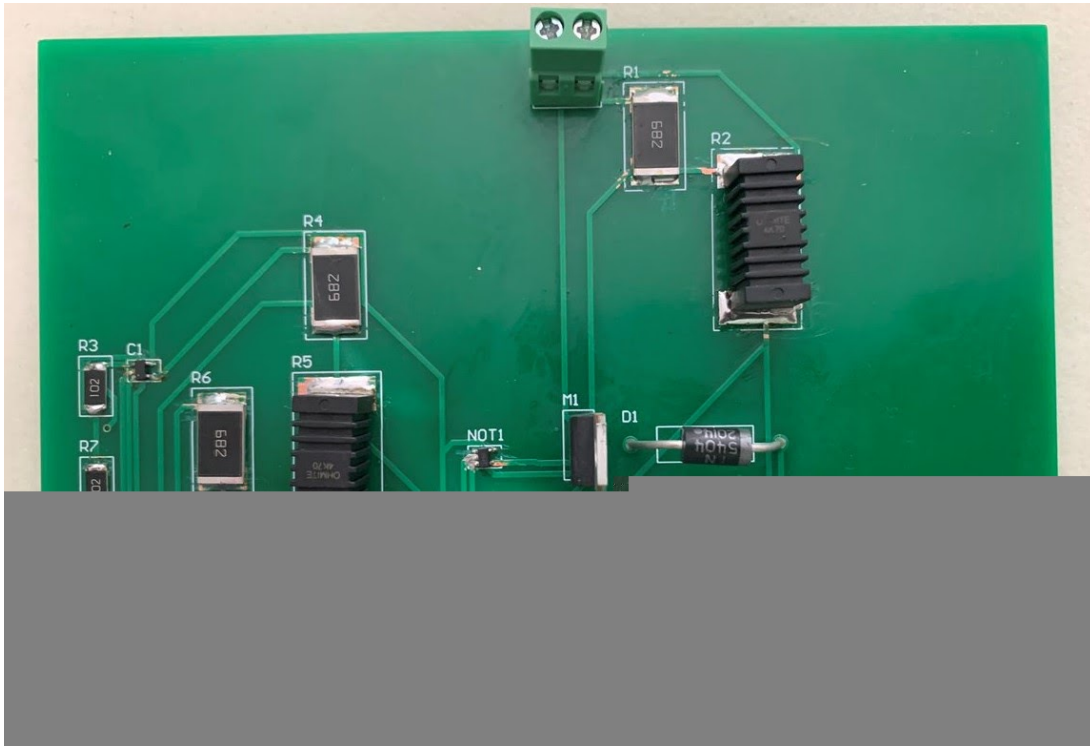


Figure 8: Received PCB Bottom View

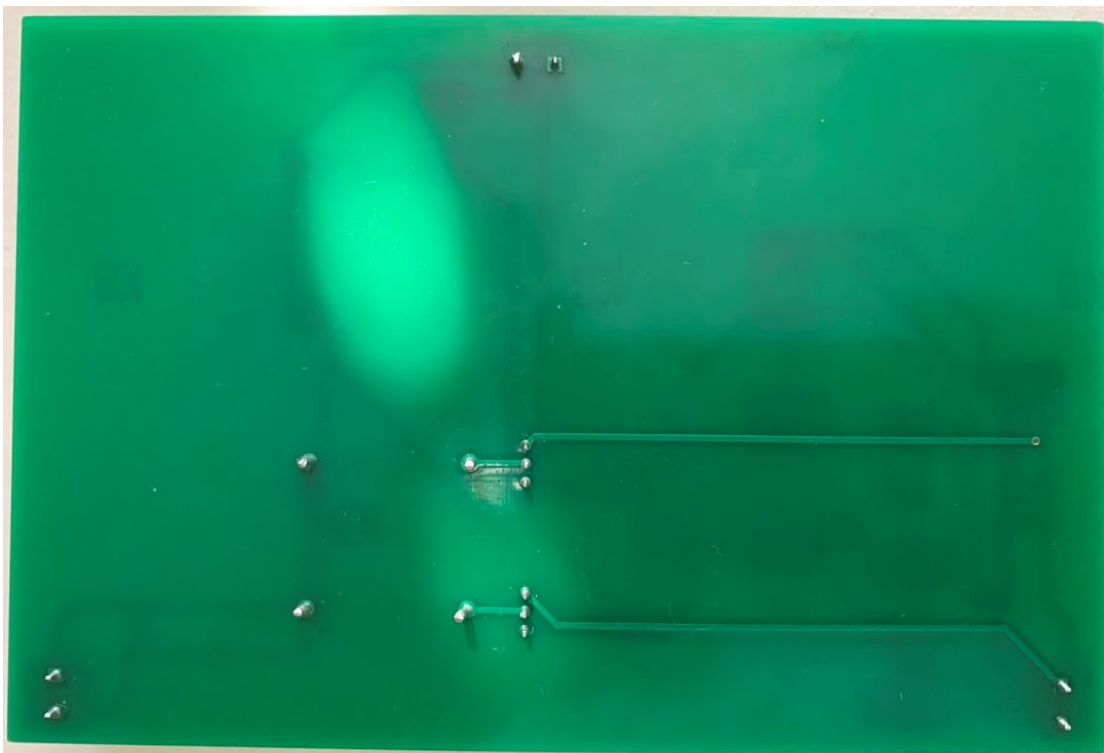


Figure 9: Received PCB Bottom View

4 Load Testing

In order to validate my battery size and load values, I had to find a suitable load to test my 3.3V 1A expectation for the Microcontroller Subsystem, my 5V 0.2A expectation for the Sensors, and my 12V 2.42A and 12V 7A expectations for each motor and the motor driver. In order to simulate these loads I used a 12V 100W car headlight to test the highest current level, as this headlight would draw 6A when tested which gives it a power draw of approximately 50W. I used a 12V 10W landscaping bulb in order to test the 5V system, and in this system this bulb draws 0.6A when tested, giving it a power draw is 3W. In order to test the lower voltage, the 3.3V at 1A I used another headlight bulb, this one rated at 12V 5W, it was outputting a power draw of 3.3A. This final bulb tested at 0.21A. The result of using a 12V bulb in both the 3.3V and 5V application is that the bulb does not reach its full luminosity. This can be seen as the lights are very dim, as shown in figures 9 and 10 below.



Figure 10: 12V 100W Light Bulb



Figure 11: 12V 10W Light Bulb in 5V Circuit



Figure 12: 12V 5W Light Bulb in 3.3V Circuit

5 Final Subsystem at the end of 403

Overall my subsystem consisted of quite a few parts that were obtained both from outside vendors, as well as from the lab room. In this section I will provide the size and weight specifications as well as a look at my assembled subsystem

5.1 Subsystem Weight at the end of 403

Component	Weight
Solar Panel	4.38 lbs
Solar Charge Controller	
Wall Plug	0.7 lbs
Battery	11.4 lbs
12V-5V Convertor	4.8 oz
12V-3.3V Convertor	30 g
12V 100W Light Bulb	0.05 lbs
12V 10W Light Bulb	0.03 lbs
12V 5W Light Bulb	0.02 lbs
Total:	16.946 lbs

Table 1: Autonomous Lawnmower Weight Specifications

5.2 Subsystem Dimensions at the end of 403

Component	Diameter	Length	Width	Height
Solar Panel	N/A	17 in	13.8 in	1.4 in
Solar Charge Controller	N/A	4.25 in	2.125 in	0.8 in
PCB (assembled)	N/A	4 in	6 in	0.7 in
Wall Plug	N/A	2.76 in	2.09 in	2.76 in
Battery	N/A	7.09 in	3.03 in	6.57 in
12V-5V Convertor	N/A	4.8 in	2.1 in	1.4 in
12V-3.3V Convertor	N/A	26 mm	36 mm	21 mm
12V 100W Light Bulb	0.45 in	0.71 in	N/A	N/A
12V 10W Light Bulb	1.375 in	1.625 in	N/A	N/A
12V 5W Light Bulb	N/A	3 in	1 in	4 in

Table 2: Autonomous Lawnmower Dimension Specifications

5.3 Subsystem Image at the end of 403

Below is an image that shows all of the components of my subsystem



Figure 13: Power Supply Subsystem Overview Image

6 Component Mounting

The previous section outlined where the power supply subsystem stood at the end of 403. In this section, I will discuss the progress made throughout 404.

6.1 Internal Mounting

The first task of 404 in regards to the power supply was to find a solution on how to protect the electrical components from prolonged exposure to elements such as rain, dust, and sunlight. In order to do this I first acquired a plastic box within which the components were mounted. The PCB was mounted onto the inside of the plastic box, as shown in figure 12, using #8-32 1-¼ inch bolts and #8 nuts, a ¼ inch spacer was used in order to provide separation for the backs of the throughhole components. The solar charge controller was also mounted onto the inside of the plastic box, as shown in figure 13, using ½ inch #6-32 bolts and #6 nuts.

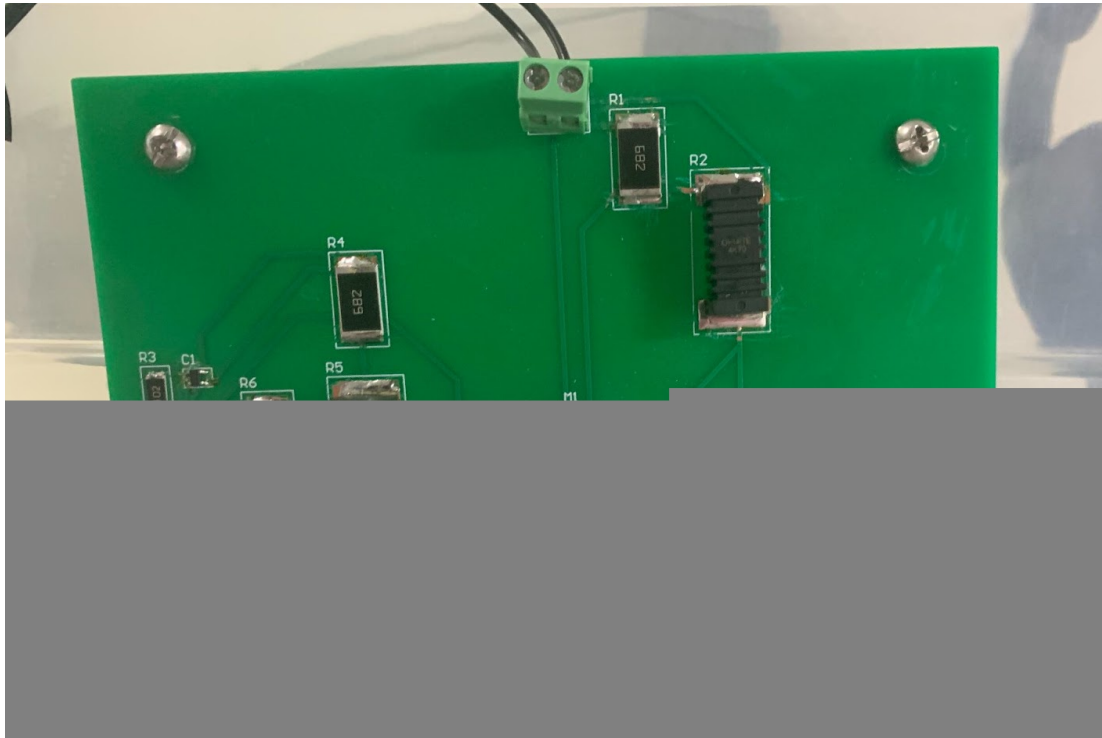


Figure 14: PCB Mounting Image



Figure 15: Solar Charge Controller Mounting Image

6.2 Wiring

In order to be able to power the electrical components mounted within the protective plastic box, wires had to be run through the box. These holes varied in size, with $\frac{1}{4}$ inch holes for the input of the wall plug and the output to the battery on the mower unit and a $\frac{1}{2}$ inch by $1\frac{1}{4}$ inch hole for the input of the solar panel.

6.3 Final Assembled Component Box

The box containing the electrical components mounted as explained in section 3.6.1 and wired as explained in section 3.6.2 can be seen in figures 14 and 15 below.



Figure 16: Power Supply Subsystem Overview Image from the side



Figure 17: Power Supply Subsystem Overview Image from the top

7 Unexpected Late Changes

Throughout the semester we encountered numerous issues with our motors subsystem. These changes, which included the installation of more powerful 250W motors and new more powerful 43A motor driver boards, are outlined in greater detail in the motors subsystem report. To compensate for the new motors and now motor driver boards, we needed a more reliable power source than we could get from a sealed lead acid battery. This resulted in the acquisition of two 11.1V Lithium Polymer (LiPo) batteries. These batteries operated at 5Ah and 8Ah, respectively. In order to maximize the power into the new motors, we connected the two batteries such that we would have one 12V output and one 24V output. The 12V output powered the power block which was connected to the PIC32 microcontroller, the ESP32 microcontroller, the GPS, the gyroscope and accelerometer, and the blade motor. The blade motor remained the original 30W motor specified during 403. The 24V output powered only the new 250W motors and the new 43A driver boards. Figure 16 below shows the voltage output of the two LiPo batteries when connected and figure 17 shows the two batteries connected to the system through the 12V and 24V outputs. Also shown in figure 17, in the top is the makeshift bin (heh tupperware), which contains the 12 to 3.3V and 12 to 5V converters, as well as a bus bar to distribute the 3.3V and 5V power levels to multiple devices.

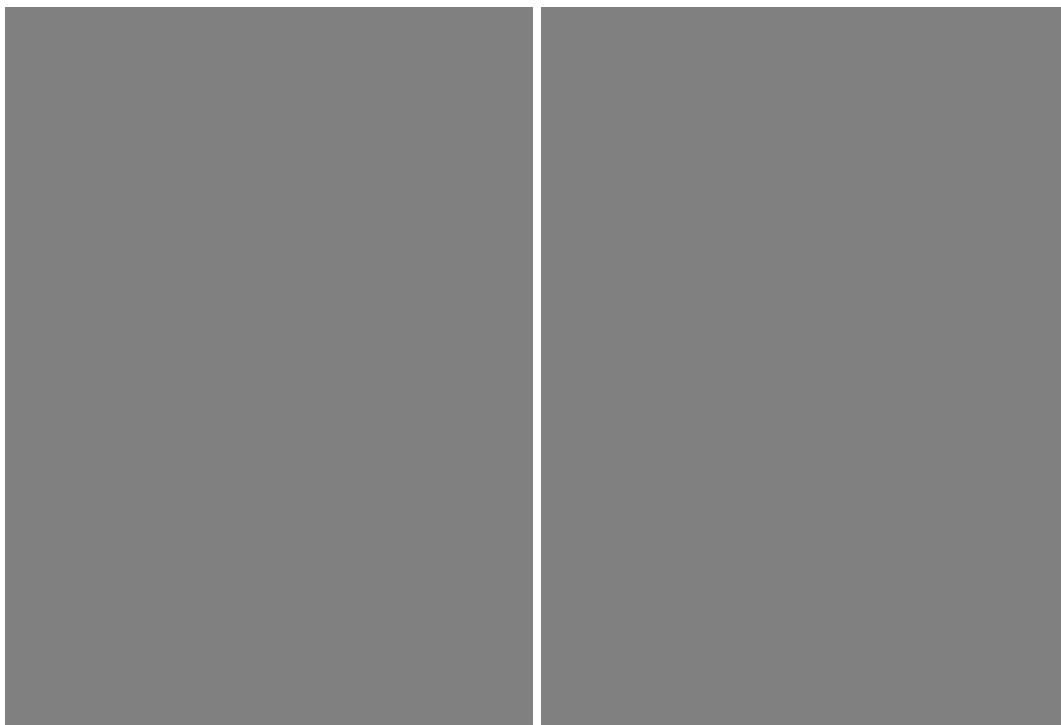


Figure 18: Voltage Output of New LiPo Batteries (5Ah - left, 8Ah - right)



Figure 19: New LiPo Batteries Connected

8 Reflection

Back in August, when it was decided that we would be doing the autonomous lawnmower project, and then subsequently that I would be designing the power supply, I had very little idea about how to implement it. My first approach was to simply utilize a solar charge controller and panel that would have stayed on the mower at all times. After further discussion and research, it was determined that I would take two different power sources and utilizing a PCB determine which voltage source to use. In order to do this I would use a voltage comparator.

8.1 *What I Learned*

In August, I had never even heard of a PCB, much less Altium Designer. Through guidance from Professor Lusher as well as the resources of the world wide web, I was able to not only discover the endless possibilities presented by printed circuit boards, but I was also able to figure out how to use the software to implement them. Although this course does not use the entirety of the services Altium Designer has to offer, I was still intrigued to learn about things like the bill of materials, which are utilized in practical applications by numerous corporations. Another thing that is an afterthought to most when it comes to a physical project like the autonomous lawnmower is the action of connecting different components. Prior to this project I had never heard of a crimper, or attempted to interconnect electrical components using anything other than breadboard jumper wires. This course allowed me to get hands on experience in wiring and system integration that I would otherwise not have had.

8.2 *What I Would Change If I Did It Again*

Looking back now at the subsystem I designed, I believe that there are changes that could be made in order to improve the way I approached this problem. First, rather than trying to design the PCB to determine which voltage source to use like I did, I believe it would've been more beneficial to manufacture a single PCB that contained 12V to 3.3V and 12V to 5V convertors. By focusing on a single PCB by which I had two down convertors, I feel like I could've been much more efficient, as the process would have been more or less the same for each, just with different components. I also feel like this would have been a more practical application, as voltage convertors are used in everyday life, whether it be the plug you use to charge your iphone, or components within your iphone.

Autonomous Lawnmower

Max Lesser

MCU SUBSYSTEM REPORT

REVISION – 3
29 April 2021

Table of Contents

1 Subsystem Introduction	122
2 Multi MCU interface	123
2.1 Interface Descriptions	124
2.2 As Built System	126
3 PIC MCU by Max Lesser	127
3.1 Hardware	127
3.2 Software	130
3.3 GPS	133
3.4 Ultrasonic Sensors	137
3.5 WIFI-Module	142
3.6 Drive Motor Control	142
3.7 Blade Motor Control	142
3.8 Accelerometer and Gyroscope Unit control	143
3.9 Battery Status Interface	143
3.10 MCU Functions Relegated to and Dependencies on Other Subsystems	144
3.11 PIC-32 MCU Subsystem Conclusion	145
4 ESP-N Subsystem by Max Lesser	146
4.1 Introduction	146
4.2 Navigation	147
4.3 Obstacle Evasion	148
4.4 Shaft Encoders	149
4.5 Accelerometer Gyroscope and Magnetometer	149
4.6 Motor Control	149
4.7 Blade Motor Control	150
4.8 Summary and Conclusion	150
5 ESP-W Subsystem	151

List of Tables

Table 1: UART Data Transfer Information	124
Table 2: GPS precision test results	133
Table 3: Distance and Error Percentage for small object detection	140

List of Figures

Figure 1: MCU Block diagram	123
Figure 2: PCB layout	128
Figure 3: General Program flow	131
Figure 4: Serial Output from PIC-32 towards ESP-N, with front and side obstacle readings, as well as parsed GPS output	132
Figure 5: GPS precision test result map	134
Figure 6: GPS accuracy test map	135
Figure 7: GPS accuracy test error to Google maps graph	135
Figure 8: GPS accuracy test Error between measurements graph	136
Figure 9: Ultrasonic sensor detection of large object test result graph	138
Figure 10: Percent Error for Large Object Detection	138
Figure 11: Ultrasonic sensor detection of small object test result graph	139
Figure 12: Percent Error for small object detection	140
Figure 13: Ultrasonic sensor detection of person test result graph	141
Figure 14: Percent Error for Person detection	141
Figure 15: Battery read circuit simulation for "Battery Charged" (t) and "Discharged" (b)	144
Figure 16: General Navigation flow without obstacle evasion	147
Figure 17: Obstacle Detection Flow Chart	148

1 Subsystem Introduction

The MCU subsystem consists of the Hardware and most of the software to control the Autonomous lawnmower. Due to issues with the PCB manufactured during 403 for the PIC-32, the new MCU subsystem consists of the PIC-32 from 403, and a ESP-32 that was added during 404. The PIC-32, abbreviated to PIC here, performed obstacle detection and GPS reading and Parsing, and relayed this data to the ESP-32. The ESP-32 consists of 2 components, Navigation and Networking. The Navigation component, ESP-N for this report, is responsible for reading information from the PIC-32 and controlling motors for navigation and obstacle avoidance. The Wifi component of the ESP-32, referred to as ESP-W from here on, is responsible for connecting with the User interface and obtaining scheduling and navigation information. This Section of the report will be split into 4 parts. Multi MCU interface, PIC, ESP-N and ESP-W. Beginning with the MCU interface, we will below present how the 2 MCU are interfaced.

2 Multi MCU interface

This project required 2 MCUs as we encountered issues with the PIC hardware, which will be elaborate upon at the end of this section. Reproduced below is the System diagram, which shows the interface between the 2 MCUs, along with their connections. The PIC was connected to the ESP via UART, as well as 2 signals wires used to inform the ESP of obstacles. The final, as built implementation, had an interface line for front obstacle detection and the UART line, but not the side obstacle detection signal wire. While the UART line was connected, it was not ultimately used. Refer to the “As built” section for detailed explanation.

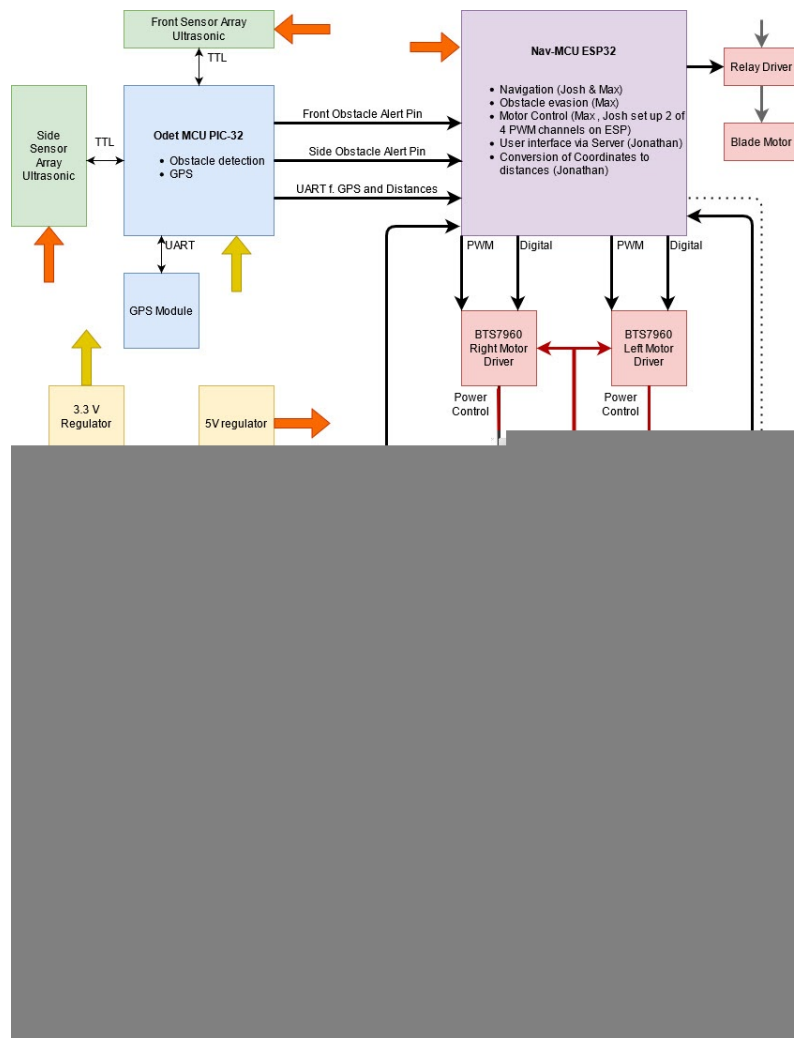


Figure 1: MCU Block diagram

2.1 Interface Descriptions

The 2 MCUs were interfaced with each other on the mower, as well as with other systems on and off the unit. This section will present a brief description of the interfaces used and their purpose

2.1.1 PIC to ESP Interface

The Pic is capable of relating information about obstacles as well as GPS to the ESP. This is achieved using UART for GPS and detailed obstacle information. As well as GPIO connections as an alert signal

2.1.1.1 UART Interface

The UART interface used between the MCU was configured as 9600Baud, 8 data bits, 1 stop bit and no parity. This interface would send text containing detailed information, as listed in the below table

Purpose	Information	Format
GPS	Latitude, Longitude, Course and Speed	[Ddd.ddd, dd.dddd, ccc.cc, s.ss] where the first 2 entries are latitude and longitude, then course and speed in knots
Front Obstacle array	Minimum distance to obstacle, side of Mower obstacle is on	[ddd, S] where ddd is distance in cm, and S corresponds to side of mower the object is on (from mowers perspective)
Side Obstacle array	Minimum distance to obstacle, angle obstacle makes with mower	[ddd, aa] where ddd is distance to obstacle in cm, and aa is angle the obstacle makes with the mower

Table 1: UART Data Transfer Information

This output was set in the PIC using UART print statements, where each data field was delimited by a comma, and entries ended with a carriage return/ Line Feed. Since all information was printed from the PIC, data formats or delimiters could easily be changed as needed.

Physically the interface consisted of the UTX and URX line, even though information only ever flowed from the PIC to the ESP.

2.1.1.2 GPIO Interfaces

In addition to the above UART interface, the PIC and ESP shared 2 wires for a GPIO interface. These wires were intended as an “alert interface” where the PIC would set a pin high if an obstacle was within a certain distance of the mower, and ESP could immediately respond by stopping, without having to wait for a UART message to be received. One wire each was assigned to the front obstacle array, and the side obstacle array. The Pins were read on the ESP via ISR, and by direct sampling of the appropriate pins. Obstacle alert thresholds can be set within the PIC to adjust for conditions.

2.1.2 PIC Interfaces to Other Systems

The PICs primary data interfaces were with the GPS module and the 2 Sensor arrays. The GPS sent information using a UART connection with the same configuration as the PIC-ESP UART connection. The Sensor arrays used GPIO pins.

2.1.2.1 UART interface

The GPS and PIC were connected using 2 wires, for UTX and URX. The Pic would send configuration information to the GPS, and the GPS would echo received NEMA sentences back to the PIC.

2.1.2.2 GPIO interfaces

Ultrasonic sensor operation is described in detail in the appropriate PIC section. The interface between the PIC and these sensors consisted of one trigger signal per array, for a total of 2. And an echo signal for each sensor, for 5 from the front array and 2 from the side

2.1.3 ESP Interfaces to Other Systems

In addition to the pic interface the ESP connected with sensors and motor controllers for the navigation logic, as well as WIFI to facilitate the user interface.

2.1.3.1 Sensor Input

The ESP-N interfaced with 2 Shaft encoder sensors. For each sensor the ESP received 2 signals, one indicating complete wheel rotation, the other indicating partial wheel rotation. Details are outlined in the appropriate ESP-N section. These 4 wires connected the GPIO pins on the ESP, which were configured as interrupts. Allowing us to track asynchronous changes

2.1.3.2 Drive Motor Control

Drive Motors were controlled using 4 wires per motor, of which 2 are PWM signals and the others are digital logic signals. This made for a total of 8 connections to the ESPs GPIO pins to control drive motors

2.1.3.3 Blade Motor Control

The Blade Motor relay was connected to the ESP via Output GPIO pin.

2.1.3.4 User Interface

The ESP-W portion of this MCU connected to the Network and User using the ESPs integrated WIFI capacity, and a local WIFI network, which was created using a mobile hotspot during testing and demo.

2.2 As Built System

Due to Time constraints at the end, some components of the MCU subsystem were not fully implemented, including:

- Side obstacle detection
- GPS based Navigation
- Mower statistics for battery, runtime or distance
- User specified navigation

The underlying systems for the above components function individually, however they were not implemented on the system as demoed, as time needed to include these was used to rebuild the Motor and Navigation subsystem.

Obstacle evasion was implemented using just the front array alert signal elaborate above, and navigation was handled using inertial guidance. Distances for the area to be mowed were hard coded in the code.

3 PIC MCU by Max Lesser

The PIC32 was originally intended as the only MCU, but due to hardware problems it was later used for only GPS and obstacle detection.

3.1 Hardware

3.1.1 Hardware Summary

The primary components of the MCU board are a Microchip PIC-32MZ, a Microchip RN1810 WIFI module and a SIM33EAU GPS module mounted on a Parallax 28504 breakout chip. The board additionally includes capacitors to smooth out power supply ripples, resistors, as well as 2 and 3 pin screw terminals and pin header connectors to allow connection to other subsystems. These components are situated on a PCB, custom designed in Altium and printed by Advanced circuits. During testing the RN1810 WIFI module sustained damage and became inoperable. Additionally and oversight in the PCB design resulted in insufficient PWM pins for motor control. This issue was later exasperated by changed Motor drivers. Due to the Small pitch of the PIC-32 package fixing the board was deemed not possible. As a result we switched to the 2MCU approach, as outlined above.

3.1.2 Hardware Selection

The Pic-32MZ-2048EFH-144 Microprocessor was specifically recommended by prof. Lusher. I followed his recommendation as Pic Microcontrollers have a host of different IO lines, and can operate in a wide range of environmental conditions. Additionally, Pics are small, lightweight and consume little power, making physical and power integration easier in our battery-operated system. Finally, PIC microcontrollers are inexpensive, which is a consideration as we are budget constrained.

The RN1810 WIFI module was selected based on research, as it interfaces easily with PIC processors since it is also a Microchip product. Additionally, it is small and inexpensive, and easy to integrate on the PCB.

The SIM33EAU GPS module on the Parallax breakout board was chosen in part for its breakout board, allowing us to mount it independently of the MCU without need for additional PCBs. Finally, it's stated performance of 2.5M accuracy and ability to receive signals from several different satellite constellations made it an attractive option.

After using this hardware setup through 404, I still agree with the above, but have come to find that firmware development on the PIC platform is rather advanced and time intensive. Given the host of other issues I needed to address during this project I now believe a simpler, easier to use Embedded platform would have been the wiser choice. While this project could have been realized only with a PIC controller, the workload of PCB design, Firmware development and unrelated integration concerns and issues with other subsystems proved too much to handle. For future iterations I would recommend using either a development board to eliminate PCB design concerns, or possibly an easier to use platform, like the ESP-32 added later during the project.

3.1.3 PCB design

The MCU PCB was designed in Altium as a 2-layer board, to reduce manufacturing cost. The board includes the above-mentioned components, as well as incidental capacitors, resistors, and screw terminals needed for IO. An Additional UART port was included for demo and debug purposes, as well as an access point to the WIFI- MCU UART connection. A Glitch in the Altium software caused us to lose the PCB layout that was ultimately printed. For illustrative purposes we show the last saved layout version, which differs from the printed versions in 2 2-pin terminals for UART output, mounting holes, and slight adjustment of components for clearance.

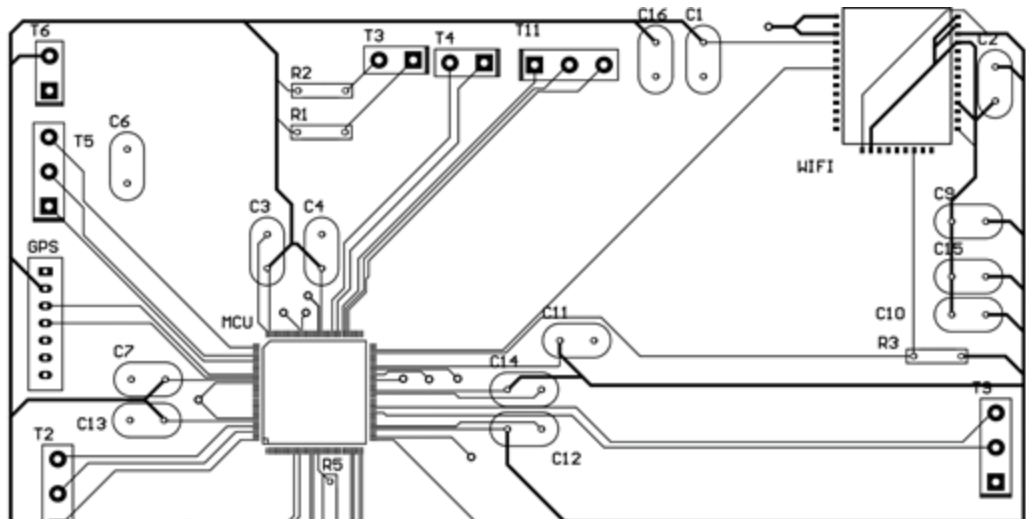


Figure 2: PCB layout

3.1.4 Hardware Faults and Failures

After Evaluating the Received circuit board some issues became apparent. Terminal T-2 in the above schematic was connected to the wrong MCU pin, preventing us from sending PWM through this terminal. Footprints for GPS and Debug interface (T12) were too small to fit the intended header pins. Jumper wires were soldered in place instead and allowed us to connect. Additionally, the MCUs Reset Pin was not connected to the boards debug port properly, the reset interface instead had to be connected to a resistor leg. Finally, pullup resistors to the WIFI modules wakeup pin were missing. This is now irrelevant however as the WIFI module sustained damage from a power supply failure during testing and is not operational.

Improper Debug interface and footprint, along with GPS footprint were overcome at assembly time of the board. Incorrect pin connection to T-2 motor output may be resolved by trading PWM signal with another pin that is PWM capable. WIFI capability is currently being implemented on breadboard, using an ESP-32 module, connected to the MCU via the UART line going to the defect WIFI module. This configuration can be switched from breadboard to Perf-board and implemented permanently next semester if desirable. The sensor subsystem has additionally been made aware of a need for more Ultrasonic sensors, to allow for better obstacle detection. Taking all of this into consideration it may become necessary to print another PCB correcting the mistakes made here and accounting for new insights.

The above issues encountered at the end of 403 were ultimately overcome by using the ESP-32 as primary MCU. The additional Ultrasonic sensors and new motor drivers added more IO than any single MCU at our disposal could provide while also accommodating the rest of the system. Since WIFI was already implemented on the ESP and it had the ability to output PWM without the need for additional jumper wires as the PIC would require, it was decided to use it as the main MCU and use the PIC for Obstacle detection and GPS reading and parsing.

3.2 Software

3.2.1 Software Overview

The Firmware for the PIC-32 MCU was developed in Microchips MP-LabX IDE with Microchip Harmony V3 configurator. Harmony allowed us to graphically configure the MCU for clock rate, Peripheral components and pin mappings. C language code was written in MP-Lab to implement the MCU functions described below. Through the MP lab IDE and PIC-Kit 4 in circuit debugger we were able to test the code piece by piece, in the target device and monitor internal variables and outputs. This proved invaluable in debugging and testing the program.

3.2.2 Configuration of MCU

A detailed overview of what MCU components are used to implement different functions is provided in function sections. The Pic has 9 connections for ultrasonic sensors, 7 for the echo signal and 2 to trigger the arrays. It connects to the GPS via UART, and has 1 UART and 2 GPIO connections for interface with the ESP-32. The Pic also needed an internal timer.

3.2.3 General Program Flow

As the Pic was relegated to Obstacle detection and GPS reading, its program flow is rather simple. It reads the front sensors array and parses the data for an obstacle, then reads the GPS unit and parses the received string for relevant data, and finally reads the side sensor array. This process continues while the PIC is powered on.

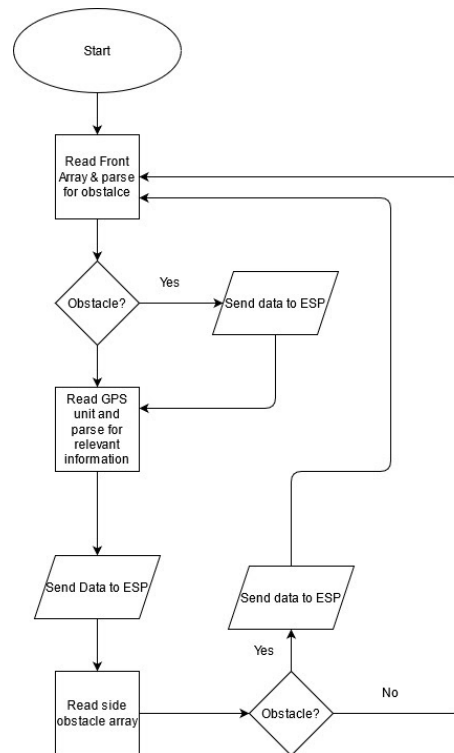


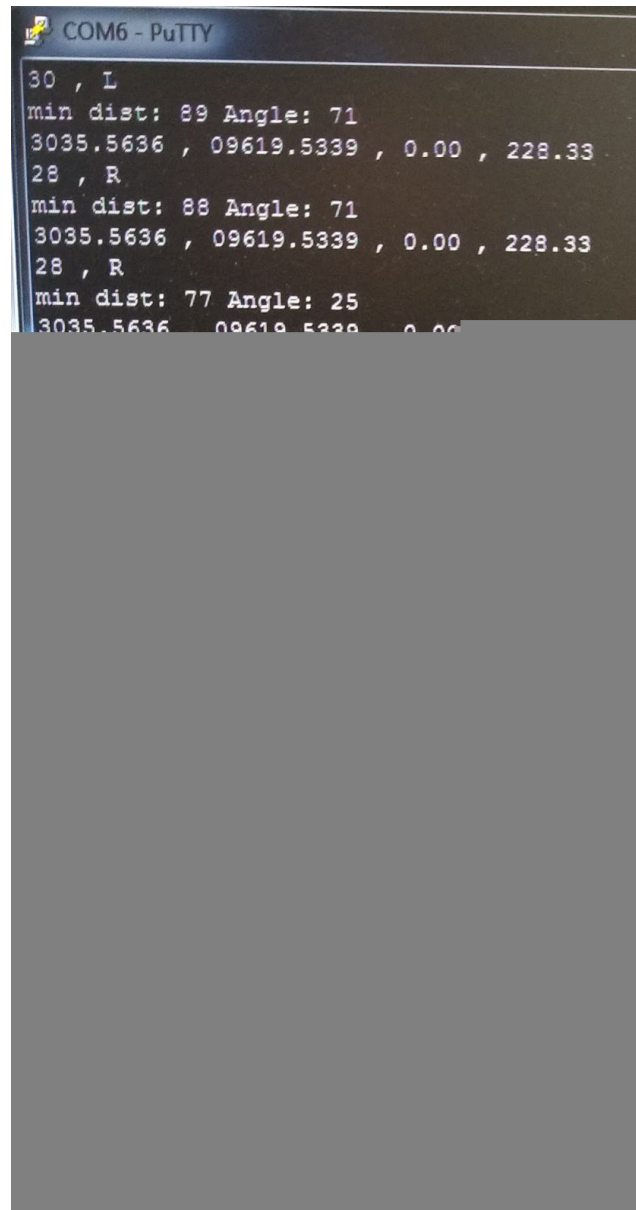
Figure 3: General Program flow

3.2.4 MCU Code Example

The Code written for the PIC-32 is provided in the Github repository under DemoedCode/PIC. The full PIC project, including configuration files is also provided in this directory for completeness.

3.2.5 Sample PIC Output to ESP-N

Presented below is a sample UART Output from the PIC to the ESP-N. The output includes Minimum distance to an object in front of the mower, in (distance, side) format. GPS reading of Latitude, Longitude, speed and course, and side array reading of minimum distance and angle the object makes with the side of the mower. Note that I had to sit in front of the side array for most of this test, hence many of its readings are 0. (as discussed in the Ultrasonic sensor section, readings against fabric do not return accurate results). This text is still formatted for human understanding, but could have easily been formatted in other ways. Unfortunately difficulties with other subsystems prevented us from using most of this information.



```
COM6 - PuTTY
30 , L
min dist: 89 Angle: 71
3035.5636 , 09619.5339 , 0.00 , 228.33
28 , R
min dist: 88 Angle: 71
3035.5636 , 09619.5339 , 0.00 , 228.33
28 , R
min dist: 77 Angle: 25
3035.5636 , 09619.5339 , 0.00 , 228.33
```

Figure 4: Serial Output from PIC-32 towards ESP-N, with front and side obstacle readings, as well as parsed GPS output

3.2.6 Comments on Program Performance

During the 403 demo the PIC code experienced some tumbles, where it would run through a bunch of statements without executing them. Later during 404 I encountered many exceptions being thrown by the pic for “Instruction Fetch or address load error expectation”. The ultimate cause of these exceptions is not known, however extensive research revealed interrupts on the PIC to be a possible source. After disabling all interrupts the exceptions greatly decreased in frequency. Building and deploying the project in Debug mode further helped. Several stamina tests were conducted during which the code performed without exceptions for 60+ minutes.

3.3 GPS

3.3.1 GPS Summary

The GPS receiver is mounted on a breakout board with male header pins. These connect to a wiring harness leaving the MCU, to allow mounting of the GPS in a location with clear reception. Communication takes place through the MCUs UART3_TX and UART3_RX lines, at 9600 Baud, 8 data bits, 1 stop bit and no parity. The GPS transmits data per NEMA0183 protocol. Upon startup the MCU sends a configuration string to the GPS module, setting it to send only RMC type sentences. The MCU then reads in the received characters and parses them for the needed data, specifically Latitude, Longitude, course and speed over ground. This information can then be accessed by other functions as needed.

3.3.2 GPS Test Results

Two different types of GPS tests were performed. One for consistency, and one for accuracy at different locations. The consistency test was presented in the final update presentation and showed that the GPS module reads identical data within one power cycle, and slightly different data after being power cycled. The second test was conducted at 4 different locations where we took two readings each and compared it with google maps results. This test revealed that accuracy is within the tolerances stated in the data sheet and google maps, however data from both tests combined suggests that the GPS module may be susceptible to noise and interference.

3.3.2.1 Precision Within and Between Power Cycles of GPS Module

For the Final presentation we have demonstrated the GPS modules ability to generate consistent readings within one power cycle. For this purpose, we power cycled the device, waited for the GPS module to acquire a fix, and then read the position 5 times. All 5 readings within one power cycle provided the exact same coordinates, speed and heading. This test was done on the Demo board, but as the accuracy and precision of readings rely on the GPS module only, which is the same module used with the target board, we present this test here. The readings were taken with the GPS module placed inside my apartment, as the system is still heavily reliant on external infrastructure, such as power supplies and multiple laptops. A summary of this test is presented below. The full test data for all 25 readings, to show precision, is presented in Appendix C.

test#	# of readings	Latitude	Longitude	speed	course		Distance to Google Maps point
1	5	3035.5572	9619.538	0	84.71 Pic32	Red	10.75M
2	5	3035.5613	9619.533	0	127.35 Pic32	purple	10.1M
3	5	3035.5583	9619.5307	0	237.76 Pic32	Blue	5.6M
4	5	3035.5612	9619.5342	0	156.33 Pic32	yellow	2.5M
5	5	3035.5629	9619.5344	0	359.03 Pic32	light blue	4.7M
control data							
Control	1	3035.5656	9619.5317		TM-D710GA	Dark Green	
Control	1	3035.5617	9619.5333		Google Maps	Green	

Table 2: GPS precision test results



Figure 5: GPS precision test result map

3.3.2.2 Accuracy of GPS Module at Different Locations

To test accuracy on the target board we took 2 readings from the GPS module in 4 different locations and found the distance between the 2 readings and to the google maps determined location. This test was conducted at my apartment complex, with the MCU board and GPS module placed on the hood of my car. The engine was off during the 1st test and running for the last 3. In the figure below, red represents the google maps location, while the 2 readings at each location are shown in blue. Distances to the google maps point and between the measurements are shown in tables below. We point out that according to google support the accuracy of the google maps for GPS positioning is around 20M, while the GPS module used gives accuracy at 2.5M in the datasheet. The tests with running engine exhibit differences between repeated measurements, in excess of the stated accuracy, while the test with engine off does not. Additionally, all points are within 10M of the google maps provide coordinates, staying within the given tolerances.



Figure 6: GPS accuracy test map

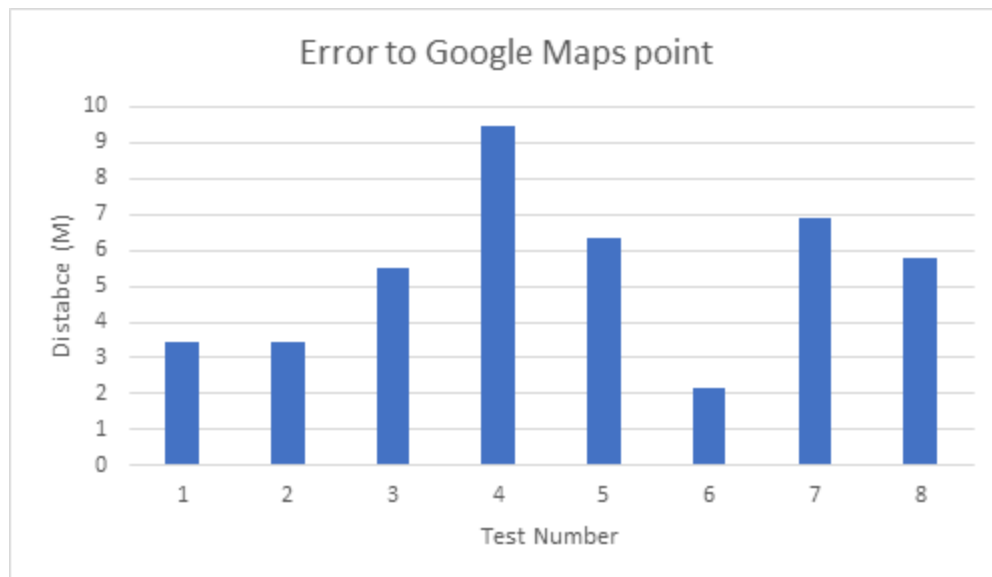


Figure 7: GPS accuracy test error to Google maps graph

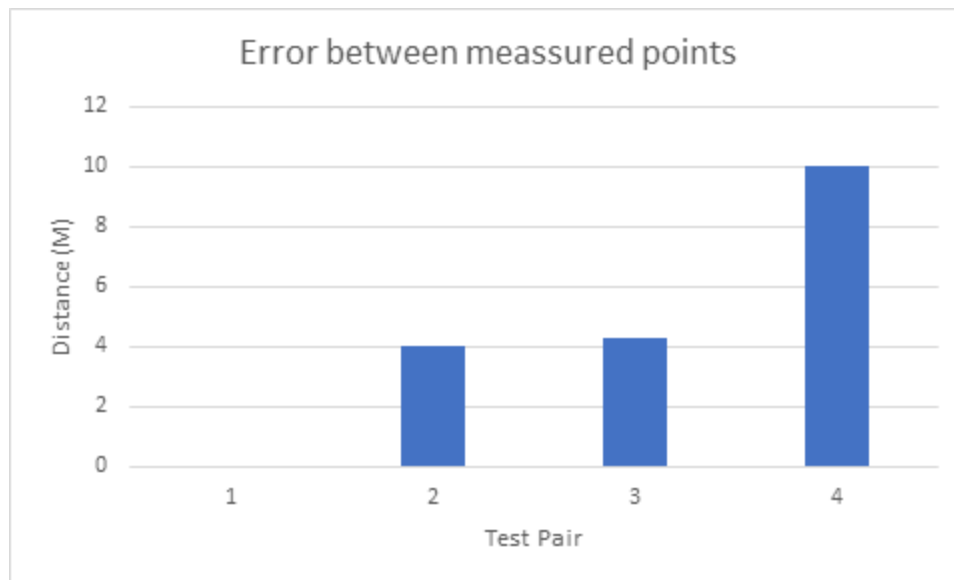


Figure 8: GPS accuracy test Error between measurements graph

3.3.3 GPS Test Summary

The 2nd series of tests exhibits errors of up to 10M for repeat measurements of the same location, the 1st test in this series, with the motor off, exhibits no error for repeat measurements. Additionally, 5 sets of 5 repeated measurements have been taken in section 1.2.4.2.1, none of which exhibit error within one power cycle. The 1st series of tests was done indoors, while the 2nd was done outdoors. It is possible that temperature changes from moving air caused errors. However, the 1st test in the 2nd series, done with the car engine off, did not produce an error. This suggests that interference from the cars electrical system, such as spark ignition, may have contributed to the error. As the development progresses it will be important to test the GPS both with the mower systems motors on and off and see if we have an error for repeat measurements at the same location. Outside of this, the accuracy from GPS measured coordinates to Google maps measured coordinates lies within the tolerances for the 2 systems. We conclude that the GPS system needs to be further investigated for effects of electronic noise but is generally operational.

3.4 Ultrasonic Sensors

3.4.1 Ultrasonic Sensor Summary

The HC-SR04 Ultrasonic sensors used require a trigger pulse as input and return an Echo pulse as output. The distance from the US to the object it is detecting is proportional to the time the echo pulse is high. To operate the sensors the MCU sends a $10\mu\text{s}$ trigger pulse and waits for the echo pulse to go high. Once an echo pulse is detected the MCU starts a timer, and the appropriate pins are sampled continuously, once the echo pin resets to low, the duration of the pulse can be calculated. This is the 2-way travel time of a sound wave from the sensor to the object. Multiplying by the velocity of sound in air and dividing by 2 gives distance to the object. The Speed of sound is variable with temperature, to account for this while keeping complexity down the speed of sound is set corresponding to a temperature of 29°C , which is within expected operating range but gives better results at higher temperatures, when the system is more likely to be used.

Speed of sound over our operating range of 0°C to 50°C ranges from 331.4m/s to 360.3m/s . For a maximum detection distance of 4M , resulting in 8M total signal travel, this gives a 0.0222s travel time at 50°C and 0.02414s at 0°C . For 29°C , as used in code, the speed of sound is 349.3m/s . plugging this and the 2 times found above into the equation used in the code. Gives results of 3.87M and 4.21M at the 2 extreme temperatures for a 4M distance at 29°C . If this error is determined to be excessive in later testing, provisions to adjust for temperature exist.

In the Final implementation we used 5 front facing sensors and 2 side facing sensors, labeled the front and side array. Each Array has its own trigger input, and each sensor its own echo return wire. By seeing which sensor reports the lowest distance to the object we can determine what side the object is on, for the front, or if we are moving towards or away from it for the side array. During system testing 2 of the 5 front sensors unfortunately failed, while the other sensors encountered an error limiting their range. Test results from 403 for a single sensor are provided below, while Sensor array results taken in the scope of system testing are provided in the validation section.

3.4.2 Test Results

Tests were conducted by placing the test target Infront of the Ultrasonic sensor, for incidence angle within 15° of Normal. Target distance was recorded, and 5 samples at each distance were read, next the target was moved back, and the process repeated. All distances were converted to cm. The US was mounted 6in above a desk, facing into the room. The 3 tests below represent 150 total data samples and show that we can detect large objects, such as a $4\text{x}4\text{ft}$ whiteboard very accurately up to $3+\text{M}$, the maximum distance we were able to test. A $3.5\text{x}5.5$ in object was successfully detected up to 2M from the sensor. We were not able to detect a person accurately at any distance.

3.4.2.1 Test of 4ftx4ft Whiteboard

The first Series of Tests was conducted by measuring distance to a 4x4 section of white board. Results are very accurate and consistent, becoming slightly less accurate at longer ranges. The Errors Graphed below show that we have a maximum error of around 3%. The relatively high error for 16cm may be due to the fact that target distance was recorded by hand, and any slight imprecision in hand measuring would be much more significant at this short distance.

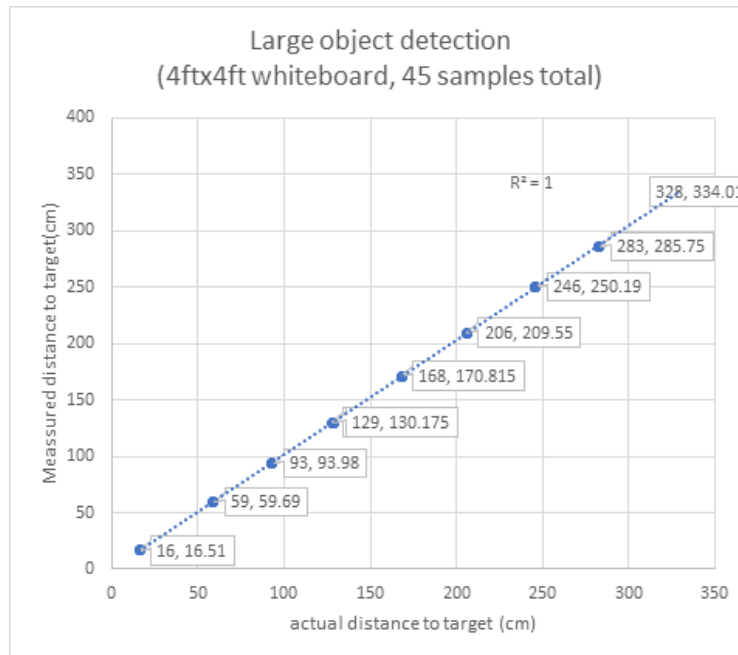


Figure 9: Ultrasonic sensor detection of large object test result graph

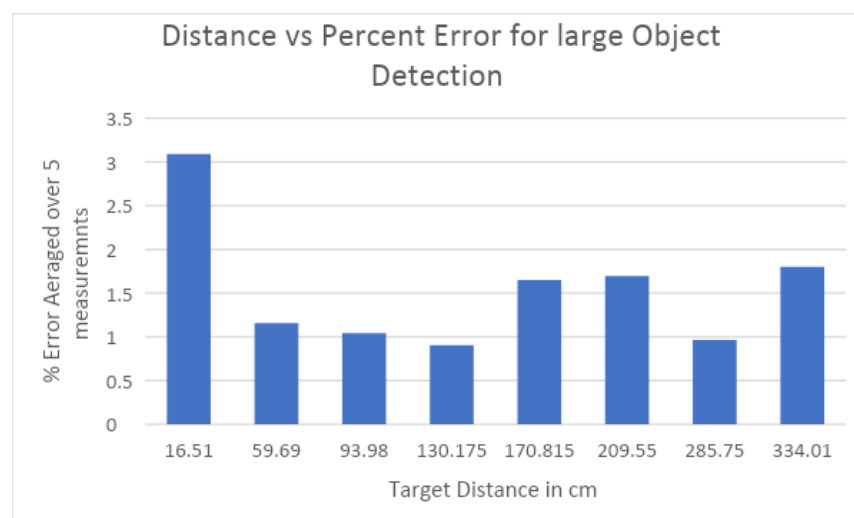


Figure 10: Percent Error for Large Object Detection

3.4.2.2 Test of 3.5inx5.5in Camera

Here we tested our ability to detect a small object. The target for this test was a DSLR camera, mounted on a tripod, with the back side facing the sensor. The area presented was approximately 3.5x5.5 inches and relatively smooth. Measurements were accurate up to about 2M. At that point the sensor was picking up the background rather than the camera. We did have 2 outlying points at ~14M. This is believed to be due to an error inside the sensor and will be elaborated upon at the end of the test section. Measured vs actual distance is shown in the first graph below, the second graph shows Percent Error vs distance, averaged over 5 measurements at each distance. The Yellow columns on the second graph show those cases where the US picked up the background instead of the object we were trying to detect. Additionally, the Red bar shows our highest error case over the valid range. As the measurements to the target were done by hand with a tape measure, it is possible that the target distance was measured incorrectly for this point, as it lies far outside of any other value. Finally, as mentioned above, we have 2 points at 14M, likely due to a sensor error that will be explained below. This data was omitted from the error graph as its high value rendered the graph useless for all other data. Instead it is presented with the other errors in a table below and marked in yellow.

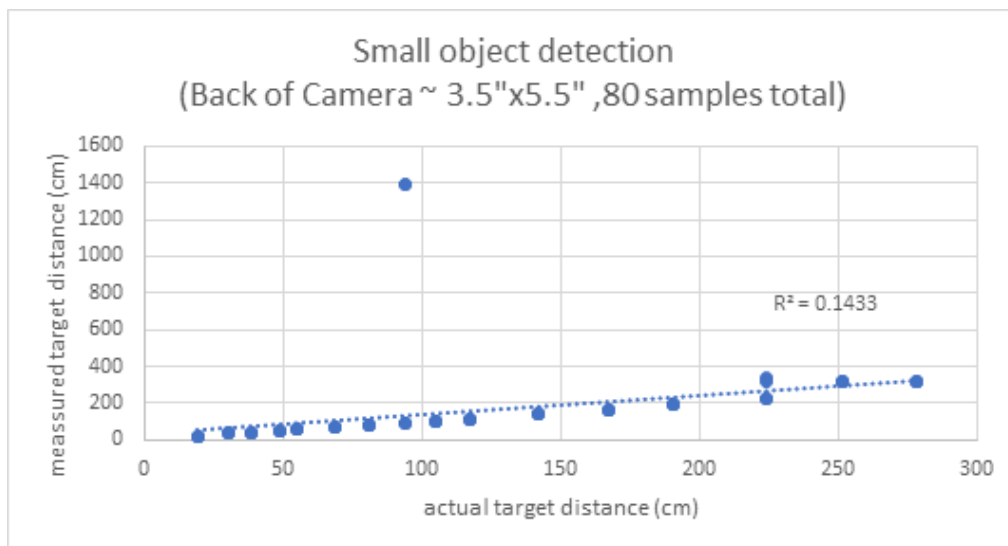


Figure 11: Ultrasonic sensor detection of small object test result graph

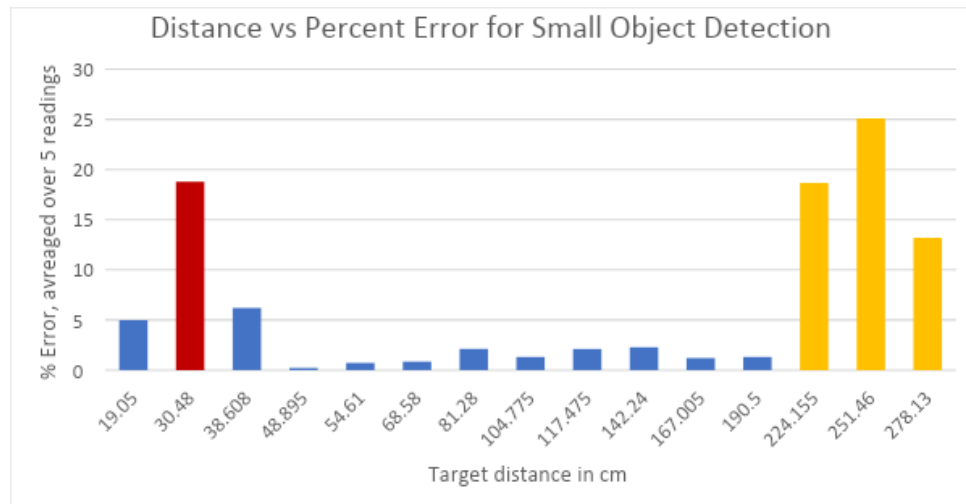


Figure 12: Percent Error for small object detection

Small Object Detection Error		
Target distance in cm		% Error to target
	19.05	4.98687664
Area	30.48	18.7664042
	38.608	6.195607128
	48.895	0.214745884
	54.61	0.714154917
	68.58	0.845727617
	81.28	2.116141732
	93.98	552.4792509
	104.775	1.312335958
	117.475	2.106831241
	142.24	2.27784027
	167.005	1.200562857
	190.5	1.312335958
	224.155	18.66788606
	251.46	25.10936133
	278.13	13.18448208

Table 3: Distance and Error Percentage for small object detection

3.4.2.3 Test of Person

These tests were done with the help of an assistant, wearing sweatpants and a sweatshirt who stood in front of the sensor at the indicated distance. It is clear that the obtained results are very poor, with only 3 out of 25 samples being within 20cm, while the others were off by 50+cm and often multiple meters. We see 14M repeated often, as in the above test of the small object, which we will elaborate upon next. We currently believe the reason we can't pick up people is due to their clothes. The fabric likely absorbs the Ultrasonic sound, and never reflects a return echo the US can pick up. The second table shown below gives percent error vs distance, averaged over 5 samples. It appears as though the accuracy gets better as the distance increases, but this is only because the sensor started picking up the background.

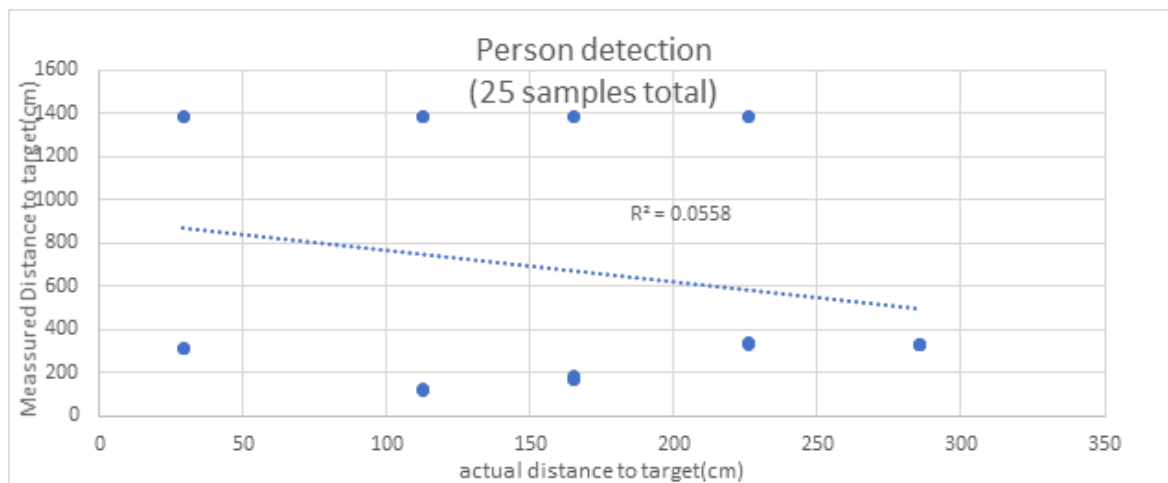


Figure 13: Ultrasonic sensor detection of person test result graph

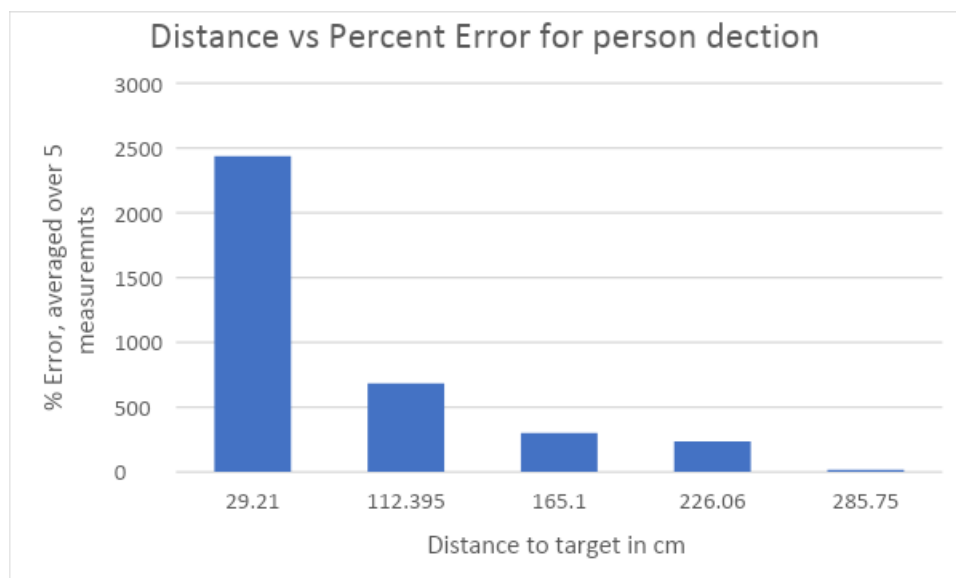


Figure 14: Percent Error for Person detection

3.4.3 Ultrasonic Sensor Test Summary

As seen above we have problems detecting small objects at distances over 2M and detecting people at any distance. Many sensor readings show 14M, which is greater than the maximum distance possible at the test site. This distance likely corresponds to the maximum time the echo output stays high if no return pulse is received. Since we can not distinguish between cases where the distance is too great to receive an echo, and cases where the incident Ultrasonic pulse was absorbed, for example by fabric, this signal is of no use to us. For 404 a front array of 5 sensors and a side array of 2 sensors was implemented to gain a better understanding of our environment. This allowed us to obtain angular information about objects. The issue of revolving obstacles covered in fabric was not solved.

3.5 *WIFI-Module*

3.5.1 WIFI Summary

As the WIFI module on the PIC failed, we used an ESP-32 to implement WIFI capacities. This is part of the ESP-W component of the MCU subsystem and was implemented by Jonathan, who will provide a report in the appropriate section.

3.6 *Drive Motor Control*

3.6.1 Motor Control Summary

As described above Hardware issues prevented us from using the PIC-32 for motor control. It was instead implemented in the ESP-N portion by Max. I will elaborate upon it in the appropriate section.

3.7 *Blade Motor Control*

3.7.1 Blade Motor Control Summary

Blade Motor Control was also implemented from the ESP-N. It could have been controlled by the PIC, but from a control logic perspective, using the ESP-N was the better idea. The hardware and software solution to this will be presented by Max in the appropriate ESP-N and Motor section .

3.8 Accelerometer and Gyroscope Unit control

We did attempt to implement a Gyroscope and accelerometer with the ESP-32, however this proved unsuccessful. See ESP-N section for details.

3.9 Battery Status Interface

3.9.1 Battery Status Interface Summary

Due to time constraints the Battery status interface was not addressed further than at the end of 403, whose extent is listed below. Additionally a changed battery requirement would have rendered any previously designed Battery status interface useless.

The Battery status interface intended to read battery level, has not been fully implemented. We are awaiting exact specifications from the power subsystem as to what voltages set the threshold for a charged and discharged battery. The proposed solution is to use a comparator with Zenner diode reference, and a voltage divider to bring the battery voltage into a range where the charged battery voltage would be above the Zenner Voltage and low charge voltage below. This would provide us with a 0V and 3.3V at the comparator output, for a charged and discharged state respectively. We can choose which we want to correspond to charged and discharged by which pins of the Comparator we connect to which signal. The voltage level at which the battery is considered discharged can be set by the resistor divider and the Zenner diode. Since we do not have this information available it is impractical to construct the circuit at this time. However, to demonstrate the design, we have Multisim simulations, using realistic components, that verify the design.

3.9.2 Battery Status Interface Simulation Results

The circuit simulated here assumes 12.0V and above for a charged battery, and 11.9V and below for a discharged battery. In operation this threshold will need to be adjusted such that the Mower system has enough time to return to the docking station once the low voltage alert has been tripped. The threshold can easily be varied by changing resistor R1. Additionally, this circuit outputs logic low when the battery voltage is sufficient, and logic high when it is low. This behavior is chosen such that during normal operation the MCUs GPIO pin takes as little current as possible, to keep MCU power dissipation, and hence heat, to a minimum. The output from the comparator can be read using a configured GPIO pin on the MCU, which can be used to provide a Boolean variable driving a "Return to base" logic.

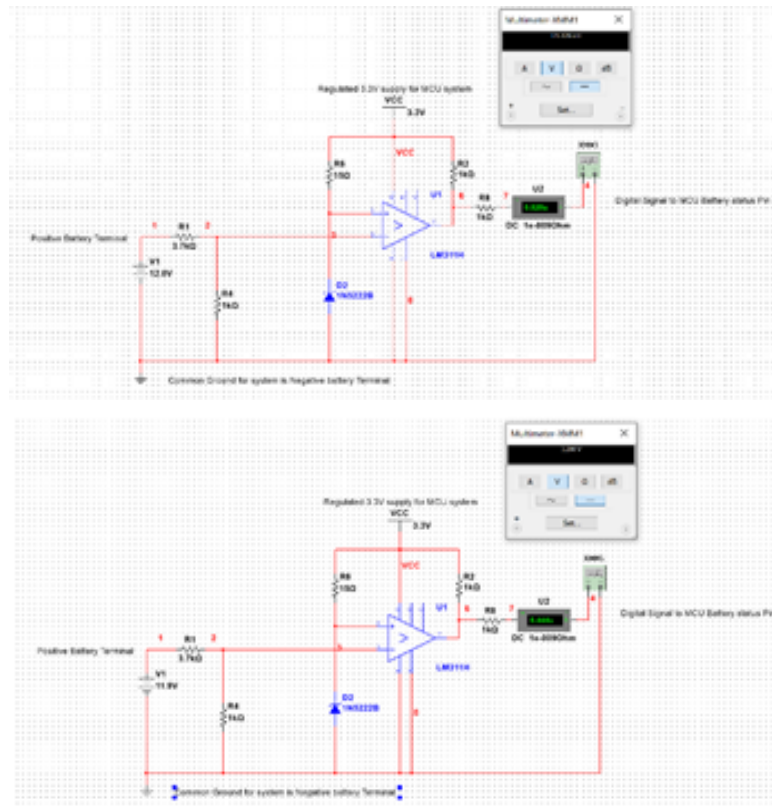


Figure 15: Battery read circuit simulation for “Battery Charged” (top) and “Discharged” (bottom)

3.10 MCU Functions Relegated to and Dependencies on Other Subsystems

The Following is a list interface issues that need to be addressed and reconciled with other subsystems. This List is by no means extensive, but it lists some of the interface issues that need to be addressed immediately next semester, and those that have limited us in development of the MCU subsystem. Additionally, we list any functions that have been turned over to other subsystems.

3.10.1 Functions Turned over to and Dependencies on Motor Subsystem

The motor subsystem was rebuilt by Max after the initial Motor subsystem failed. Its control was implemented on the ESP-32, also by Max. See motor subsystem for Hardware details, and ESP-N subsystem for control details.

3.10.2 Power System Dependencies

Due to time constraints and component changes any dependence on the power subsystem, outside of power delivery, was removed.

3.10.3 Sensor System Dependencies

The Sensor subsystem was taken over by Max, additional sensors for angular resolution were added, however the issues surrounding detection of fabric covered obstacles was not solved.

3.10.4 User Interface Dependencies

The Connection to the user interface took place on the ESP-W, and was handled by Jonathan.

3.11 PIC-32 MCU Subsystem Conclusion

The PIC-32 was intended as the sole MCU for the system. Due to failures of the WIFI module and a misprint of the PCB an ESP-32 was added. Aside from solving the WIFI issue, this also provided enough IO lines to include additional Ultrasonic sensors, as well as accommodate new motors drivers, elaborated upon in the Motor SUBsystem section. The PIC-32 performed well in obstacle detection and reading of GPS.

In hindsight the PIC firmware and hardware development was a very large task, and a better approach may have been to use a development board to save time on hardware development. Nonetheless learning embedded system design on the pic proved very educational, and has certainly made me a better programmer, and set me up to be a better engineer.

4 ESP-N Subsystem by Max Lesser

4.1 Introduction

As elaborated above, the MCU subsystem is split into the PIC component, ESP-N for navigation and ESP-W for user interface, where both ESP components are on the same Microcontroller, but since they perform distinctly different functions and were executed by different team members they have separate sections.

Here I will elaborate upon the Navigation and motor control component of the ESP-32. Jonathan will explain the User interface component in the next section.

The ESP-N is tasked with controlling the drive and blade motors, following a rectangular path and avoiding obstacles as needed. It also reads Shaft encoder data, and has provisions for reading IMU data.

I will note here that the navigation component was originally in the scope of Josh's subsystem. I began developing an alternative towards the end of the semester when it appeared the original navigation subsystem would fail. The solution implemented here was very much an emergency effort on my part, parallel to me rebuilding the motor subsystem and working on general integration. This subsystem is hence not as sophisticated as it should have been, given more time and not as extensively tested as it could have been.

4.2 Navigation

Navigation is implemented using inertial guidance from shaft encoders. By measuring the number of wheel rotations we can conclude where we are relative to a starting position. Turning is implemented in the same way, by breaking one wheel and rotation the other one until a 90 degree turn is complete. The issue with this scheme is that error accumulates. I attempted to address this issue using a Gyroscope, as well as Accelerometer and Magnetometer. This approach proved unsuccessful due to a faulty sensor. A second approach was made using the PICs GPS unit, this however failed due to the GPSs slow update rate of 1Hz. With more time one or both of the above schemes could have been successful. As a result the mower drifts of course the longer it runs, as can be seen in submitted videos. Details about the sensors will be addressed further below.

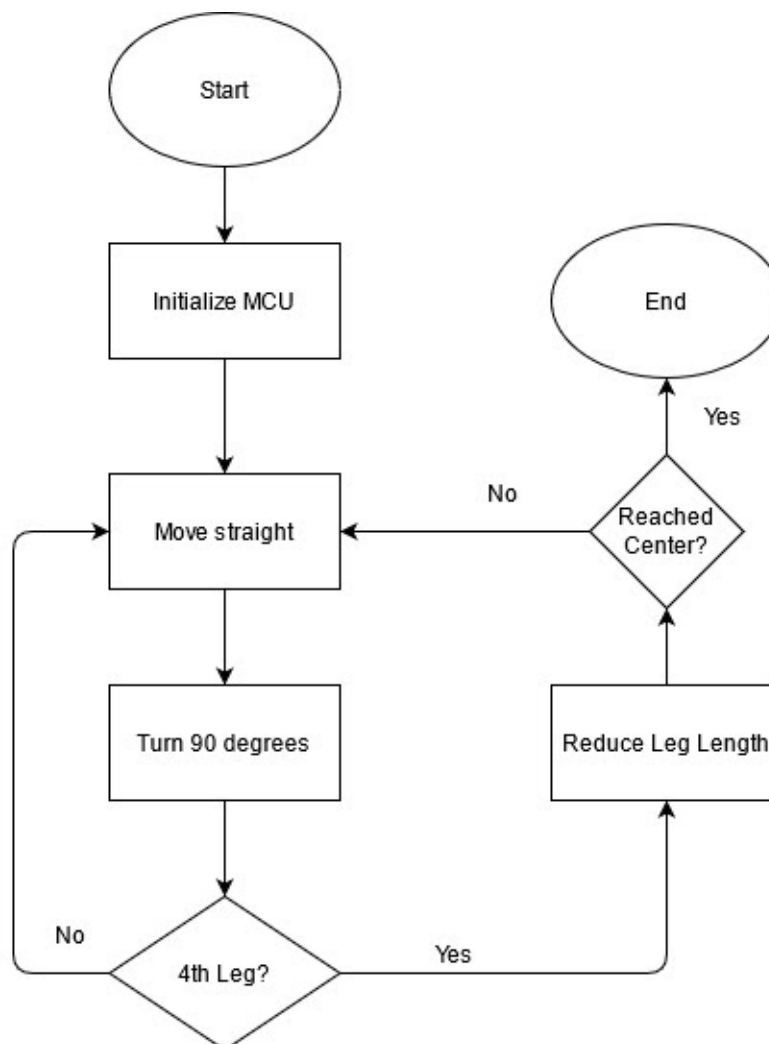


Figure 16: General Navigation flow without obstacle evasion

4.3 Obstacle Evasion

Information about Obstacles was obtained from the Pic, as explained above. Once the PIC located an obstacle within 40 cm of the unit, it would set a GPIO pin high, which the ESP polled during maneuvering. This function was originally implemented using an ISR on the ESP, but stray voltages on the alert pin would cause unexpected triggers of the ISR. The GPIO pin approach proved more robust.

Once the ESP was alerted to an obstacle, it was to immediately break and disable the blade motor. It was then to turn away from the object until the front array no longer reported anything. Then it was to move straight to clear the object. Once cleared the side array would be checked. If the side was clear the mower was to return to the original path, resulting in a triangle pattern. If the side was not clear, the mower was to move parallel to it until the side became clear, and then return, following a trapezoidal pattern. This is outlined in the Flowchart below. Unfortunately we encountered a failure of the Left Shaft encoder. Which rendered the above approach impossible. The work around to this problem was time based obstacle evasion, where the mower would turn for a set time, then move straight, turn for twice the original time, move straight again, and finally turn back onto the original path, resulting in a triangle shaped evasion path. This approach did allow us to clear obstacles, but with limited fidelity.

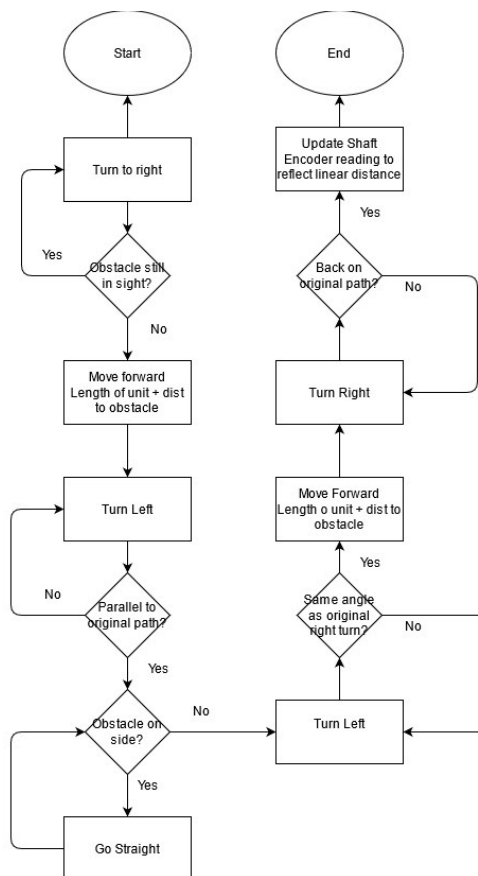


Figure 17: Obstacle Detection Flow Chart

4.4 Shaft Encoders

For Inertial Navigation the system uses 2 AMT102-V shaft encoders on the rear wheels. These allow us to measure shaft rotations with an accuracy of 48 counts per revolution. Translating Shaft Rotations to Wheel rotations, taking into account the existing gearing used to attach to the drive wheels then gives us distance traveled. This measure would let us accurately turn up to a 48th of a rotation, as well as travel in a straight line and use the above described obstacle evasion scheme. Unfortunately the Left wheel shaft encoder failed during late stage testing, rendering impossible obstacle evasion as described above. This also forced us to turn towards the left, instead of the right as intended. Which meant our side detection array was useless as it would now point away from any obstacle, towards the inside of the yard.

Discounting the issue above, both shaft encoders are powered by 5V, and send an indicator pulse for full wheel rotation, as well as a quadrature pulse train signal with 48CPR, where the lead or lag of the pulses determine forward or backwards rotation. As we can control the rotational direction of the motor, only one of the 2 quadrature signals was connected for each sensor, in an attempt to reduce complexity. For mounting details see Motor subsystem section

4.5 Accelerometer Gyroscope and Magnetometer

In an attempt to improve inertial navigation, and overcome the issues presented by the failed shaft encoder I added a miniIMU 9-v5 to the system. This sensor includes a Gyroscope, Accelerometer and Magnetometer. The Accelerometer and Magnetometer can be combined to obtain a compass heading, compensated for Pitch and Roll of the unit. In Bench tests of this setup the heading varied by over 60 degrees when the sensor was stationary. A second approach was to integrate the Gyroscope reading to obtain angle turned. This approach was successful in indicating 90 degree turns. It was however highly unreliable as the Gyroscope output would sometimes read all 0s or NaN. This combined with the issues with heading lead to the conclusion that the sensor is defective. It was too late to obtain a replacement at this time. See additional code/MAX section in the github repo for software implementation details.

4.6 Motor Control

The original motor subsystem proved underpowered for our unit, new motors and driver boards were required. The hardware details are outlined in the New Motor Subsystem section. The New drivers required 2 PWM signals per motor, for forward and reverse. As well as 2 digital logic pins each to set functionality. The ESP-N attached to these new drivers with the above signals. Setting the logic pins high permanently at the beginning, and then giving PWM to either the forward or reverse input of each driver, and setting the other one low allowed for straight line or turning motion. The PWM sent was at 5 KHz, with a maximum duty cycle in turn of 62%, 27% for straight motion, 2.7% reverse for breaking.

4.7 Blade Motor Control

The Blade motor was controlled by the ESP-N via relay driver. The hardware details are outlined in the New Motor subsystem section. The software side of this implementation is rather simple, as the mower simply sets a GPIO pin high to activate the blade, and low to stop it. This control scheme results in the blade being shut off when the MCU loses power. For safety the blade was deactivated during obstacle evasion.

4.8 Summary and Conclusion

As my attempt to find a solution to navigation and obstacle avoidance was very last minute, it clearly lacks sophistication a semester long subsystem might have. Additionally we were plagued by failures of shaft encoders and IMU sensors, not to mention the physical rebuild of an entire subsystem. Given this the ESP-N subsystem performs reasonably well, and is close to being a successful subsystem. A means to successfully read heading would have solved our navigational challenges. Additionally with more time a true SLAM algorithm could have been implemented using the PIC's GPS output. Unfortunately time constraints and other issues prevented this. Given perhaps 2 more weeks many of these functions could have been implemented. Nonetheless, considering the short time and other issues the Navigation and control subsystem presented above was reasonably successful in achieving the mission.

5 ESP-W Subsystem

To learn more about the ESP-W system refer to Section 4 of the User Interface Subsystem Report

Autonomous Lawnmower

Max Lesser

MOTORS SUBSYSTEM REPORT

REVISION – 3
29 April 2021

Table of Contents

1 Introduction and Purpose	155
2 Part Selection	155
3 Mounting and Attachment	157
4 Motors	159
5 H-bridge	159
6 Blade Motor	160
7 Transition from Old Motor System to New Motor System	162
8 Conclusion	162

List of Tables

Table 1: Control Signal Table for New Motor Drivers'	159
Table 2: Control Signals for Different Actions as a Percentage of Maximum	159

List of Figures

Figure 1: New 240W Motor with gears attached, with first shaft coupler attempt attached	157
Figure 2: Original Lawnmower's gears and the manufactured nuts to mount to the shaft	157
Figure 3: Gear Configuration of both the gear on the motor shaft and the wheel	157
Figure 4: Mounted Shaft Encoder on Motor	158
Figure 5: Shaft Adapter Assembly and Shaft Encoder	158
Figure 6: Mower Assembly Showing Mecanum Wheels in the Front	158
Figure 7: Relay Driver Circuit Diagram	160
Figure 8: Assembled Relay Driver	160
Figure 9: Bottom view of blade Mounted to Blade Motor. Flap disk used in testing for safety	161
Figure 10: Top view of blade motor, Cover not shown but can be added	161

1 Introduction and Purpose

During testing it became apparent that the previous Motor subsystem was not powerful enough to allow for navigation. Hence a new motor Subsystem was needed. This section details the new motor subsystem, developed and implemented at the very end of 404. As it was developed in the last 2 weeks of the semester, it was rather rushed, leaving this section somewhat empty of testing detail as could have been achieved by a semester long development.

This subsystem consists of two 230W 24V motors, and an off the shelf H-bridge for each, shaft encoders for the drive motors, as well as the 12V blade motor from the original motor subsystem, and its relay driver. In addition, this system includes mechanical components, such as front wheels and drive shaft assembly needed in the rear, along with mounts and incidental hardware to attach the motors. The H-bridges and relay driver are connected to the ESP-N subsystem which controls the motors. Shaft encoders are mounted to the motors, and over the motor shafts to measure wheel rotations, outputting data to the ESP-N. Driving of the wheels is achieved via the gearing and cogs that existed on the original mower. However, custom parts had to be manufactured to attach to the Cogs and Motor shafts. Mecanum wheels are used in the front to allow for 0 radius turns.

2 Part Selection

Due to delayed testing the need for new motors did not become apparent until the end of the semester. Hence the prime criteria for new motors was power, followed closely by delivery time, and cost. The Motors meeting all of these requirements were YaeTek 24V Brushed DC motors. As these new motors were substantially more powerful than the old ones, new drivers were also required, with again the same urgency as the motors. The BTS-7960 driver boards meet our needs and were available in a short time frame. The downside to these new parts was that the motors required new mounts due to their larger size, and the driver boards required a different control logic from the ESP-N. As well as a need to run the Motors on 24V. This was not ideal, but seeing as this motor and driver combination was the only one that could get here in time while being strong enough and affordable, the choice was made.

3 Mounting and Attachment

The new motors were mounted using L-brackets bolted to the side of the mower. With holes drilled to match the pattern on the motors. Due to the heavy motors, not flat mower sides, and limited manufacturing capacities, the motor shaft did not perfectly align with the hole through which it needed to fit. Fortunately some inherent play in the gearing absorbed the difference.

The gears themselves were attached to the old motors by means of custom manufactured M-8 nuts. As the original mower was self driving, allowing for either powered or pushed operation, the gears interior surface was roughly triangular, to allow a slotted key to either grab or slip. For attachment to the old motors triangular nuts were manufactured to grab into this pattern and allow the wheels to spin forward and backwards. These Nuts were attached to M-8 threaded shafts which were then bolted to the old motor shafts. These were reused for the new motors, after a way to mount the shafts was found.

A larger issue was presented by the Motor shaft itself, which was a left-hand metric thread. This was not apparent from the vendors website, and instead had to be addressed after receiving the motors. As a Left hand to right hand metric coupler was locally unavailable, manufacturing one using a regular M-8 coupler with the threads drilled out to slip over the left handed thread was the next solution. Drilling perpendicular through the New motor shaft and this coupler allowed the insertion of a cotter pin. This solution was not ideal as the cotter pin broke repeatedly during testing. With more time a proper adapter could have been obtained, but this was not an option for us.

A final issue was presented by the shaft to wheel attachment. While the actual drive used cogs, the mowers original drive mechanism was a single shaft enclosed in an axel bolted to the frame in multiple places. Since we did not have the luxury of an axle, nor the space or manufacturing ability to add shaft bearings we had to improvise. By forcing backwards a rubber bushing in the hole the shaft would pass through, roughly 1/8th inch recess was created. Using an M-8 inside diameter washer with outside diameter greater than the hole, I was able to grind down the outside using a drill and angle grinder. Once small enough to fit into the hole but not the bushing, we had our own shaft bushing. Pinned with 2 nuts from the outside, the shaft was held in place. This configuration allows us to drive the wheels with our new motors, without excessive play. Credit for this saving idea goes to Vincent McMasters.

In order to allow 0-degree turns yet maintain stability, mecanum wheels were used in the front. To mount these custom brackets were made to extend the wheels forward enough. This mounting solution provided for relatively stable forward motion, while allowing us to turn from a stand still. Pictures detailing these components are presented below.



Figure 1: New 240W Motor with gears attached, with first shaft coupler attempt attached



Figure 2: Original Lawnmower's gears and the manufactured nuts to mount to the shaft



Figure 3: Gear Configuration of both the gear on the motor shaft and the wheel



Figure 4: Mounted Shaft Encoder on Motor

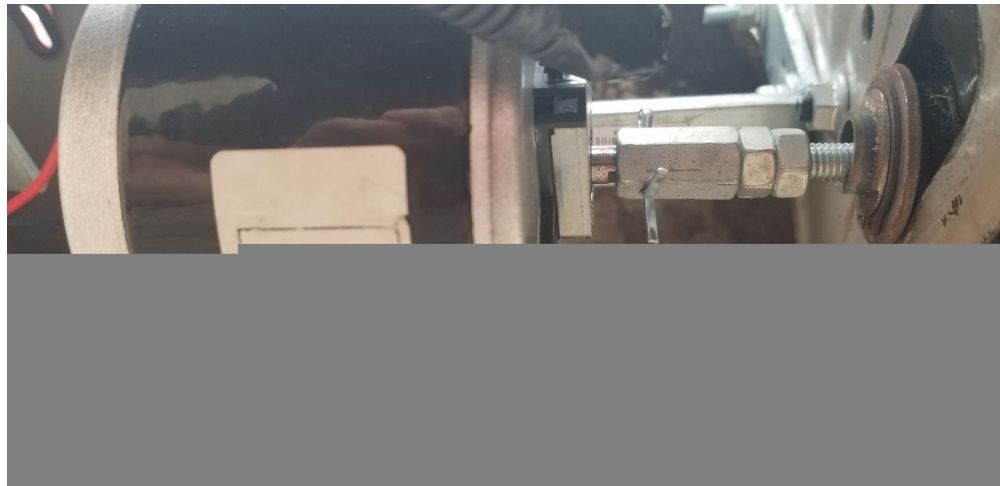


Figure 5: Shaft Adapter Assembly and Shaft Encoder



Figure 6: Mower Assembly Showing Mecanum Wheels in the Front

4 Motors

The New motors used were straight forward once the shafts were attached (see section 4.3). A 24V power supply was needed to provide sufficient power, which unfortunately eliminated our ability to use the docking station and charging system. See Section 4.7 for details.

5 H-bridge

An initial attempt was made to reuse the previous H-bridge from the old motor subsystem, as described in section 4.7 this attempt was abandoned due to overheating problems. The replacement H-bridges, ordered along with the motors, were BTS-7960, capable of handling up to 43A per motor. This did change the control inputs from the previous motor driver to that listed in the table below. Pin 3&4 had to be set to enable operation, with pin 2 low and PWM input into Pin 1 the mower would move forward, Pin 1 low and PWM into Pin 2 resulted in reverse. One such H-bridge was needed per motor.

Pin Number	Function
1	Forward PWM signal
2	Reverse PWM signal
3	Forward enable
4	Reverse enable

Table 1: Control Signal Table for New Motor Drivers'

The following PWM settings were used to perform the different navigation actions.

Function	Left Motor	Right Motor
Go straight	Forward 27.3 %	Forward 27.3%
Turn	Reverse 23.4%	Forward 39 %
Stop	Reverse 2.7%	Reverse 2.7%

Table 2: Control Signals for Different Actions as a Percentage of Maximum

6 Blade Motor

The blade motor was the original 12V blade motor, interfaced with the ESP-N via darlington pair relay driver controlling a G5LE relay. This Allows the ESP-N to control the blade motor without drawing too much current. And shuts down the blade in case the MCU loses power.

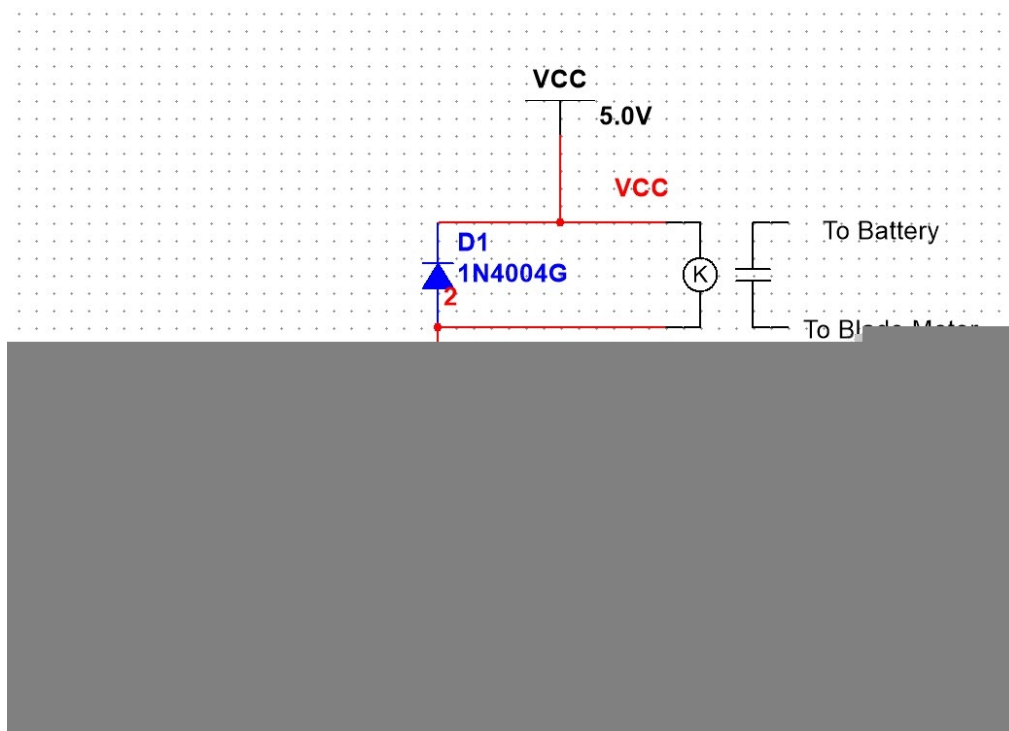


Figure 7: Relay Driver Circuit Diagram



Figure 8: Assembled Relay Driver



Figure 9: Bottom view of blade Mounted to Blade Motor. Flap disk used in testing for safety



Figure 10: Top view of blade motor, Cover not shown but can be added

7 Transition from Old Motor System to New Motor System

After the new motors were acquired, an attempt was made to reuse the previous power and motor control subsystem with the new motors. In initial bench tests of the new motors, a constant input of 12V at 5A seemed to provide sufficient power, so it seemed as though we could reuse previous power and control components. In system tests however, a loss of power with increasing runtime was observed. After less than 60 seconds the power output decreased to the point the mower was no longer able to move. An initial consultation with Prof Kalafatis suggested the motors may be overheating. To overcome this fans were added, however this had no observable effect. Further investigation suggested that the Lead Acid battery may not be providing sufficiently consistent power. Through a gracious loan by a friend I was able to abstain LiPo batteries for RC planes. The capacity of these batteries was limited, and since LiPos removed the charging ability developed by the docking station regardless of configuration, it was decided to assemble them in series, powering the motors from 24V to compensate for lower capacity, and tapping 12V from one of the batteries for the remaining mower systems. This approach did improve power output. The torque however still declined with increasing runtime, just more slowly now. After temperature measurements we found the old H-bridge to reach excessive temperatures. Replacing it with the new components described in section 5 resolved the issue entirely. While this rebuild was very hectic, and not ideal, it did allow us to achieve our core mission of lawn mowing. As such it was in our opinion the best option considering all alternatives.

8 Conclusion

This Subsystem was very much scraped together at the end of the semester after the initial motor subsystem did not function. While the part selection was not ideal, and gave many mechanical problems, as outlined in section 3, there ultimately was no alternative. The mechanical and control challenges presented were overcome, but this time cost time that couple have been applied to the implementation of other subsystems such as the ESP-N. Nonetheless, given the short notice I had to develop this component I am satisfied with my solution. In hindsight I believe we as a team should have been more demanding of validation and test results from team members, such that this shortcoming could have been detected sooner, if not entirely prevented.

Autonomous Lawnmower

Jonathan Poulouse

USER INTERFACE SUBSYSTEM REPORT

REVISION – 3
29 April 2021

Table of Contents

1 Subsystem Introduction	166
2 Android Studio IDE	166
2.0 Features Added This Semester	166
2.1 Main Activity Screen	167
2.2 Setup Activity	168
2.2.1 Default Lawn	169
2.2.2 Start/Stop Changing	170
2.2.3 Battery Level	170
2.3 LawnActivity	171
2.3.1 Lawn Name	172
2.3.2 Scheduling	173
3 Firebase	174
3.1 Google Cloud Platform	175
4 ESP32	176
4.1 WiFi ESP32	177
4.2 Navigation ESP32	178
4.3 Problems	179
4.4 Fixes/Improvements	180
4.5 Summary for ESP32 Code	181
5 Conclusion for User Interface/Server/WiFi	182
6 Moving on	182

List of Figures

Figure 1: Home Screen Prior to Login	167
Figure 2: Home Screen When Logged in	167
Figure 3: Add Multiple Lawns	168
Figure 4: Default Lawn	169
Figure 5: Start/Stop	170
Figure 6: Battery level	170
Figure 7: 4 coordinates	171
Figure 8: 6 coordinates	172
Figure 9: Days of the week	173
Figure 10: Time	173
Figure 11: Full Firebase Server for 1 user	174
Figure 12: Screenshot of API credentials created	175
Figure 13: Successful communication between the two ESP32s	176
Figure 14: Snippet of WiFi MCU Code	177
Figure 15: Snippet of Nav MCU Code	178
Figure 16: Rebooting Issue with Josh's code	179
Figure 17: Shown Distances Calculated from the 4 coordinates (Serial Monitor)	180

1 Subsystem Introduction

The User Interface subsystem consists of the Java and XML code that was used to construct the android application used to control the autonomous lawnmower. The software used to write and test the code is Android Studio. The android application will write directly to the Firebase Server Database and upload information and commands in realtime. The main function of the application will be to set the specific coordinates of the corners of the user's lawn to draw the path of the lawnmower. To create this application API Keys and OAuth 2.0 Client IDs will be needed for Google Maps, Google Account login, and Firebase integration. Another part of my subsystem was the WiFi component of the ESP32.

2 Android Studio IDE

The entirety of the application will be coded and tested on the integrated development environment, Android Studio version 4.0.1. This software also allows me to test the application on any virtual android device. I have been emulating the apk file on the Pixel 2 and 3a as well as the Nexus 5 & 6 with android 9.0 and 10.0 to test. Android Studio made the integration with google maps very easy especially because there were many built in functions made specifically for google maps.

2.0 Features Added This Semester

- Stop Function
- Scheduling ability
 - Days of the week/time
- Default lawn
- Battery status

2.1 Main Activity Screen

Allows user to login with any google account

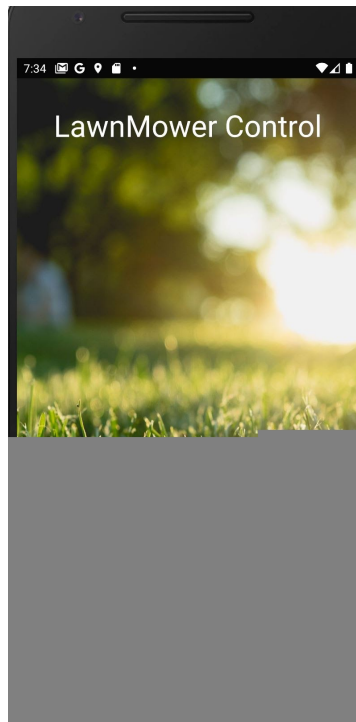


Figure 1: Home Screen Prior to Login

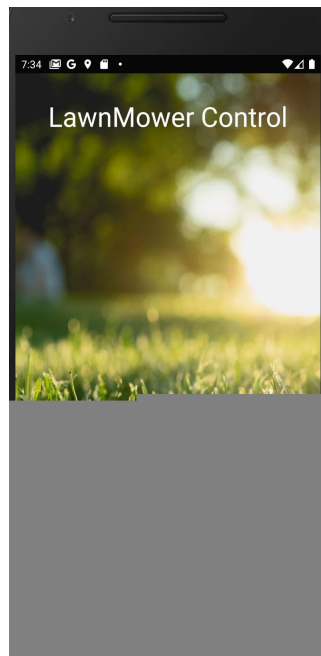


Figure 2: Home Screen When Logged in

2.2 Setup Activity

Allows user to add multiple lawns and shows lawnmower information



Figure 3: Add Multiple Lawns

2.2.1 Default Lawn

Allows users to select default lawn (lawn that will be cutting in real time). This part of the application was crucial because this allowed the ESP32 to get only the coordinates from the selected lawn. The ESP32 code could only get the coordinates from a certain path in firebase so I had to create a “default lawn” section that would change once it was selected in the application.

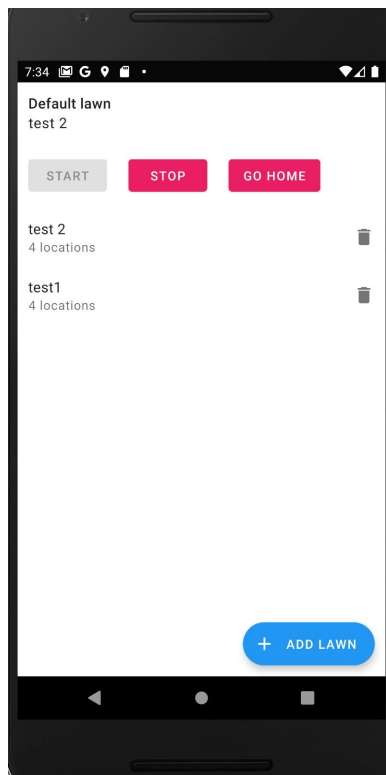


Figure 4: Default Lawn

2.2.2 Start/Stop Changing

Start and Stop gets Greyed out when one of the buttons is clicked

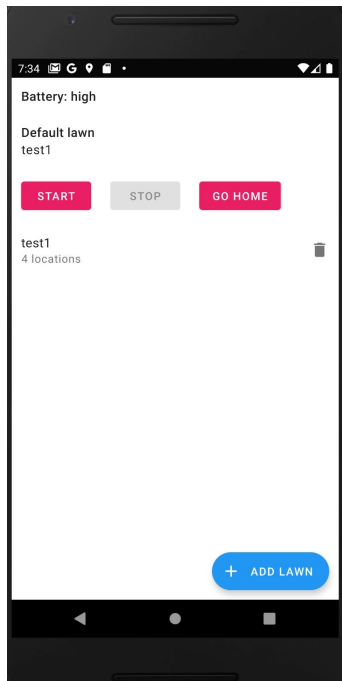


Figure 5: Start/Stop

2.2.3 Battery Level

Battery Level is shown as high or low

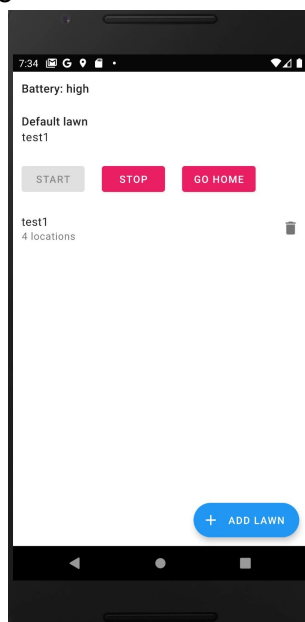


Figure 6: Battery level

2.3 LawnActivity

Allows users to use current location, set the path of the lawn, and name the lawn. Each marker placed starts from 1 to desired amount. Can add unlimited coordinates

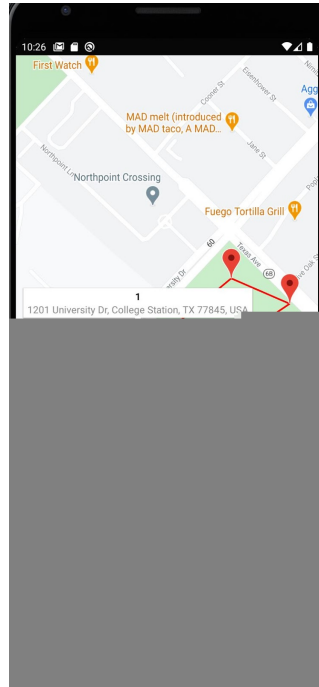


Figure 7: 4 coordinates

2.3.1 Lawn Name

Allows user to add lawn name and unlimited number of coordinates

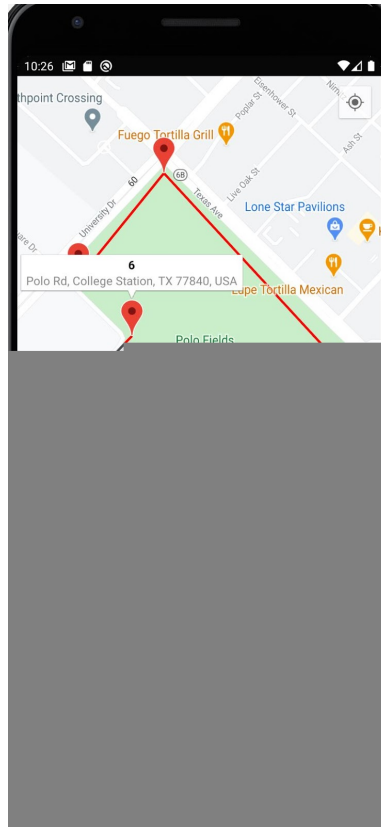


Figure 8: 6 coordinates

2.3.2 Scheduling

Allows user to schedule the lawn mower to go at a certain time and day of the week

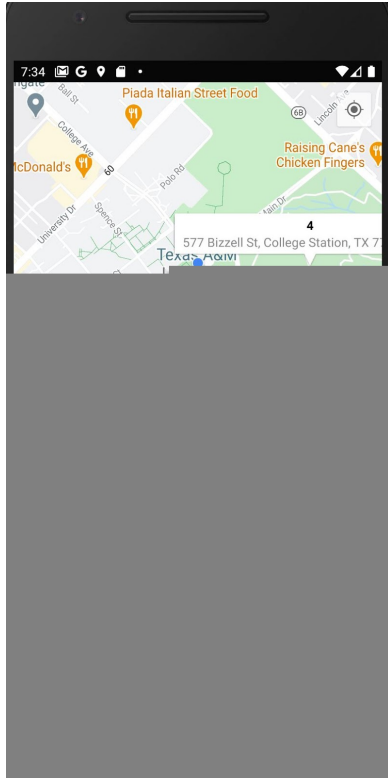


Figure 9: Days of the week

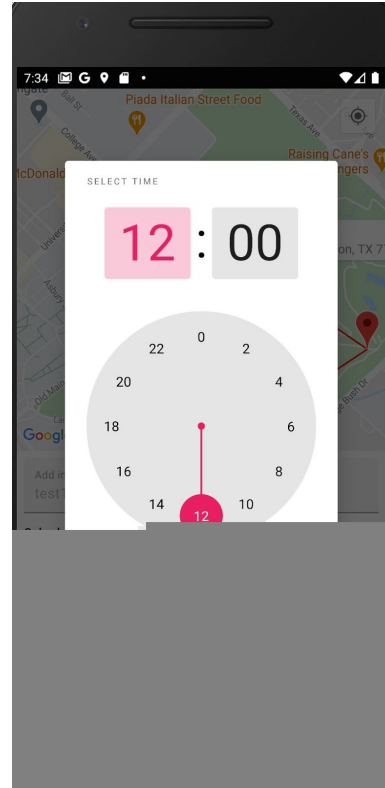


Figure 10: Time

3 Firebase

Using the Realtime Database in Google Firebase platform, the latitude and longitude coordinates are constantly updated as soon as a user signs in and saves a “lawn”. The child hierarchy goes in order from Application Name>user ID>lawn name>marker number>latitude and longitude. When connecting the server to the WiFi module, the coordinates have to be in the same format as whatever the MCU can read it in. Once the application is complete all information such as usage statistics and navigational data will be transferred through the wifi module attached to the MCU and the Google Firebase Database.

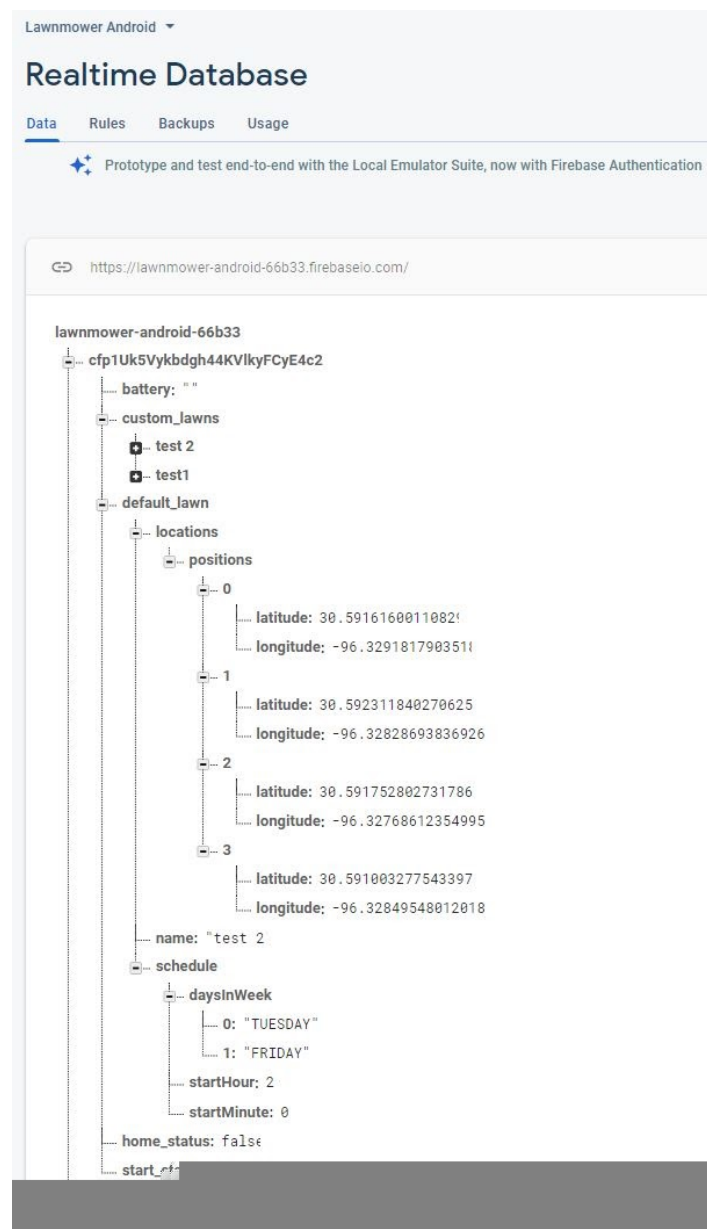


Figure 11: Full Firebase Server for 1 user

3.1 Google Cloud Platform

The Application credentials types that will be used are API keys & OAuth 2.0 client credentials. The only API key used in the code is the Google Map SDK API key to allow users to use google maps. All the other credentials were autocreated. Google Firebase auto created an android and browser key to allow the information to get updated in the realtime database. Google Services auto created an android client and web client to allow users to sign in to the app using a google account.

API Keys

<input type="checkbox"/>	Name	Creation date ↓	Restrictions	Key			
<input type="checkbox"/>	Google Map SDK	Nov 21, 2020	None	AIzaSyBsW1...cNXQLJsSV8			
<input type="checkbox"/>	Android key (auto created by Firebase)	Nov 21, 2020	None	AIzaSyAHc5...HtLqJpq91c			
<input type="checkbox"/>	Browser key (auto created by Firebase)	Nov 21, 2020	None	AIzaSyAnLF...QMmaLOfTTE			

OAuth 2.0 Client IDs

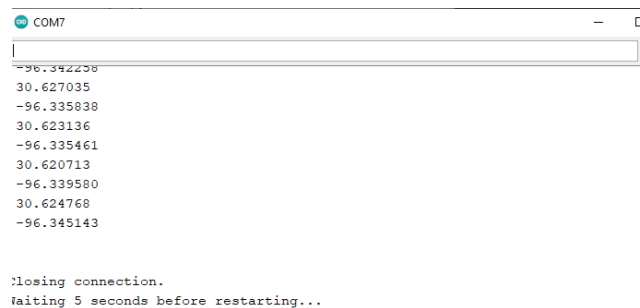
<input type="checkbox"/>	Name	Creation date ↓	Type	Client ID				
<input type="checkbox"/>	Web client 2	Nov 21, 2020	Web application	287950388755-ap9s...				
<input type="checkbox"/>	Android client for lawnmower app (auto created by Google Service)							

Figure 12: Screenshot of API credentials created

4 ESP32

My first versions of the code were for two separate ESP32s, WiFi and Navigation. In the beginning of the semester, we decided that I would work on the WiFi ESP32 and Josh would work on the Navigation ESP32. I ended up writing code for both so that they could communicate with each other and send variables back and forth.

I tested both of the old files/ESP32s without the navigation code to see if they would execute all functions and communicate effectively and there seemed to be no issue. You can see that coordinates successfully are being sent from one ESP to the other in the figure below



```
COM7
-96.342258
30.627035
-96.335838
30.623136
-96.335461
30.620713
-96.339580
30.624768
-96.345143

losing connection.
Waiting 5 seconds before restarting...
```

Figure 13: Successful communication between the two ESP32s (Serial Monitor from Nav ESP)

4.1 WiFi ESP32

For the WiFi ESP, the functions were:

- Connect to Wifi for Firebase Server & Connects to Nav ESP
- Receive coordinates & start_status from Firebase Server
- Receive battery information/statistics from Nav ESP
- Send battery information/statistics to firebase server
- Send coordinates & start_status to NAV ESP

In the figure below you can see the variables being sent and received as well as connecting to my routers network as well as the other ESPs network

```
WiFi_MCU
1 // #ifndef ESP32
2 #include <WiFi.h>
3 #include <HTTPClient.h>
4 #include <FirebaseESP32.h>
5 // #include <FirebaseJson.h>
6
7 // Txt files
8 #include <SPIFFS.h>
9 #include "FS.h"
10
11 #include <Arduino.h>
12
13 // Set web server port number to 80 - SoftAP
14 WiFiServer server(80);
15
16 // ESP32 wifi network connection data
17 #define ssidEspServer "ESP32Server"
18 #define PassEspServer "87654321"
19
20 // Connection data to the user's local wifi network
21 #define ssidWiFiLocalSet "Poulose"
22 #define passwordSenhaWiFiLocal "2147707702"
23
24 // Data for connection to the "Firebase" database
25 #define FIREBASE_HOST "https://lawnmower-android-66b33.firebaseio.com"
26 #define FIREBASE_AUTH "4YOEWZHzqEOpZokJi5bhdcmPuiy15Qwgh45MKTO2"
27
28
29 // Define FirebaseESP8266 data object for data sending and receiving
30 FirebaseData fbdo;
31
32 // Variables
33 String receiveCar; // Car Received Data
34 String sendReceive; // Selection of active Tasks
35 String firebaseReceive; // Data Received from Firebase
36 String strJsonFirebase; // Data Json Received from Firebase
37 String strJsonClient = "{\"battery\":\" \"\", \"start_status\":\" \"\", \"latitude\":\" \"\", \"longitude\":\" \"\",}"; // Data Json Received from client
38
```

Figure 14: Snippet of WiFi MCU Code

4.2 Navigation ESP32

For the Nav_MCU, the functions were:

- Create WiFi Network so WiFi MCU can connect to it
- Send battery information/statistics to WiFi ESP
- Recieve coordinates & start_status from WiFi ESP

Below you can see the variables being sent and received to the Navigation ESP32

```
NavMCUwithoutNav
1#include <WiFi.h>
2#include <WiFiMulti.h>
3
4//Txt files
5#include <SPIFFS.h>
6#include "FS.h"
7
8WiFiMulti WiFiMulti;
9String receiveData;
10String battery ; // Percentage value of battery
11String start_status; // Car status
12String latitude; // Current car latitude
13String longitude; // Current car longitude
14String _0_latitude; // Orientation coordinates
```

Figure 15: Snippet of Nav MCU Code

4.3 Problems

Once I added Josh's old navigation code, the variables would send to each other but the navigation was not working. There were many errors and bugs that led to a rebooting issue with the Navigation ESP.

```
receiveData :[{"start_status":" ","_0_latitude":" 30.621404 ","_0_longitude":" -96.334650 ","_1_latitude":" 30.622638 ","_1_longitude":" -96.335838 ","_2_latitude":" 30.622638 ","_2_longitude":" -96.334650 ","_3_latitude":" 30.622638 ","_3_longitude":" -96.335838 ","_4_latitude":" 30.621404 ","_4_longitude":" -96.334650 "}]\n\nWaiting for WiFi...\nWiFi connected\nIP address:\n192.168.4.2\n30.6214\n30.621404\n-96.334650\n/home/runner/work/esp32-arduino-lib-builder/esp32-arduino-lib-builder/esp-idf/components/freertos/queue.c:1442 (xQueueGenericReceive)- assert failed!\nabort() was called at PC 0x40088869 on core 0\n\nBacktrace: 0x4008c434:0x3ffcf30 0x4008c665:0x3ffcf30 0x40088869:0x3ffcf30 0x400d6023:0x3ffcf30 0x400d1a32:0x3ffcf30 0x40088b7d:0x3ffcf30\n\nRebooting...\nets Jun  8 2016 00:22:57\n\nrst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)\nconfigsip: 0, SPIWP:0xee\nclk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00\nmode:DIO, clock div:1\nload:0x3fff0018,len:4\nload:0x3fff001c,len:1044\nload:0x40078000,len:8896\nload:0x40080400,len:5816\nentry 0x400806ac\nreceiveData :[{"start_status":" ","_0_latitude":" 30.621404 ","_0_longitude":" -96.334650 ","_1_latitude":" 30.622638 ","_1_longitude":" -96.335838 ","_2_latitude":" 30.622638 ","_2_longitude":" -96.334650 ","_3_latitude":" 30.622638 ","_3_longitude":" -96.335838 ","_4_latitude":" 30.621404 ","_4_longitude":" -96.334650 "}]\n\nWaiting for WiFi...\nWiFi connected\nIP address:\n192.168.4.2\n30.6214\n30.621404\n-96.334650\n/home/runner/work/esp32-arduino-lib-builder/esp32-arduino-lib-builder/esp-idf/components/freertos/queue.c:1442 (xQueueGenericReceive)- assert failed!\nabort() was called at PC 0x40088869 on core 0\n\nBacktrace: 0x4008c434:0x3ffcf160 0x4008c665:0x3ffcf160 0x40088869:0x3ffcf160 0x400d6023:0x3ffcf160 0x400d1a32:0x3ffcf160 0x40088b7d:0x3ffcf160\n\nRebooting...\nets Jun  8 2016 00:22:57
```

Figure 16: Rebooting Issue with Josh's code

4.4 Fixes/Improvements

To fix this problem, Max ended up rewriting the Navigation Code and I decided to combine all the code into one ESP to have one less step of the line of communication with the variables. This fixed the problem.

The functions for the final code:

- Connects to Wifi for Firebase Server
- Receives coordinates,start_status from Firebase Server in realtime
- Sends battery information/statistics to Firebase Server in realtime
- Navigates the Mower (explained in detail above in navigation section)

Added to new combined code

- Converts 4 coordinates into Distances which you can see in the figure below

```
Set string data success
Local-Control
start_status:
0
_0_latitude:
30.591616
_0_longitude:
-96.329182
Local-Control
_1_latitude:
30.592312
_1_longitude:
-96.328287
_2_latitude:
30.591753
_2_longitude:
-96.327686
Local-Control
_3_latitude:
30.591003
_3_longitude:
-96.328495
{"battery": " ", "start_status": "0", "_0_latitude": "30.591616", "_0_longitude": "-96.329182", "_1_latitude": "30.592312", "_1_longitude": "-96.328287", "_2_l
Local-Control

Distance to destination(M): 0.000115

Distance to destination(M): 0.000085

Distance to destination(M): 0.000114

Distance to destination(M): 0.000095
```

Figure 17: Shown Distances Calculated from the 4 coordinates (Serial Monitor)

4.5 Summary for ESP32 Code

The ESP32 was set up in STA Mode to connect to the WiFi network. In the final code/demo, we ended up using my iPhone as a hotspot to connect to. In the code you can see the network

name as "iPhone" and the password as "jonathan". By connecting to WiFi, I was able to receive the start_status (Start & Stop) boolean and the longitude and latitude from the coordinates selected in the App in realtime as well as send battery status/statistics to the server in realtime. The ESP checked the the server for these variables every 110 ms. The hardest part was finding the correct format from the Firebase Server & JSON file to display all of these variables correctly which is a lot of my code.

I added a while loop to the navigation code so that when start_status is 0, the mower will stop and when it is not, it will start.

I also added code to convert the coordinates (8 total variables) longitude and latitude into distances for the outside legs of the navigation using the haversine formula.

The WiFi part of the code worked fine without any issues.

5 Conclusion for User Interface/Server/WiFi

I am very satisfied with all the work I have put into this project. I think I did all I could and to the best of my ability. Although the lawnmower did not have hardware to check for battery level, I still added code to both ESP32s and UI that would allow this information to be passed through to the server and to the mobile application. I successfully created the user interface which had all of the following functions:

- Allows user to login through Google Account
 - as many google accounts as you want
 - saves information for each account
- Allows user to add multiple lawns using Google Maps API
 - Select Default Lawn
 - Select current location
 - Unlimited markers/corners for each lawn
 - records coordinates for each marker and sends to server
 - Numbered each corner from one to desired amount
 - Add lawn name
- Scheduling
 - Add time
 - Days of the week
- Control of Lawnmower
 - Start
 - Stop
 - Go Home
- Lawnmower status
 - Battery level
 - Other statistics
- Uploads all information to Firebase Server

6 Moving on

I was honestly scared to code my first mobile application because I had no idea how. After doing this project, I know that I will be coding more applications in the future which I am very excited about. I am glad I went through this so it will be easier for me in the future. Thanks and Gig'em.