

Autonomous Lawnmower

Max Lesser

MCU SUBSYSTEM REPORT

REVISION – 3
29 April 2021

Table of Contents

Table of Contents	25
List of Tables	26
List of Figures	26
1 Subsystem Introduction	27
2 Multi MCU interface	28
2.1 Interface Descriptions	29
2.2 As Built System	31
3 PIC MCU by Max Lesser	32
3.1 Hardware	32
3.2 Software	35
3.3 GPS	28
3.4 Ultrasonic Sensors	42
3.5 WIFI-Module	47
3.6 Drive Motor Control	47
3.7 Blade Motor Control	47
3.8 Accelerometer and Gyroscope Unit control	48
3.9 Battery Status Interface	48
3.10 MCU Functions Relegated to and Dependencies on Other Subsystems	49
3.11 PIC-32 MCU Subsystem Conclusion	50
4 ESP-N Subsystem by Max Lesser	51
4.1 Introduction	51
4.2 Navigation	52
4.3 Obstacle Evasion	53
4.4 Shaft Encoders	54
4.5 Accelerometer Gyroscope and Magnetometer	54
4.6 Motor Control	54
4.7 Blade Motor Control	55
4.8 Summary and Conclusion	55
5 ESP-W Subsystem	56

List of Tables

Table 1: UART Data Transfer Information	29
Table 2: GPS precision test results	38
Table 3: Distance and Error Percentage for small object detection	45

List of Figures

Figure 1: MCU Block diagram	28
Figure 2: PCB layout	33
Figure 3: General Program flow	36
Figure 4: Serial Output from PIC-32 towards ESP-N, with front and side obstacle readings, as well as parsed GPS output	37
Figure 5: GPS precision test result map	39
Figure 6: GPS accuracy test map	40
Figure 7: GPS accuracy test error to Google maps graph	40
Figure 8: GPS accuracy test Error between measurements graph	41
Figure 9: Ultrasonic sensor detection of large object test result graph	43
Figure 10: Percent Error for Large Object Detection	43
Figure 11: Ultrasonic sensor detection of small object test result graph	44
Figure 12: Percent Error for small object detection	45
Figure 13: Ultrasonic sensor detection of person test result graph	46
Figure 14: Percent Error for Person detection	46
Figure 15: Battery read circuit simulation for “Battery Charged” (t) and “Discharged” (b)	49
Figure 16: General Navigation flow without obstacle evasion	52
Figure 17: Obstacle Detection Flow Chart	53

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.

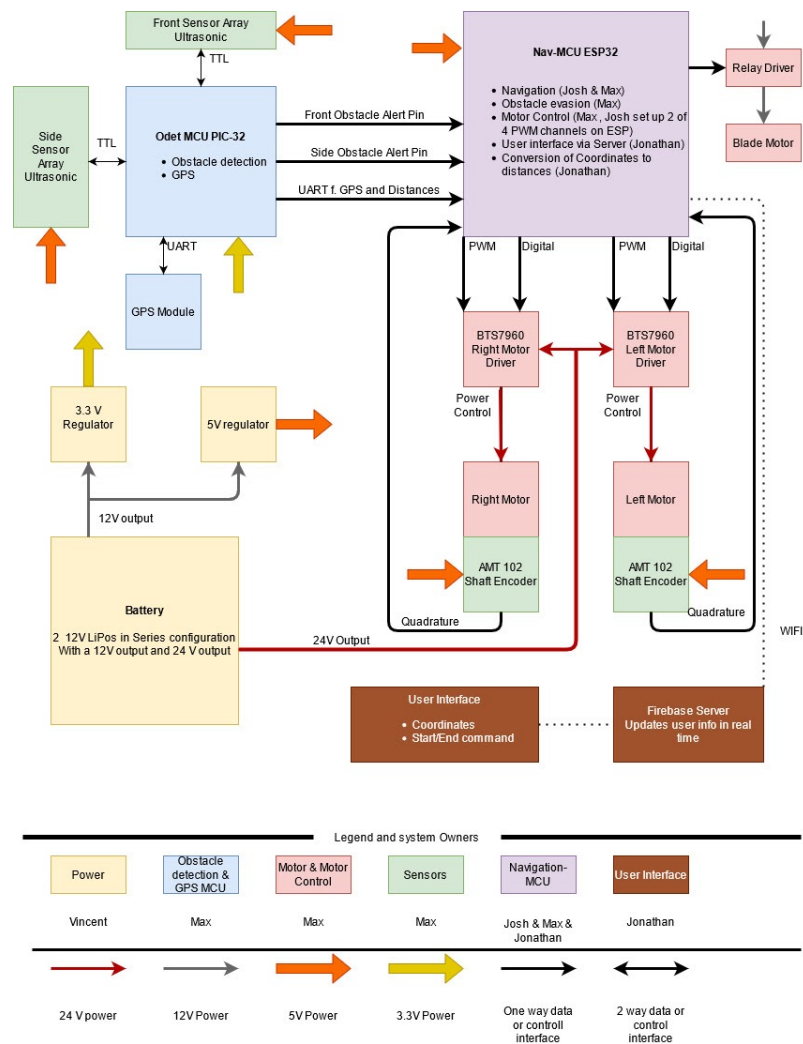


Figure 1: MCU Block diagram

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.

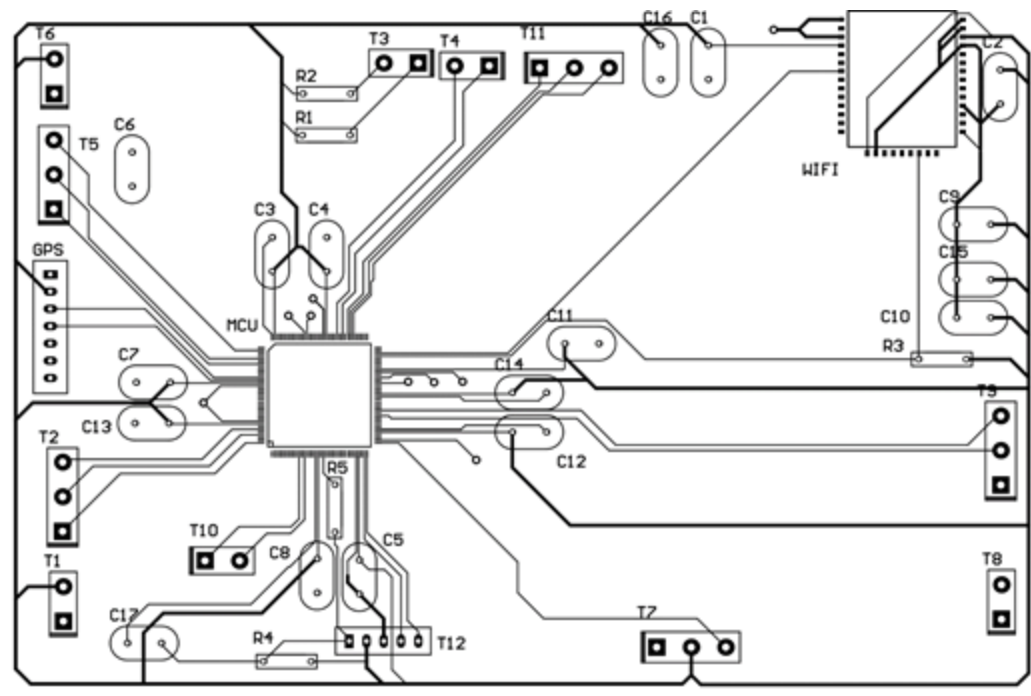


Figure 2: PCB layout

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.

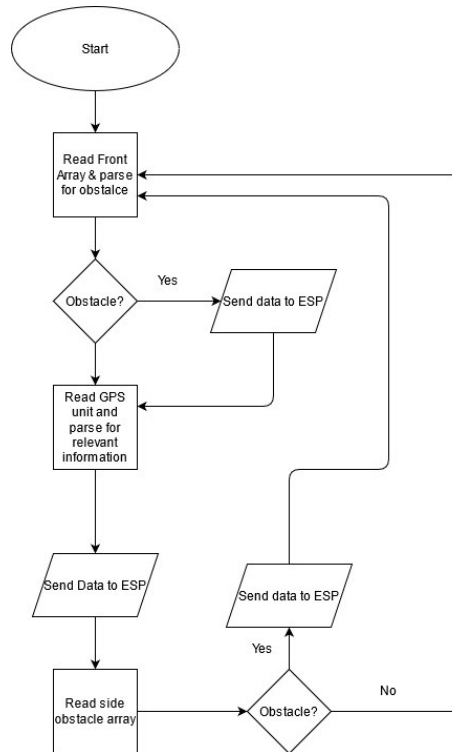


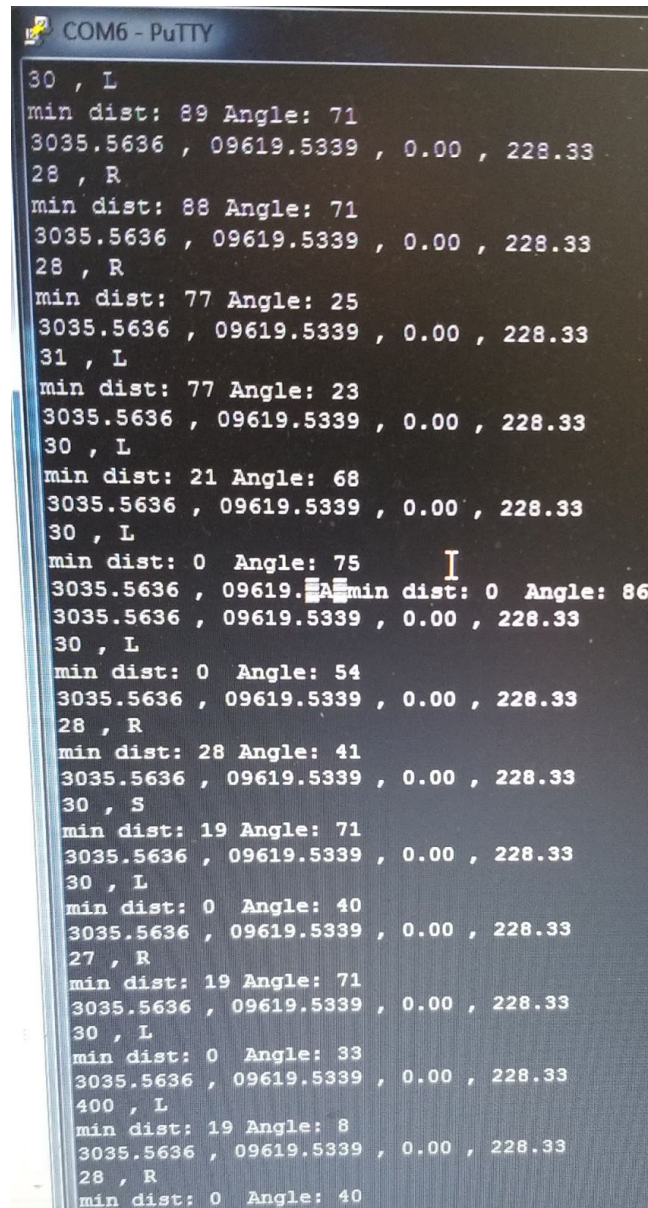
Figure 3: General Program flow

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
31 , L
min dist: 77 Angle: 23
3035.5636 , 09619.5339 , 0.00 , 228.33
30 , L
min dist: 21 Angle: 68
3035.5636 , 09619.5339 , 0.00 , 228.33
30 , L
min dist: 0 Angle: 75
3035.5636 , 09619.5339 , 0.00 , 228.33
min dist: 0 Angle: 86
3035.5636 , 09619.5339 , 0.00 , 228.33
30 , L
min dist: 0 Angle: 54
3035.5636 , 09619.5339 , 0.00 , 228.33
28 , R
min dist: 28 Angle: 41
3035.5636 , 09619.5339 , 0.00 , 228.33
30 , S
min dist: 19 Angle: 71
3035.5636 , 09619.5339 , 0.00 , 228.33
30 , L
min dist: 0 Angle: 40
3035.5636 , 09619.5339 , 0.00 , 228.33
27 , R
min dist: 19 Angle: 71
3035.5636 , 09619.5339 , 0.00 , 228.33
30 , L
min dist: 0 Angle: 33
3035.5636 , 09619.5339 , 0.00 , 228.33
400 , L
min dist: 19 Angle: 8
3035.5636 , 09619.5339 , 0.00 , 228.33
28 , R
min dist: 0 Angle: 40
```

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.

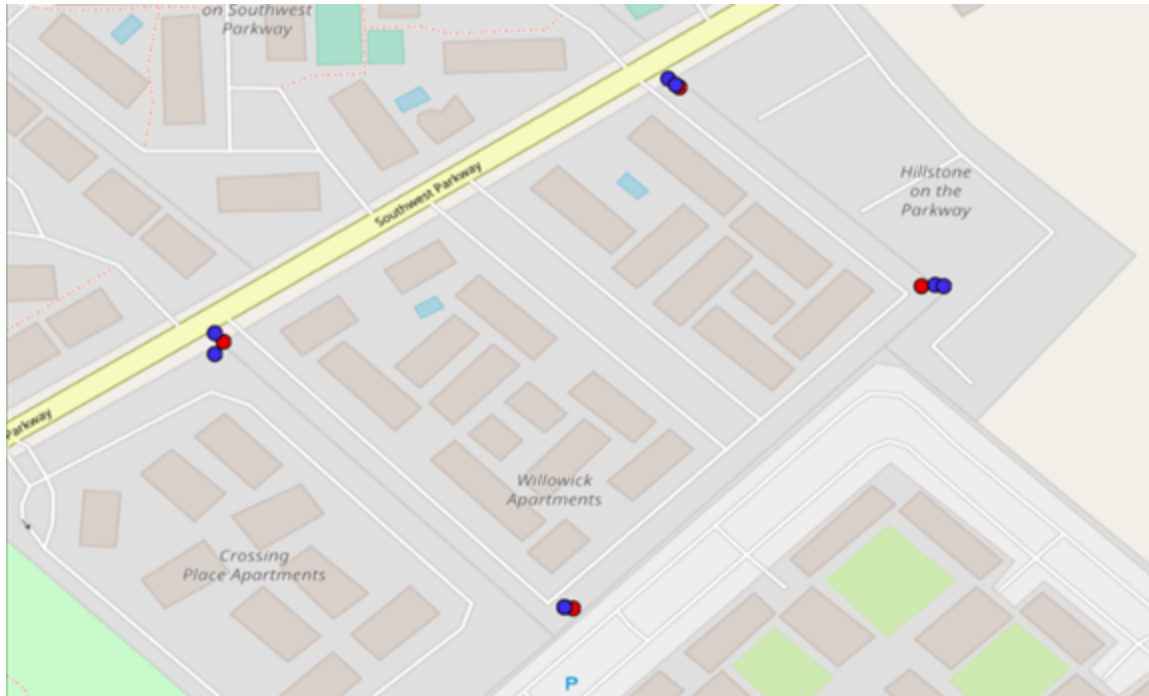


Figure 6: GPS accuracy test map

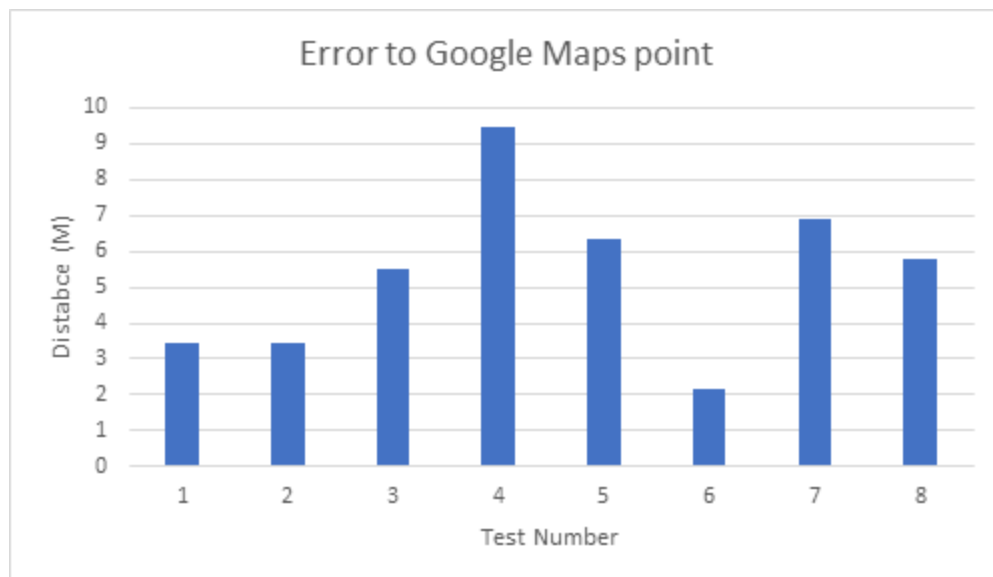


Figure 7: GPS accuracy test error to Google maps graph

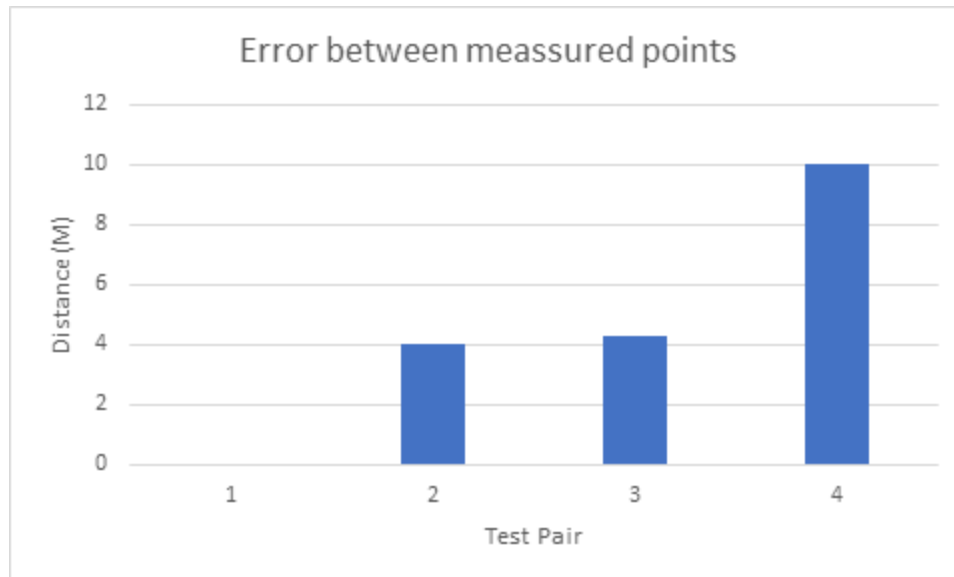


Figure 8: GPS accuracy test Error between measurements graph

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 μ 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 360.3 m/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 4x4ft whiteboard very accurately up to 3+M, the maximum distance we were able to test. A 3.5x5.5 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.

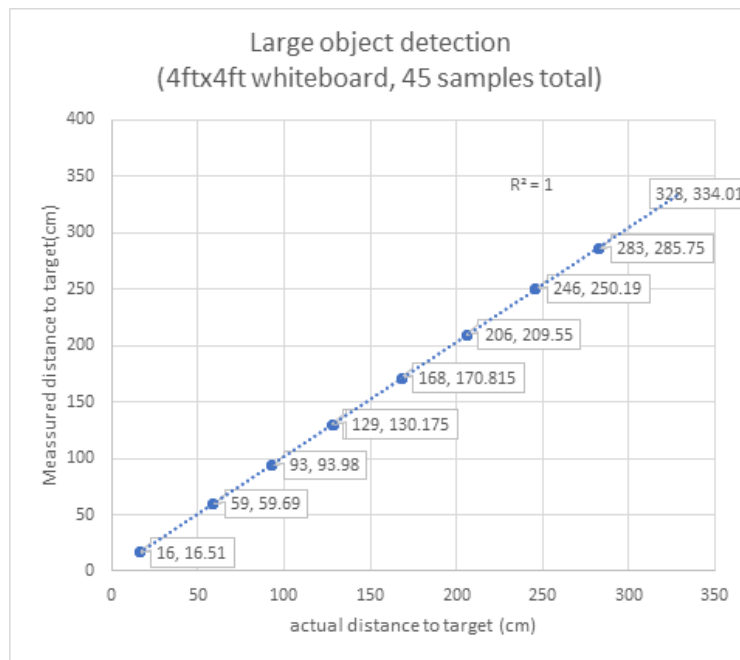


Figure 9: Ultrasonic sensor detection of large object test result graph

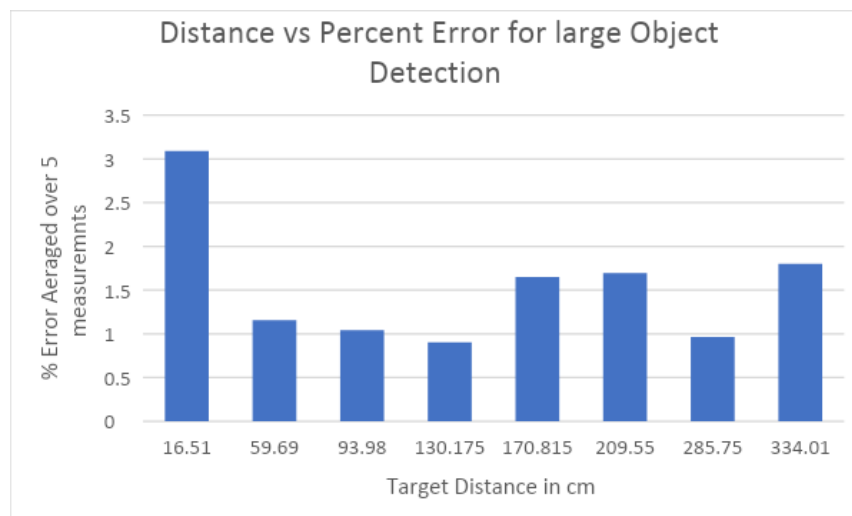


Figure 10: Percent Error for Large Object Detection

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.

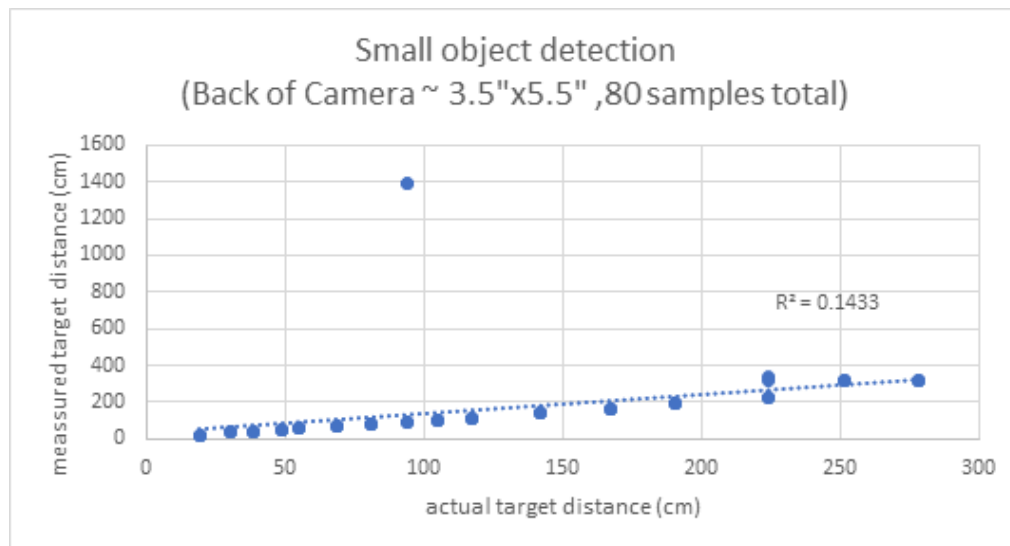


Figure 11: Ultrasonic sensor detection of small object test result graph

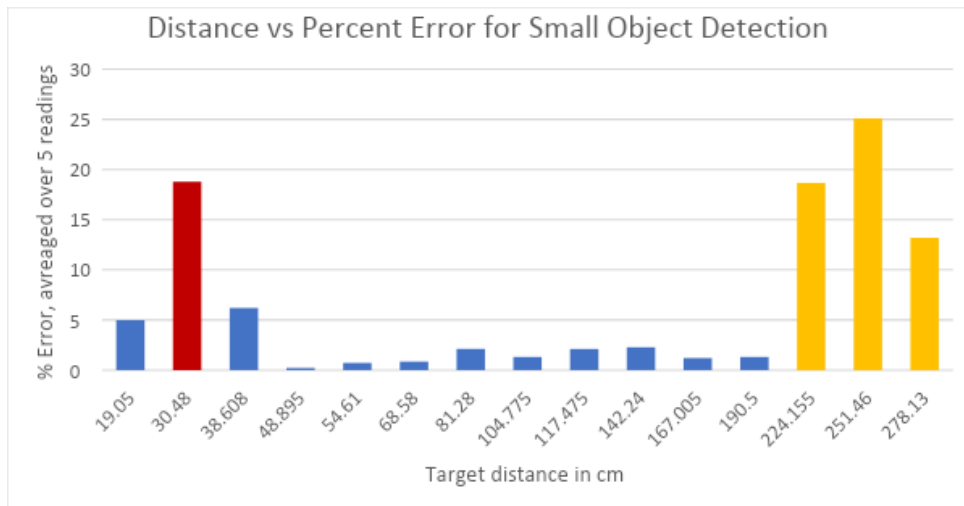


Figure 12: Percent Error for small object detection

Small Object Detection Error		
Area	Target distance in cm	% Error to target
	19.05	4.98687664
	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.

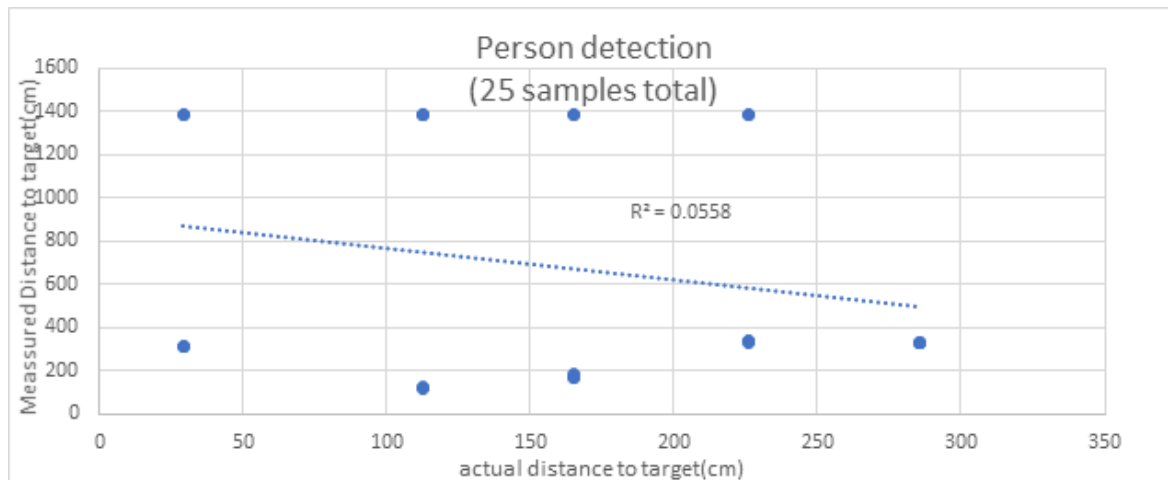


Figure 13: Ultrasonic sensor detection of person test result graph

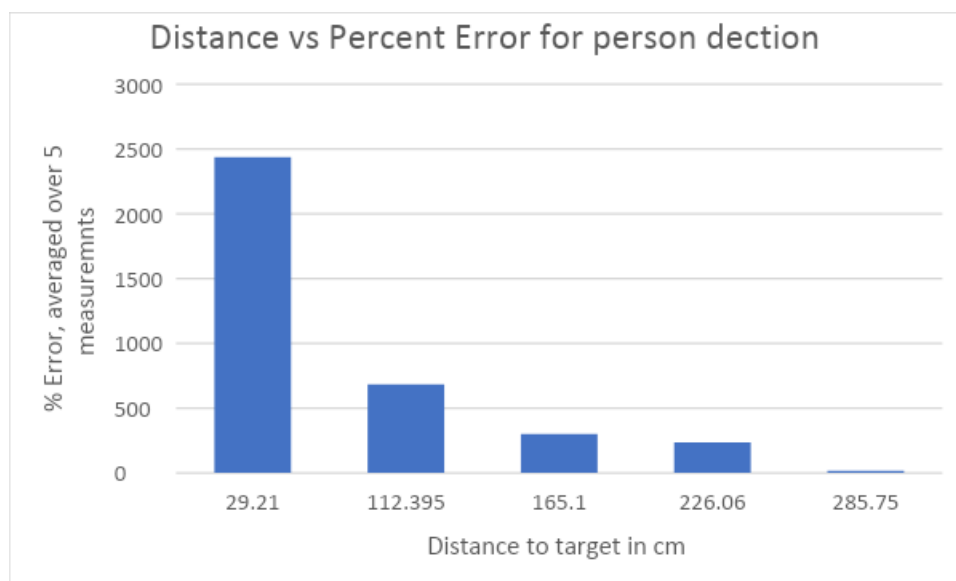


Figure 14: Percent Error for Person detection

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.

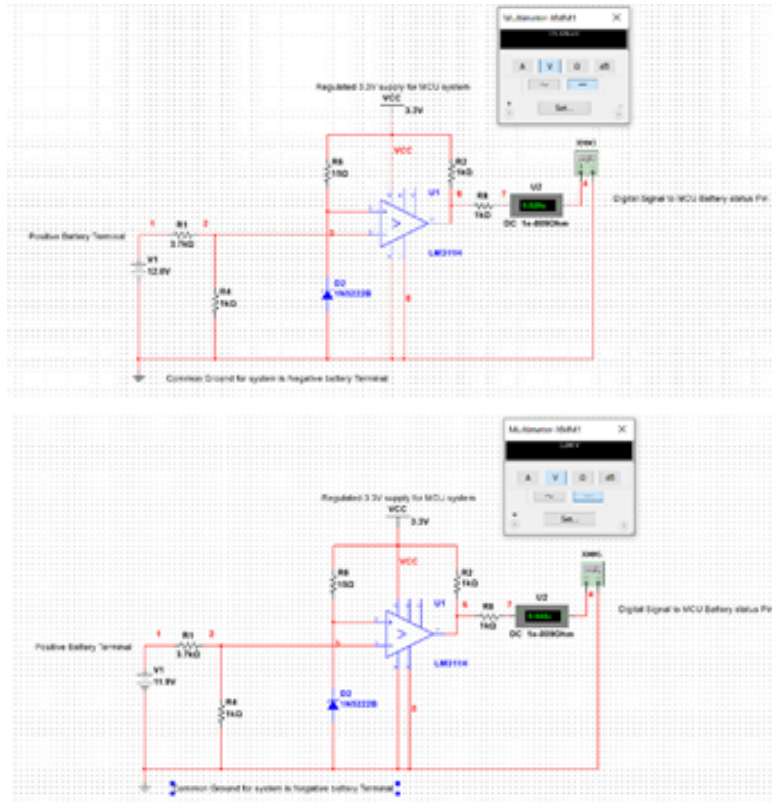


Figure 15: Battery read circuit simulation for “Battery Charged” (top) and “Discharged” (bottom)

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.

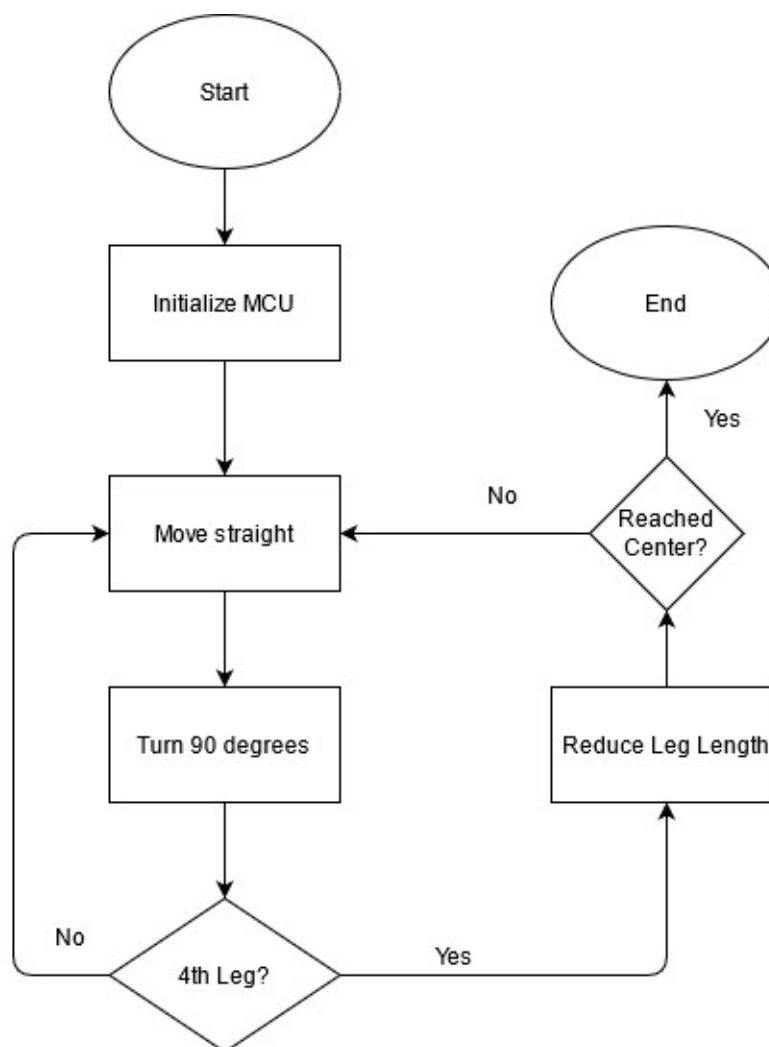


Figure 16: General Navigation flow without obstacle evasion

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.

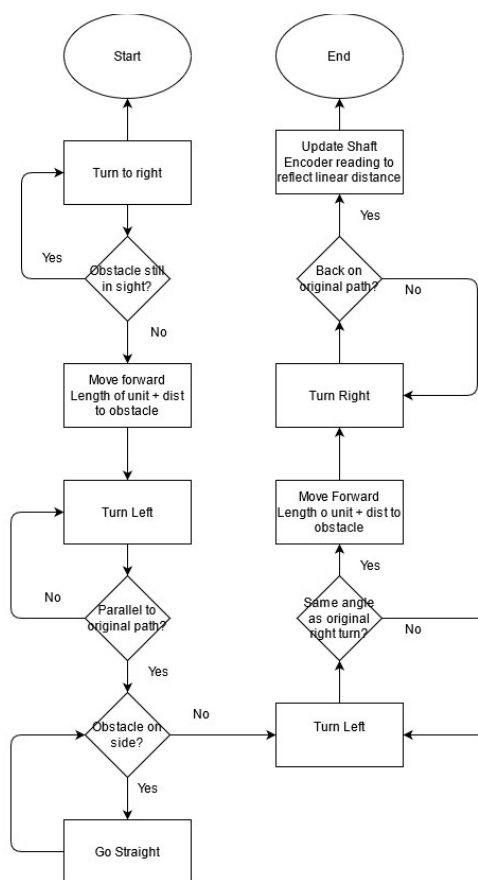


Figure 17: Obstacle Detection Flow Chart

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