Autonomous Lawn Care Unit

Final Report

Team Number 22
Micron Technologies
Dr. Phillip Jones

Sam Tinklenberg, Andi Li, Bryton Hayes, Grant Duncan, Joel Seaser

Email: sddec18-22@iastate.edu
Website: https://sddec18-22.sd.ece.iastate.edu

Table of Contents

1 Introductory Material

1.1 Acknowledgement

If a client, an organization, or an individual has contributed or will contribute significant assistance in the form of technical advice, equipment, financial aid, etc., an acknowledgment of this contribution shall be included in a separate section of the project plan.

1.2 Problem Statement

The problem we intend to solve concerns the time and financial commitment required to upkeep a well-groomed lawn. There is a long list of reasons a certain individual may not be able to mow their lawn, ranging from lack of time to physical incapabilities. Someone who falls into this category does not have many options to get the job done, without hiring expensive, third-party help.

1.3 Requirements

- Algorithm to efficiently mow entirety of area given a mapped perimeter
  - Intelligent routes will ensure straight lines
- System to map perimeter of the lawn
  - Easy installation
  - Low power
- Object detection and avoidance
  - Use various sensors to accurately detect and avoid objects
  - Navigate around hazardous areas in a lawn
- Mobility through standard lawns
  - Drive up 30% grade
  - Lightweight
- Algorithm to detect and avoid safety concerns
  - Easy shutoff consistent with sensor stimuli
  - Indicate current state
- Power efficiency
  - Auto-charging feature
  - Charge and mowing time to fall between 60-90 minutes
- Streamlined interfacing
  - Android app for communication and diagnostics
- GPS module for directional guidance and mapping

1.4 Constraints Considerations

- Battery life to complete job or return to charging station
- Lightweight enough to be easily moved by hand
- Algorithm execution rate of 100Hz
- Design consists of affordable parts, and is reasonable for the domestic market
- 15% incline maximum
- 3.5 inch maximum grass height
- The motors will shut off if the mower is lifted off the ground
- Bump sensors to prevent unwanted mowing of objects

2 Revised Project Design

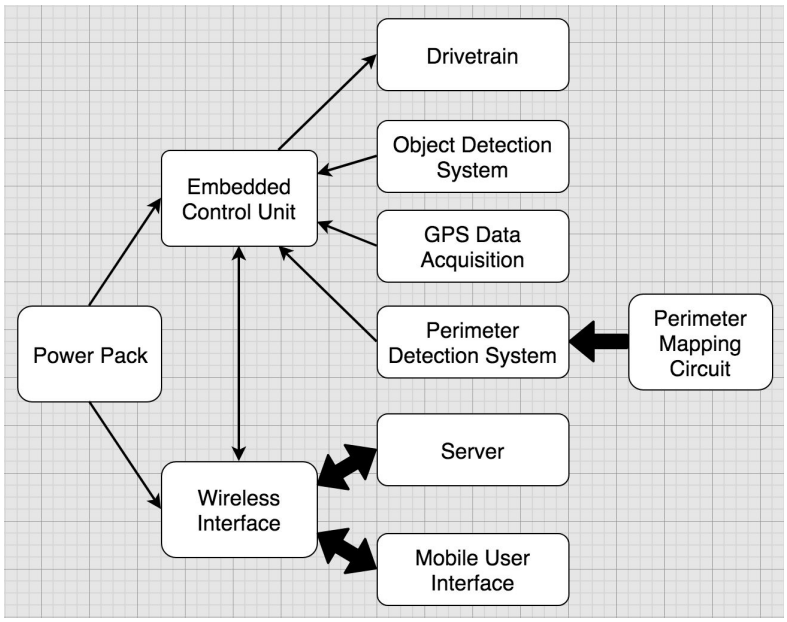


Figure 2 - Conceptual Diagram

2.1 Chassis, Blade, and Drivetrain

For our proof of concept, we decided to utilize aluminum C-channels. These allow us to create a lightweight, durable and modular chassis to mount all of our components for the mower. An acrylic plate mounted to the top of this chassis will give us room for all electrical and control units needed to run the mower. The C-channels are very universal and provide a simple method of connecting the wheels, motors, and blade in the future. We chose to utilize a reel-style blade in lieu of the traditional horizontal spinning blade used on most modern

mowers. This type of blade is much safer since it spins at a much lower RPM and provides a much cleaner cut. The blade will be driven by a high rpm DC motor.

The drivetrain consists of two DC motors coupled with planetary gearboxes to provide high torque on the drive wheels. Two 8-inch rubber placing-tread wheels will be mounted directly to the motors using hex-hubs and the motors will be connected to the C-channel with motor mounts. Two 4-inch 360-degree swivel caster wheels mounted to the front of the chassis will provide near zero-turn capabilities.

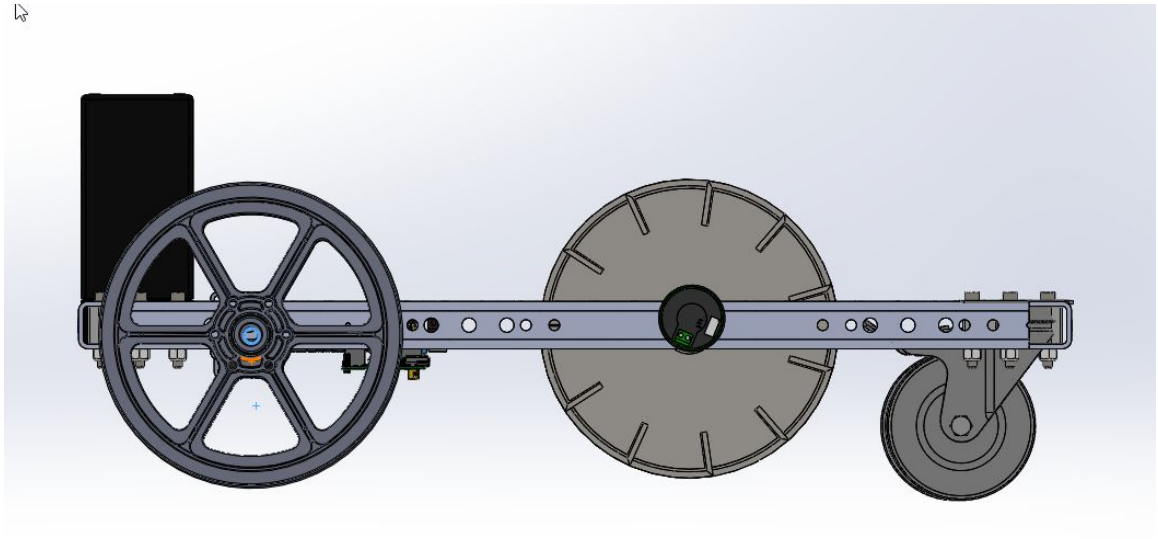


Figure 2.1.1 - 3D Model Side View

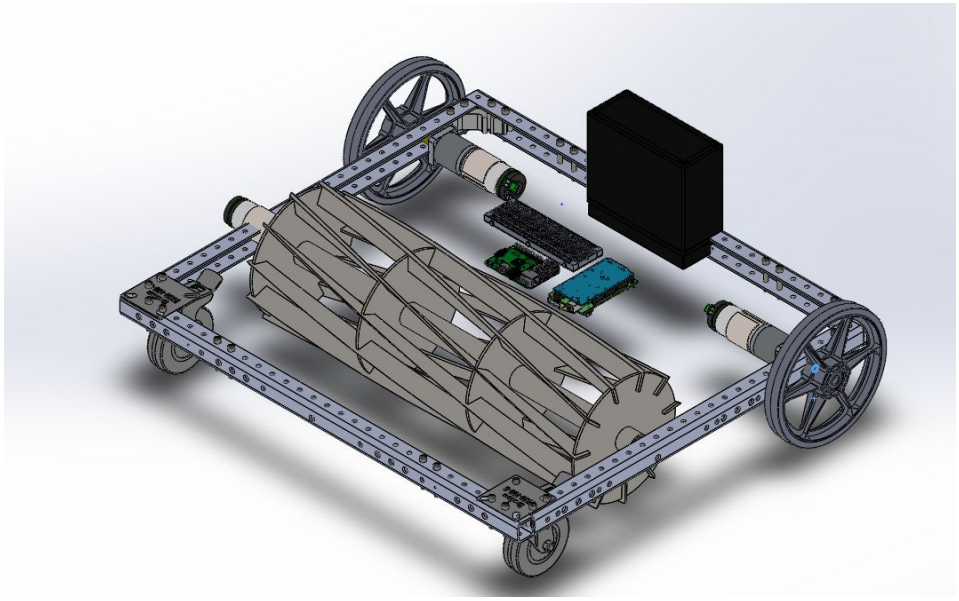


Figure 2.1.2 - 3D Model Top View

2.2 Power System

For our lawn mower we decided to go with a 12V 18Ah battery. This would ensure that we would have a average of 60 minutes of runtime so that we could mow the average US lawn on one charge. The motors for the wheels are 775 motors with a PG71 gearmotor with a gearbox reduction of 71:1. These gave us our desired max torque of 11.5Nm and a max velocity of 6 ft/s. The safety features that we included for the electrical system include a fuse block with 30A breakers for the motors and a 12V to 5V voltage step down for the Arduino and Raspberry Pi. We also included a manual breaker switch so that in the event of a emergency you can completely cut power to the mower.

2.3 Embedded Control System

The main control system for the embedded components on our mower is based on an Arduino Mega 2650. We chose this device because it is cost-effective, has sufficient GPIO, has multiple hardware serial ports, and allows for easy integration with custom C++ libraries. This system is in charge of controlling the motors, data transfer with the wireless interface (see section 2.4), interpretation of position data, and object detection (see section 2.6).

The Arduino built-in functions only support 8-bit PWM signals but we decided to set up the timers for 16-bit resolution to gain a little better control. The Arduino Mega has 2 built-in 16-bit timers that we were able to utilize to achieve this. The timers were set up to use non-inverting, fast PWM. The Arduino has a 16MHz clock and dividing that by 2^16 yields a 4ms period, which is well within the operating range of the motor controllers. We then set a match value for the timer to achieve the desired high pulse. The motors each include a 2-channel hall effect encoder that gives us the ability to set up a control loop and verify that we reached the desired RPM.

2.4 Wireless Interface

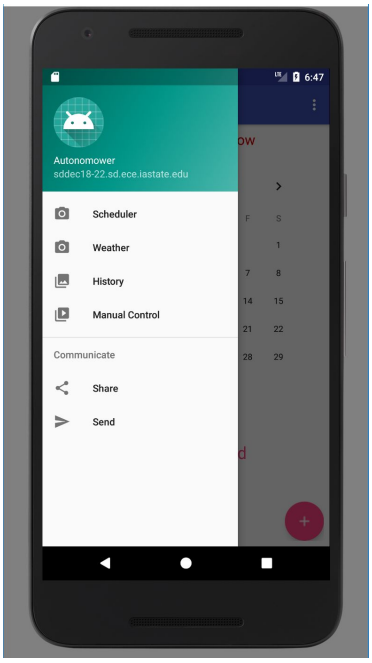
Communications between the mower and the android app will be handled by the raspberry pi. The pi has a wireless card that allows it to be added to a wireless network. For information to be sent to the mower, the phone and pi both need to be on the same local network. When the phone is unable to contact the pi, it will store the information in an SQLite database on the phone. Once communication has been restored, the information will be sent to the pi. The raspberry pi is hosting a REST API that is being served with a flask. The backend, such as the database interactions, is built in python. The pi is using MariaDB to run the database. Information is sent to the Arduino, which is controlling the mower, over a physical

serial connection. All communication is formatted in JSON since there are many libraries that can handle JSON efficiently.

2.5 Mobile Application

The mobile application for this project was done for Android devices. This choice was because over 50% of mobile users in the United States use Android, and it has a free development platform. The code was written in Java, Kotlin, SQL, and XML. Three third-party APIs were used, OpenWeatherMap, Google Maps, and Room Persistence. OpenWeatherMap was used to fetch the current weather data for the app, Google Maps was used to show the mowing history coordinates, and Room was used as a layer of abstraction over SQLite. The app uses MVVM (Model/View/ViewModel) pattern, which is nice because the fragments can share data easily, observers can be placed on data, screen rotation is handled nicely, and the database is not directly accessed by any fragment.

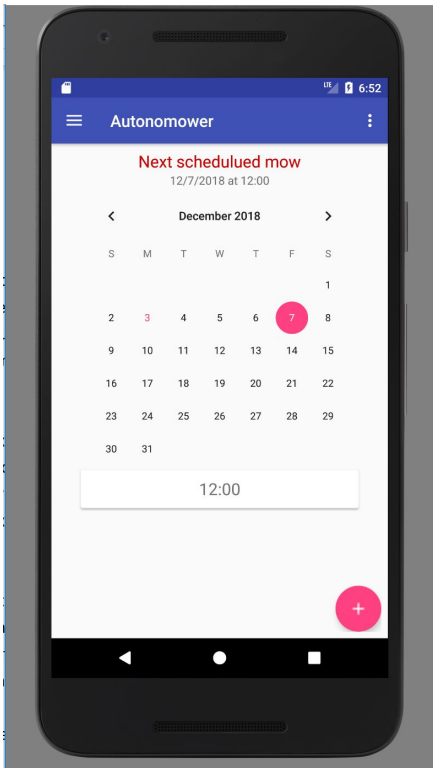
The app consists of four sections. A side navigation tray is used to select among Scheduler, Weather, History, Manual Control. Each of these services are handled using fragments, and the fragments are contained in a single activity.



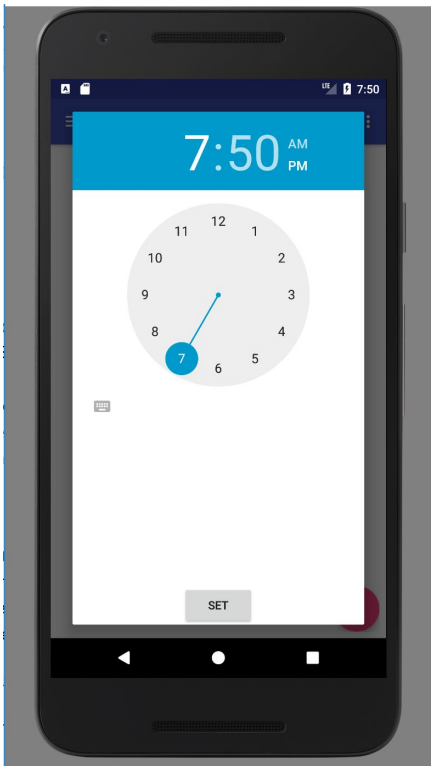
Navigation Menu

In Scheduler, the user can see, at a glance, when the next scheduled mow is at the top of the screen. They also have an interactive calendar where they can see, add, or update new mowing schedules. The schedule data is stored using a Room database. When the user adds,

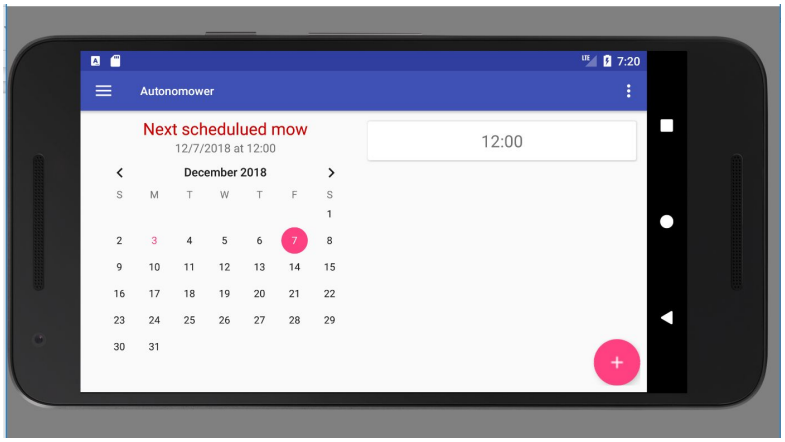
updates, or removes a scheduled mow, a new entity is added to the SQLite database and the next mow time is recalculated.



Scheduler

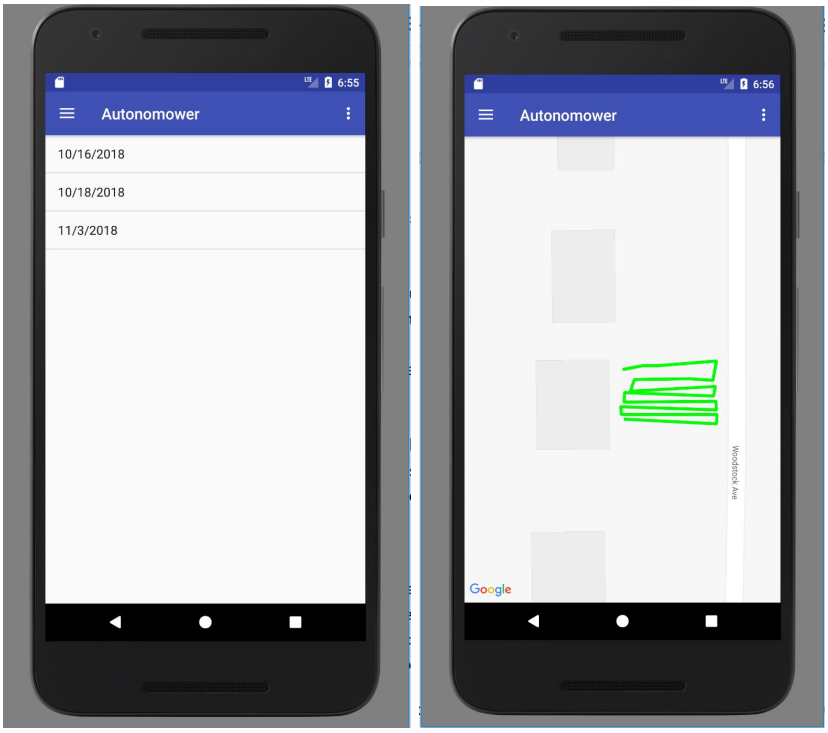


TimePicker



Scheduler (Landscape)

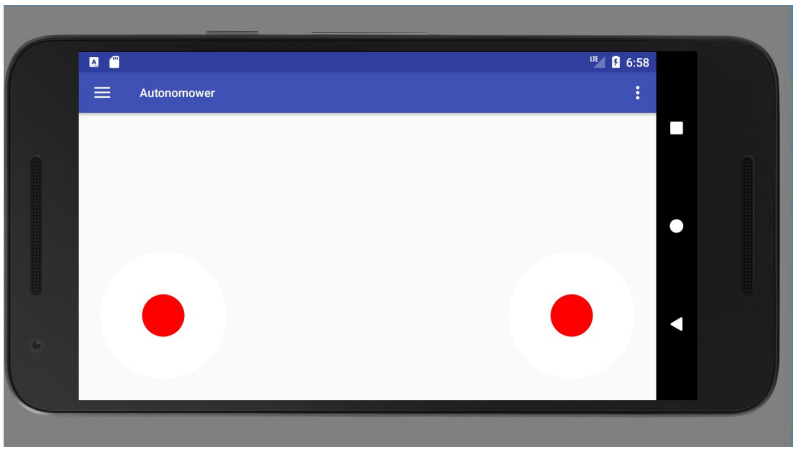
The next section of the app is History. The user sees a list of past mows, which they can select and see data about that mow. Google Maps API is used to show the GPS path that the mower took overlayed a satellite-view map.



History Selection Screen *History Coordinates View*

The next aspect of the app is Weather. This screen shows the current weather for the location of the mower. This part of the app also will send a notification to user to alert them of rain in the area. Weather uses the OpenWeatherMap API to retrieve up-to-the minute data.

The last part of the app is Manual Control. When opened, the app will request a Bluetooth connection with the mower. When connected, the user can manually control the mower via joysticks or by use of the accelerometer.



Manual Control

The connection between the app and mower refreshes when the app is on the same network as the mower. When the user adds, removes, or updates the schedule, or when the user clicks refresh, the schedule data is sent and history data is retrieved from the mower.

2.6 Position Acquisition and object detection

For the perimeter wire, we used an NE555 timer to create a square wave that would operate between 32 kHz and 44 kHz. This signal would be sent through a wireless than 300' that the user or installer would put around the perimeter of the lawn. For our detection system, we have a LM324N op-amp connected to the two 1mH inductors that will read the analog signal into the Arduino which will then output the signal digitally between 0 and 1024 the closer that you get to the perimeter. The inductors will be placed at the front corners of the lawn mower.

3 Testing

We ran into a lot of time constraints and issues ordering the necessary parts to get the proof of concept assembled. We learned a lot this semester about how third-party contributors can affect the success of a project. We were able to test all individual components of the system but were not able to test a lot of aspects about their operation in the realm of the entire system.

3.1 Motor Controller

We were able to successfully run multiple motors with 16-bit resolution using a custom C++ library to interface with the embedded control unit. 16-bit control provided much smaller error and cleaner signals than 8-bit control (See figures 3.1.1 and 3.1.2). The process for verifying motor control included powering the motor controller with our battery and controlling speed and direction by sending a pulse between 1ms and 2ms (See figure 3.1.3). A 1ms pulse corresponds to full speed clockwise, 1.5ms corresponds to stopped, and 2ms runs the motor full speed counter-clockwise.

The result was successful motor control of multiple motors simultaneously with negligible error in our waveforms. We set up the Arduino to read the hall effect sensors on the motors using the built in pulseIn() function on digital pins. Our findings were consistent with what we expected when testing various speeds.

	Expected Pulse Width (ms)	Expected Duty Cycle (%)	analogWrite() Value	Measured Pulse Width (ms)	Measured Duty Cycle (%)	Error in Pulse Width (%)
Full Speed Backward	1	4.896	12 13 (12.48)	.974 1.056	4.765 5.141	2.6 5.6
Half Speed Backward	1.25	6.12	15 16 (15.606)	1.216 1.293	5.956 6.332	2.72 3.44
Stopped	1.5	7.344	18 19 (18.727)	1.456 1.533	7.085 7.461	2.93 2.2
Half Speed Forward	1.75	8.568	21 22 (21.848)	1.693 1.776	8.276 8.652	3.26 1.49
Full Speed Forward	2	9.792	25 (24.97)	2.008	9.812	0.4

Figure 3.1.1 - 8-bit PWM Analysis

	Expected Pulse Width (ms)	Expected Duty Cycle (%)	setMotor() value	Measured Pulse Width (ms)	Error in Pulse Width (%)
Full Speed Backward	1	24.414	16000 (15999.71)	.9991	0.09
Half Speed Backward	1.25	30.518	20000 (19999.97)	1.249	0.08
Stopped	1.5	36.621	24000 (23999.57)	1.499	0.06
Half Speed Forward	1.75	42.725	28000 (27999.83)	1.748	0.11
Full Speed Forward	2	48.828	32000 (31999.43)	1.998	0.1

Figure 3.1.2 - 16-bit PWM Analysis

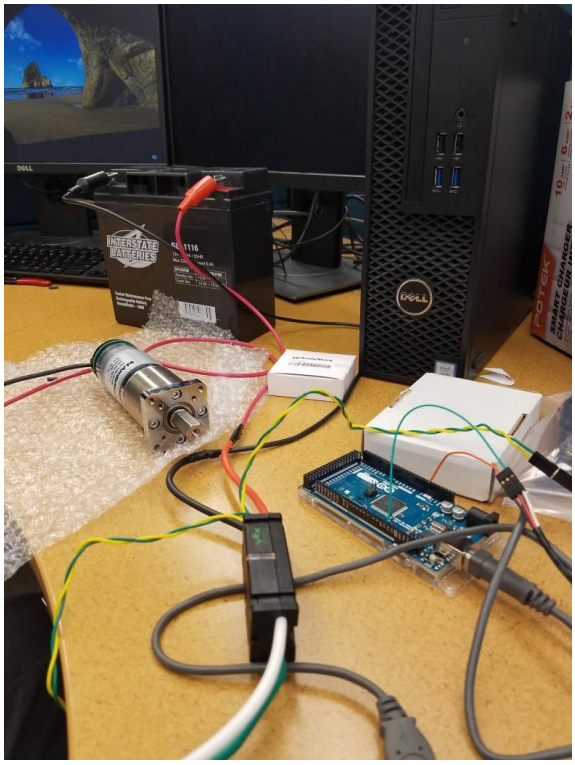


Figure 3.1.3 - Motor control testing setup

3.2 Position Acquisition

The perimeter wire usability was verified by connecting a small loop of wire to the emitter circuit and reading the output of the detection circuitry with the Arduino analogRead() functionality when it was at varying distances and orientation to the loop (See figures 3.2.1 and 3.2.2). We used an oscilloscope to verify both the square wave being emitted upon the wire and the output of the detection circuitry (See figure 3.2.3). This gave us an easy way to tune the emitter to a resonance frequency consistent with the inductor being used by adjusting a potentiometer on the emitter circuitry. We successfully read the analog signal produced by the detection circuit when approaching the perimeter wire (See figure 3.2.4).

It became difficult to test with real GPS data without the physical device fully assembled. To get around this and verify successful parsing of NMEA 0183 data, we generated some spoofed tracks that represented expected paths of the mower (See figure 3.2.5). We were able to successfully parse the desired information such as latitude, longitude, time, and speed. These 4 fields were assembled into JSON format to be transmitted to the Raspberry Pi over RS232 serial.

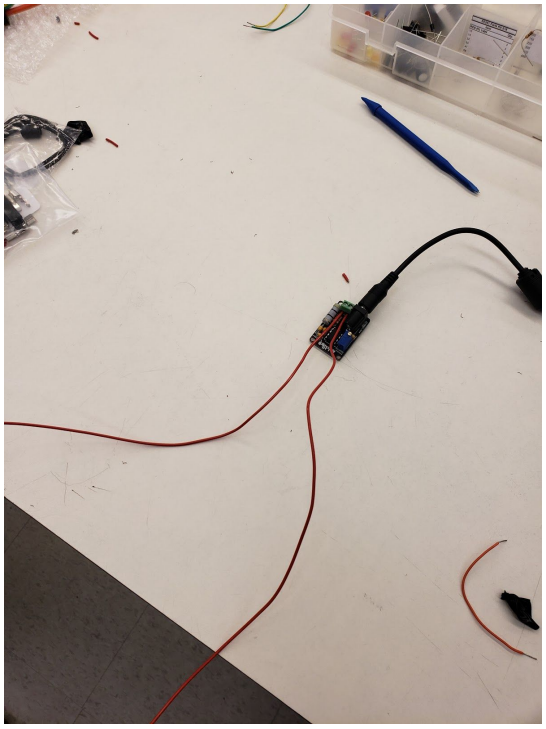


Figure 3.2.1 - Perimeter Wire Emitter Circuit

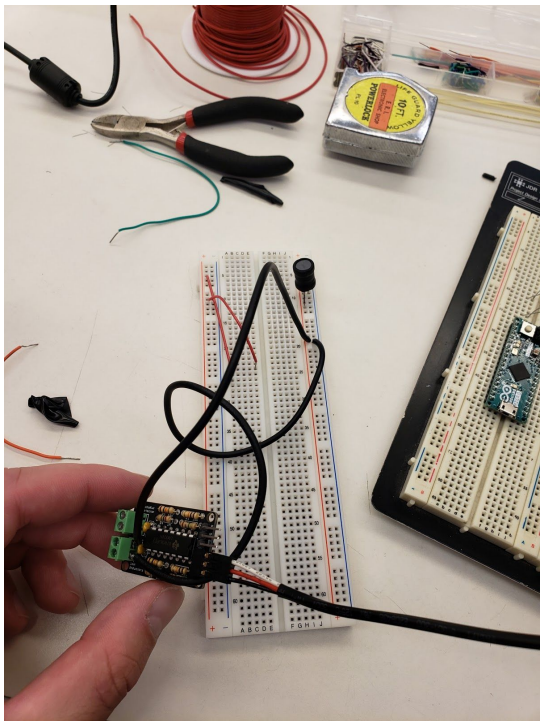


Figure 3.2.2 - Perimeter Wire Detection Circuit

Figure 3.2.3 - Perimeter Wire Emitter Square Wave

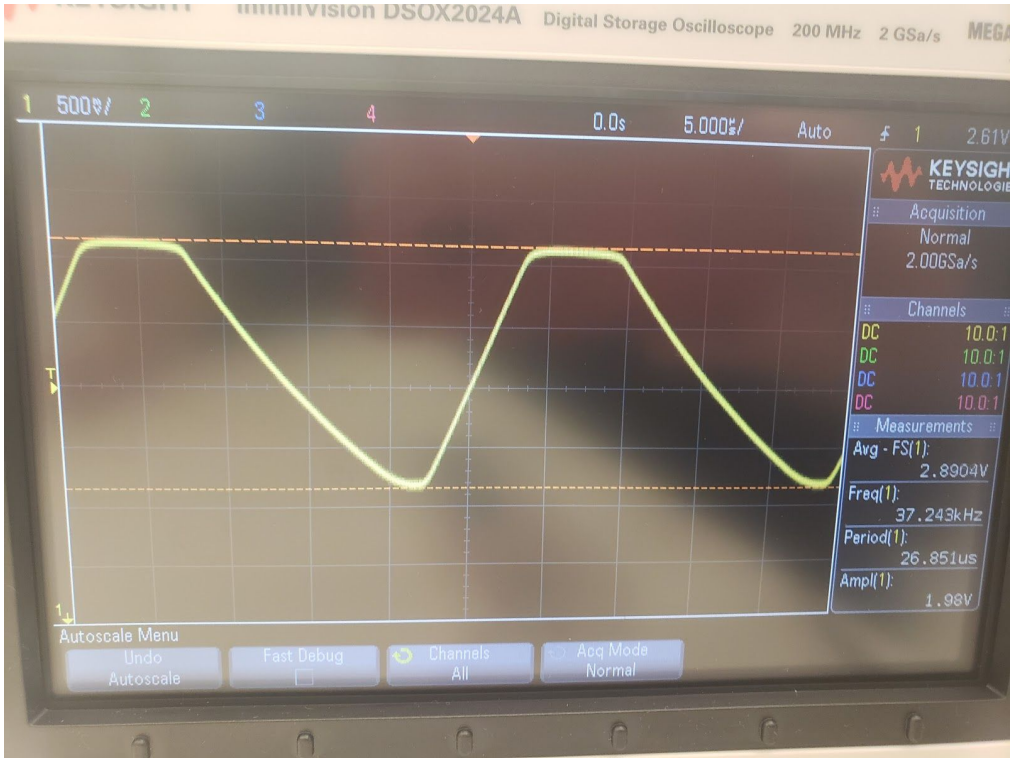


Figure 3.2.4 - Perimeter Wire Detection Signal

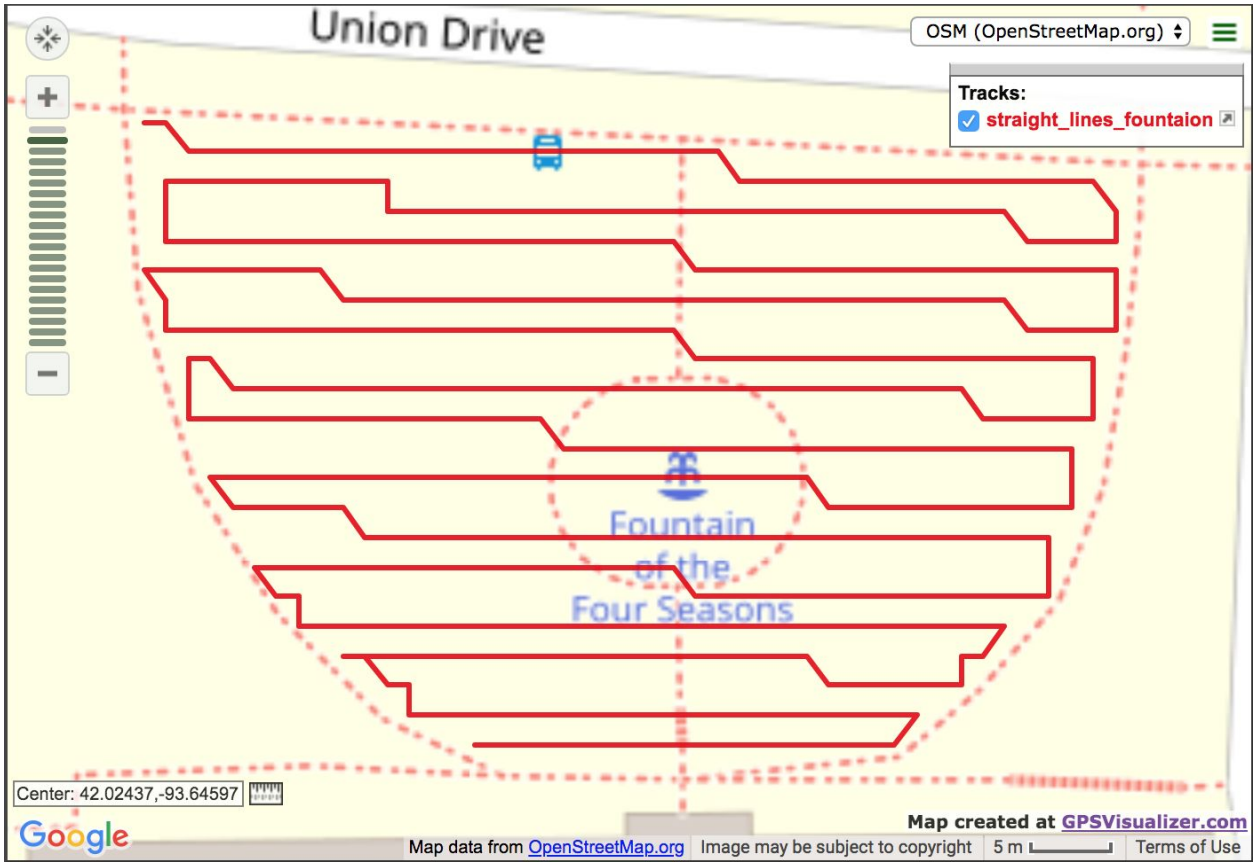


Figure 3.2.5 - Simulated GPS track

3.3 WireLess Communication Testing Communication

Wireless communication was quite straightforward. Once we had the REST API running, it was as simple as sending POST and GET requests to it. I did this from my own computer on the same local network as the pi. I sent a GET request to get the schedule and the pi responded with the schedule in a JSON. Next, I sent a POST request to the pi. This POST request contained information that was to be added to the schedule. To verify, I sent another GET request to the pi and verified that the date I sent in the POST request was n

3.4 Mobile Application testing

The testing for the mobile app mostly revolved around testing the SQLite database. Verifying that data was accurately captured by it was done using the Device File Explorer tool, which shows the content of all of the databases on the phone. Testing involved adding a new schedule, updating a schedule, and deleting a schedule, then verifying the data at each step. Testing was also performed by slowing down queries and closing the app in the middle of the query to verify that the database does not become corrupted.

Testing for common app crashes was also performed. In each screen, the app was rotated and verified that it would come back to the open fragment and retain the data in it. Similarly, the app was placed in the background and then navigated back to ensure the data was not lost. These kinds of tests are important due to the life cycles of Android. When rotating, Android destroys all the activities and fragments, then recreates them. If precautions are not taken, then data will be lost and screen position will revert to the home screen.

4 Context

4.1 Market Research/Related Products

There are many autonomous lawn mowers from several large companies which are currently for sale. One of the first things we did was to see what other companies were doing to get a general idea of what to expect. We made a spreadsheet with the specifications of many different autonomous lawn mowers to decide what our mower should be capable of. Currently, the top companies which sell autonomous lawn mowers are Husqvarna, Worx, and Robomow. Their mowers currently function very similar to that of a Roomba vacuum and use a rotating blade to cut grass within a set perimeter wire. We also took a look at some other home-made lawn mowers on Youtube, but instead of using a Roomba vacuum many of them used a metal chassis with tank treads for maneuvering and several of them used a reel blade. All these products were very useful in helping create or prototype.

4.2 Factors of Design

After reading through the data from the spreadsheet we determined the most important factors that our mower should try to optimize: Price, cutting width, and operating time on one charge. Currently, in the autonomous mower market, the average cost of a mower is between $2500 and $3000 USD. These mowers have an average cutting width of about 12 inches with an average operating time of one hour. There were numbers we were looking to improve on. With our proposed design our current cost is under $1000 USD with a mowing width of 14 inches and a predicted operating time of one hour.

5 Conclusion

5.1 Conclusion

Overall, this project was a big learning experience for our team. We learned the hard way as a team not to put all our eggs in one basket like we did when we ordered parts from a third party seller. We also learned the importance of communication, as our project had lots of separate parts. Due to this, our group had to be in constant communication with one another to make sure we were always on the same page. Although, we were not able to get a successful prototype built due to extenuating circumstances with parts being out of stock. We were able to come up with what we believe is a successful autonomous lawnmower design.

Appendices

I. Operation Manual

I.A Device Setup

The C-channels for the chassis are to be connected together in a square using the C-base corner connects. The drive motors are attached to the chassis using two mounting brackets and mounted on the bottom of the chassis for optimal ground clearance. The ½ inch hex output of the gearboxes are designed to be easily coupled with the hex hubs and mounted to the 8-inch plastic-tread wheels. A retaining ring ensures the wheels will not disconnect from the motor shaft. Two 4-inch caster wheels shall be bolted straight through the c-channel at the front of the mower.

The motor controllers are to be powered by a 12V line coming from the battery which is connected to the fuse-box. The motor controllers are then both connected to the motors to supply power and detect feedback. A 12V to 5V regulator will be used to power the Arduino, Raspberry Pi, motor encoders, and perimeter wire detection circuit. The GPS module is powered by the 3.3V output of the Arduino.

Both the android phone and raspberry pi must be connected on the same wireless network. As of right now, the pi is set to connect to the eduroam network on campus automatically.

I.B Running Motors

The drive motors are controlled by 16-bit timers 1A and 1B on the Arduino and their output corresponds to digital pins 11 and 12, respectively. The left drive motor will be controlled by timer 1A and the right drive motor is controlled by timer 1B. The blade motor is driven by 16-bit timer 3A which is tied to digital pin 5. The green wire for the control signal of each motor controller will be tied to ground and the yellow wire will be connected to the digital pin with the PWM output of the desired timer. Speed match values and acceleration coefficients can be tuned in the header file for the motor control library.

The output of channel A for the motor encoders are to be tied to digital pins on the Arduino. Pin 8 will accept the left drive motor encoder, pin 9 for the right drive motor encoder and pin 10 for the blade motor encoder. These pulses can be processed after being read by the pulseIn() function on the Arduino.

I.C Receiving Position Data

The GPS module can send WAAS NMEA 0183 data over raw serial to the Arduino RX1 interface. This interface shall be connected to the TX0 output on the GPS module. The sentence we are interested in is the NMEA 0183 $GPRMC message. From this sentence, we are able to extract latitude, longitude, time, and speed information to be processed and passed up to the wireless interface.

The perimeter wire emitter circuit is to be powered off of a 12V power adapter connected to a wall outlet. The 12-gauge perimeter wire is to be run along the entirety of the edge perimeter of the lawn, with both ends connected to the screw terminals on the emitter circuit. The potentiometer on this circuit should be adjusted to achieve a roughly 45 KHz square wave upon the wire.

The perimeter wire detection circuit will be set up so there is one coil on each of the front corners of the mower. The left and right coils should be tied to the Arduino analog pins 8 and 9, respectively. The Arduino analogRead() function can then be used to determine distance relative to the perimeter after passing through a software averager. These values can be processed to determine the distance and approach angle of the mower to the wire.

Three bump sensors lined across the front of the C-channel will be tied to Arduino digital pins 51, 52, and 53. They are active low and will be powered by the 3.3V line on the Arduino.

I.D Module Communication

To add new schedule data to the mower, The user must navigate to the Schedule portion of the app, select the day they wish to add to, and hit the "+" button. Then the user uses the timePicker to select a time and clicks the "Set" button. At this point, a new DateEntity is created and added the the SQLite database. The entity is converted to JSON and sent to the pi REST API. Then, the pi receives the JSON and enters it to the database. Then, it sends the JSON to the Arduino over a simple RS232 serial connection. GPS data will be sent to the Raspberry Pi over the serial connection on the Arduino's RX2 interface. The data will be parsed into JSON format to easily be interpreted by the database.

To receive mower history, the app is refreshed and incoming JSON is converted to DateEntity. The SQLite database is then updated with the new information.

II. Alternative Versions/Designs

Version 1 (3D printed mower): One of the original ideas was to 3D print a lawn mower which would behave and look similar to a Roomba vacuum. The blade would be comprised of a tough string similar to that of a trimmer instead of traditional blades. The main benefit of this design is that it is inexpensive and easy to design given the resources that Iowa State provides. This design also allows the customers to possibly print their own mower chassis which lowers cost for the consumer.

Version 2 (Modify an existing gas mower): Modify a store-bought lawnmower with electronic controllers and parts. The main benefit of this design is that we will not have to create a mower from scratch. There are several downsides to this design: This design would be rather expensive as there are many large parts which are very costly. Another downside is that we would need an extensive background in mechanical design and welding. Safety is also another major concern here, most gas mowers use very large metal blades which in an unfortunate event could lead to injury.

III. Other Considerations

We hit a lot of hurdles in the final leg of this project. Many of our struggles stemmed from our attempt to build the device from scratch. We ran into a lot of time constraints and trouble ordering all of the necessary parts. We are disappointed we were not able to have more to demo and test, but we think we ideated a really solid proof of concept. This was a huge project, but we are proud of the progress that we made with the testing of all of the individual components. We believe we have a great basis for a group to build on next semester.