

Agricultural SWARM Robotic System

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Robotics Engineering

Authors:

Anqi Shen

John Stegeman

Advised By:

Michael Ciaraldi

Nicholas Bertozzi

Abstract

This project aimed to help farmers improve crop production, in order to feed the world with limited resources. With the surge of the Internet of Things devices and ever smaller, more power efficient, cheaper electronic components, farmers can accurately measure microclimate conditions and thus manage their fields more effectively. To solve the socio-economic issues that the farmers encounter, an agricultural swarm robotic system was designed. Our project continued the work of a previous MQP team to develop an Internet of Things SWARM agriculture system that would allow farmers to monitor environmental conditions in their fields with sensor Nodes and manage and maintain the Nodes using autonomous Rovers. The battery-powered Nodes collect temperature, humidity, light levels, and air quality to give insights into the growing conditions in the fields. The Rovers pick up, carry, and place Nodes autonomously so that the farmer does not need to worry about placing and recovering the Nodes manually. With the developed system, farmers would be able to accurately assess conditions in their fields at a low cost.

Executive Summary

This report discussed in detail the need for such agricultural swarm robotics system, the design procedure for the needs, the final outcome, and areas of improvement. Our objective for this project was to provide a low-cost, low-maintenance, user-friendly, and effective solution to monitor microclimate conditions so that farmers would be able to make well-informed decisions on managing the growing environment of their crops.

We researched the socio-economic issues that the farmers face, as well as current technologies and systems that attempt to solve these problems. From the results of the research, we concluded that farmers needed an Internet of Things system that would allow them to closely monitor the environmental conditions but would cost less and be easier to use than the existing solutions. Therefore, with the existing Node-Rover structure from the previous MQP team as a foundation, we explored alternative solutions to the previously mentioned issues.

We proposed alternatives in the Methodology section, where we designed the Nodes, Gateway, Rover, and User Interface system. Sensors, microcontrollers, and other electrical components were selected to perform the data collection and transmission tasks. Node enclosure and Node pickup mechanism solutions were discussed and compared to be able to withstand the environmental conditions on a farm. Software was designed to relay data and commands between the User and the components of the system.

As the result of the project, we successfully produced a system that could accurately measure temperature, humidity, air quality, and light, then present the data on the User Interface graphically, meeting our objective. In addition, we made an autonomous Rover that followed the Users' commands, reducing human labor involved in setting up and maintaining the system. The

cost of the system was analyzed and confirmed to be less than any currently available similar commercial solutions.

Although we were satisfied with accomplishing our objective, there could be improvements to the product that were out of the scope of our project. These potential improvements could be found in Discussion, along with suggestions for mass-production to further cut down the cost.

Table of Contents

Abstract	i
Executive Summary	ii
Table of Contents	iv
List of Figures	vii
List of Tables	viii
Introduction.....	1
1. Background.....	3
1.1 Commercial Needs.....	3
1.2 Existing Solutions	4
1.3 Previous MQP	8
2. Methodology.....	11
2.1 Nodes	11
2.1.1 Environmental Sensing	11
2.1.2 Enclosure.....	13
2.1.3 Radio Network.....	14
2.1.4 Power Consumption.....	15
2.2 Gateway	16
2.3 Rover.....	17
2.3.1 Hardware.....	17
2.3.2 Software	18
2.3.3 Node Placement and Recovery	19
2.4 Software and Web Server	20
2.4.1 Server	21

2.4.2 Database.....	21
2.4.3 Web Interface.....	22
3. Results.....	23
3.1 Nodes and Gateway	23
3.1.1 Node Sensors	24
3.1.2 Radio	26
3.1.3 Enclosure.....	28
3.1.4 Power	32
3.1.5 Cost of Nodes and Gateway.....	33
3.2 Rover.....	34
3.2.1 Lift Mechanism.....	35
3.2.2 Electronics.....	38
3.2.3 GPS	39
3.2.4 Cost of Rover	40
3.3 Server and Web Application.....	41
3.4 Database.....	42
3.5 User Interface.....	44
3.5.1 User Login	44
3.5.2 Node Data Visualization.....	46
3.5.3 Rover Commands.....	49
4. Discussion and Recommendations	54
4.1 Node Circuitry	55
4.2 Rover Drive Base.....	55
4.3 Rover Navigation	56
4.4 User Interface.....	57

5. Citations	58
Appendices.....	60
Appendix A – Smart Agriculture Solutions Comparisons.....	60
Appendix B - Wireless Range Calculations.....	61
Friis Transmission Equation:	61
Node to Gateway Parameters:.....	61
Appendix C – Node Power Consumption.....	62
Appendix D – Node Cost.....	63
Appendix E – Gateway Cost.....	64
Appendix F – Rover Cost	65
Appendix G – MongoDB Setup.....	66
Appendix H - Node Software Setup	67
Appendix I - Gateway Software Setup	68
Appendix J - Rover Software Setup.....	69
Rover Arduino	69
Rover Raspberry Pi.....	69
Appendix K - Server Software Setup.....	70

List of Figures

Figure 1. DigiBale Farm Automation Starter Kit with 3 sensor packages and a gateway.	5
Figure 2 Libelium Smart Agriculture Xtreme sensor package	6
Figure 3. Arable Smart Agricultural Sensor Node.....	8
Figure 4. Previous MQP Sensor Node and Rover	10
Figure 5. Node System Diagram.....	11
Figure 6. Node Enclosure from Bud Industries	13
Figure 7. FRM69HCW Packet Radio – 433MHz.....	15
Figure 8. Feather Ethernet Shield	16
Figure 9. Dagu Wild Thumper 4WD Chassis.....	17
Figure 10. Software Structure Diagram	20
Figure 11. Node Sensor Test.....	25
Figure 12. Node Sensor Response Times	26
Figure 13. Node Outdoors During Testing	28
Figure 14. Waterproof Submersion Testing the Node.....	30
Figure 15. Assembled Node with Acrylic Feet.....	31
Figure 16. Gateway Mounted on the Wall.....	32
Figure 17. Rover Picking up a Node.....	36
Figure 18. Node Lifted ½” off the Ground	37
Figure 19. Lift Engaging the Node’s Handle.....	37
Figure 20. Rover Electronics	39
Figure 21. User Login Screen	45
Figure 22. User Login Screen with Incorrect Username and Password Combination	46
Figure 23. Node Map Screen with 3 Nodes Deployed	47
Figure 24. Node Modal that Allows Editing Node Parameters	48
Figure 25. Node Modal Sensor History Data View	49
Figure 26. Rover Map Scree with 1 Rover Deployed.....	50
Figure 27. Rover Modal Details.....	51
Figure 28. Rover Programing View Selector.....	52
Figure 29. Rover Command View and Popup	53

List of Tables

Table 1. AT Commands Implemented.....	19
---------------------------------------	----

Introduction

Agricultural SWARM robotic project was created to help farmers monitor field conditions and make well-informed decisions to increase production and reduce costs while being environmentally conscious. SWARM stands for Self-assembling Wireless Autonomous Reconfigurable Modules. In our case, the self-assembling network modules are the wireless sensing nodes, the Nodes, in the agricultural fields. All of the Nodes gather environmental data autonomously when activated and transmit it to a web server for storage and display to the farmers. They can be placed and collected easily by the users or by another robot. Therefore, these Nodes can be scattered in areas of fields that farmers would like to collect data from.

This project consists of three parts: a network of Nodes that collect data, a Rover that deploys and retrieves Nodes, and a web server with an interface to manage and view data. Farmers decide what variables that they would like to collect. These specific sensors would be installed into the Nodes before they are deployed. As these sensors collect data from the environment, an onboard low-power microcontroller in the Nodes gather data from the sensors and send data packets back to a gateway via wireless links. All the Nodes in the field would function identically, making a localized environmental condition map of the field. The farmers can then view the data stored on the server and take corresponding actions such as watering specific areas that are dry. To deploy the Nodes to desired areas, the farmers need not go to the field themselves; instead, they can command the Rover to place or retrieve Nodes at specific locations through the interface. The Rover then autonomously drives to a designated location and sets up the Nodes, providing better accuracy at a cheaper cost than using people.

This report explores the needs of the Users in the Background research portion. We explain the design process which we used to arrive at a low-cost, user-friendly, and effective solution for monitoring agricultural environmental data in the Methodology. The Results section analyzes the project for any potential Users wanting to evaluate this agricultural swarm system to determine if it would be suitable for their farms. Individuals interested in further work on this project can find information about recommended improvements in the Discussion session. The Appendix D contains more information on detailed charts, calculations, and instructions on how to set up the system.

1. Background

1.1 Commercial Needs

As a predominant industry in the United States, agriculture is crucial to the daily lives of many Americans. However, this common industry in the U.S. has lost employees to perform basic, yet important tasks. In the state of California, where half of the U.S.'s fruits, vegetables, and nuts are from [Daniels, 2018], a major labor shortage was reported in 2017 [Cfbf.us, 2017]. According to a California Farm Bureau Federation report, 59% of the agricultural labor demanded higher wages, and 50% of the workers were leaving for higher paid jobs, causing the labor shortage [Cfbf.us, 2017]. Such labor shortages require farmers to find substitutes for laborers or raise the amount paid to workers, driving up the prices of their produce. Agricultural Robotics is a viable substitute to help solve labor shortages and provide farmers with more reliable, informed methods of running their farms.

Farmers are looking for solutions to help reduce their workload and make their farms more efficient. The primary driving factor behind the demand for labor is data collection. Farmers need data in order to make the informed decisions used to make their farm more efficient [Post, 2018]. The exact kind of data that each farmer wants depends on what the farm produces; a dairy farm has different requirements than an agricultural strawberry farm. However, primary areas that agricultural crop farmers want insight on can be common: water usage, fertilizer levels in the soil, the weather conditions, and crop growth and height [Aleksandrova, 2018]. By combining all these data points together with artificial intelligence, farmers hope to get actionable insights that give them real-time updates on how their produce is developing, predicted yield levels, and what needs to be done to increase yield [Post, 2018].

Despite the need and want for more sensors in the field to provide data, there are several problems standing in the way of their widespread deployment. The first major problem is connectivity. On a farm, there is very little in terms of an internet connection. A majority of farms only have internet on a few of the main buildings, and even then, it is not the high-speed internet [Seibel, 2017]. This means that traditional wired sensor solutions do not work. Cellular options have similar problems with patchy coverage on farms as well as the fact that cellular radios tend to draw a lot of power for battery operation. Another major problem with deploying sensors is a lack of experience. Farmers are not system designers and wireless experts, so putting in large deployments of sensors and managing them is often beyond their expertise making adoption slower [Seibel, 2017]. The final major problem is maintenance. A single farm can be thousands of acres and could have hundreds to tens of thousands of sensors deployed. This represents a challenge keeping all the sensors running [Aleksandrova, 2018]. A farmer might have to hire people to manage and maintain the sensors, which does not cut down on the labor requirement, it just shifts the type of labor required. Despite these challenges, there are some solutions available.

1.2 Existing Solutions

There are several solutions already available to farmers for smart agriculture sensors. These solutions come from many different types of sources including original equipment manufacturers (OEMs), system integrators, and full stack solution providers.

The first of the categories, OEM are the solutions where a company is just selling sensors that work on someone else's platform. An example of such a device is the Farm Automation Sensor from DigiBale Ltd as shown in Figure 1. This package has a battery sensor, a temperature sensor, and a relay switch for controlling a remote actuator such as a sprinkler valve. Three

sensors plus a gateway sell for \$1000 and the sensors send data to the DigiCloud platform owned by Digi International [Digibale.com, 2018).



Figure 1. DigiBale Farm Automation Starter Kit with 3 sensor packages and a gateway.

[<https://www.digibale.com/shop/products/farm-automation-starter-kit-pre-order/>]

Another example of such a device is the Libelium Smart Agriculture Xtreme Sensor Node shown in Figure 2. The Libelium sensor node allows attachment of a variety of weather station sensors such as humidity, temperature, pressure, wind speed, precipitation, and soil sensors such as soil pH and soil moisture. A single node retails for at least \$3000 and upwards of \$5000 depending on the sensors selected. These solutions are expensive when considering a farmer might put down hundreds of sensors.



Figure 2 Libelium Smart Agriculture Xtreme sensor package

[<http://www.libelium.com/products/plug-sense/models/#smart-agriculture-xtreme>]

The second category of solutions is the integrators. These companies take data from other companies' products, analyze it, and repackage it for farmers. One such company is Digi International. They provide a platform on which device data can be gathered, stored and processed. Another example of such an integrator is Amazon Web Services (AWS). AWS does not make devices but rather works with companies to process and store the data from devices. Often times integrators have access to different resources from the OEMs which makes them better suited to process the data and find actionable items within it. The downside for farmers is that there is one more piece in the technology chain to learn and one more company involved if anything goes wrong, making it harder to get answers because the integrator usually does not understand how the devices work. The integrators do not need a deep understanding, just

enough of an understanding to make the device function and pass data to the other parts of the system [Systems Integrator, 2018].

The final category involves the full stack solution providers. These companies both build the devices as well as create a platform for them to hook into. This style of companies is more common in the agriculture sector based upon our research through numerous search engines. There are numerous companies such as Arable, Semios, WaterBit, and others that provide full solutions for crop monitoring, orchard management, and irrigation control respectively. One such solution from Arable is shown in Figure 3 below. These solutions are nice for farmers as they offer everything in one package from one supplier. If there is ever a problem, they can contact one company to get all the answers. The other benefit is that the systems are traditionally much simpler to operate than others because all the components in it were designed to work together. The downside is that the farmer is largely stuck with a system as most full stack solutions do not interface with external integrators or services.



Figure 3. Arable Smart Agricultural Sensor Node

[<https://www.arable.com/>]

1.3 Previous MQP

The previous MQP team provided the structure of the Agricultural SWARM system; however, their project was unfinished. Through research, they decided to use 4 sensors in the Nodes: a humidity sensor, an air-quality sensor, a temperature sensor, and a light sensor. To determine these sensors, they cross-referenced a project at Carnegie Mellon University by Kantor, George, and Lea Cox and an Egyptian SWARM project by El-kader, Sherine, and El-Basioni [Jefferson et al., 2017]. Each Node contained a Beaglebone Black board with a NodeJS application to do sensor processing and networking. All 4 of the sensors were connected to the Beaglebone through the I2C interface. The Beaglebone used an XBee radio with a Zigbee

protocol stack to create a mesh network and relay collected sensor data to a node with a LAN connection acting as a gateway.

For their data storage and server, the previous MQP team used NodeJS again. The NodeJS server used '.txt' files to store sensor data retrieved from the sensors. The server used a simple API to receive POST data from the gateway Node with the most recent sensor data. This particular architecture caused some scalability problems, as the NodeJS server was unable to keep up with updating the '.txt' files used as a database when the sensors reported faster than once every 30 minutes. The previous architecture also had performance issues reading the stored data in the files as none of the data was indexed, so a full scan was necessary to recover and display any data to the user. The Human-Computer Interface aspect of the web server that displayed the data collected was incomplete. The website was supposed to contain 5 pages: Home, Nodes, Network, Map, and About. At the completion of the previous MQP, it only contained a page showing a temperature heat map and some text with the raw sensor data.

For the Rover, the previous MQP modified a Dagu Wild Thumper 4WD chassis pictured in Figure 4, which they recycled from a previous project to save budget. The original TRex board on the Wild Thumper was used along with a PIXY camera for vision tracking to find the deployed Nodes. The TRex board proved troublesome for the previous team as it has no schematics and over the course of their project developed an untraceable problem where they could not get it to turn on and accept programming. The previous project did not integrate the Rover with the server or mesh network and required a separate connection to the LAN network.

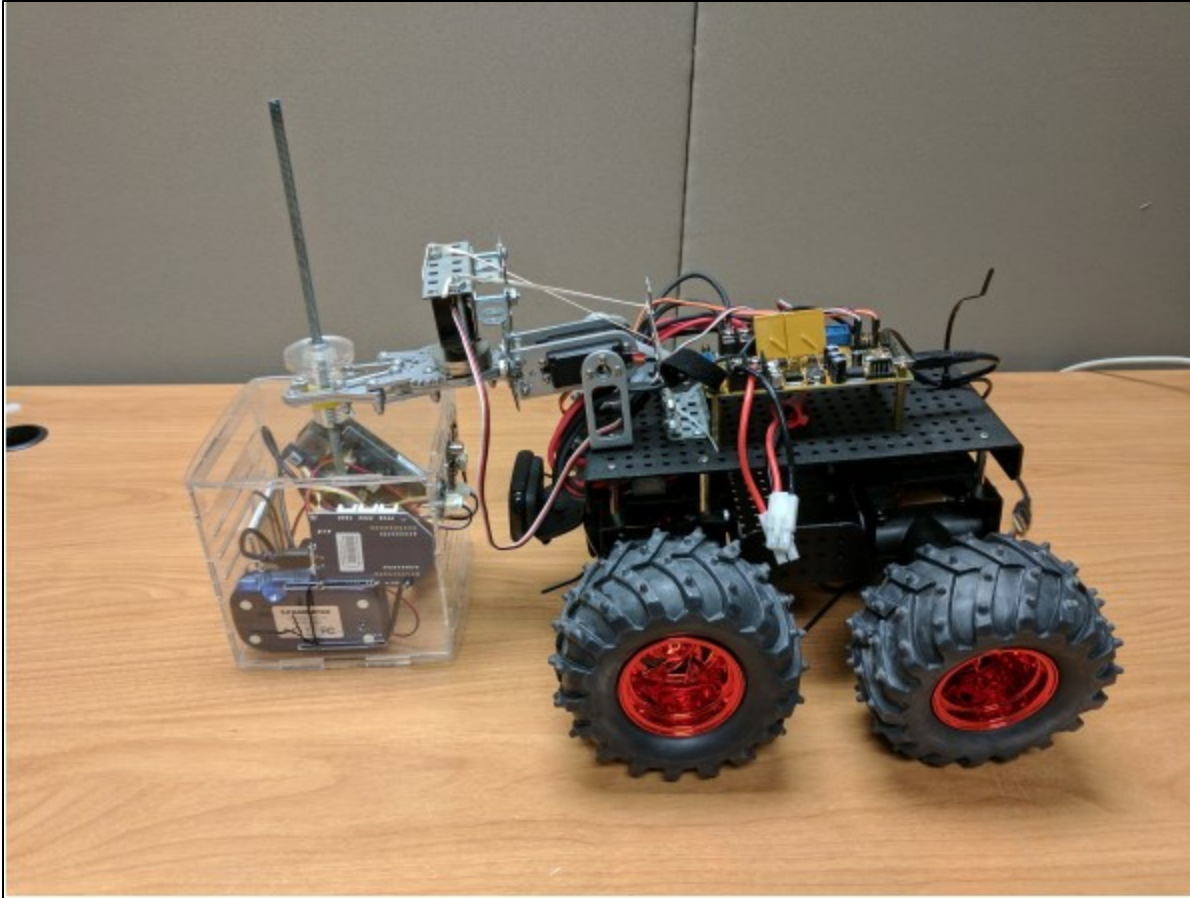


Figure 4. Previous MQP Sensor Node and Rover
[<https://web.wpi.edu/Pubs/E-project/Available/E-project-042717-000935/unrestricted/Agricultural-Swarm-Robotics-Report.pdf>]

2. Methodology

2.1 Nodes

2.1.1 Environmental Sensing

Four environmental sensors were placed in the Nodes: a humidity sensor, an air-quality sensor, a temperature sensor, and a light sensor. The node microcontroller took samples from each of these sensors through an I2C interface or analog input and then stored these values to send to the server at a later time. Most of these sensors were obtained from the previous team, with the exceptions of one air-quality sensor and one I2C hub. The Node system diagram can be seen below in Figure 5.

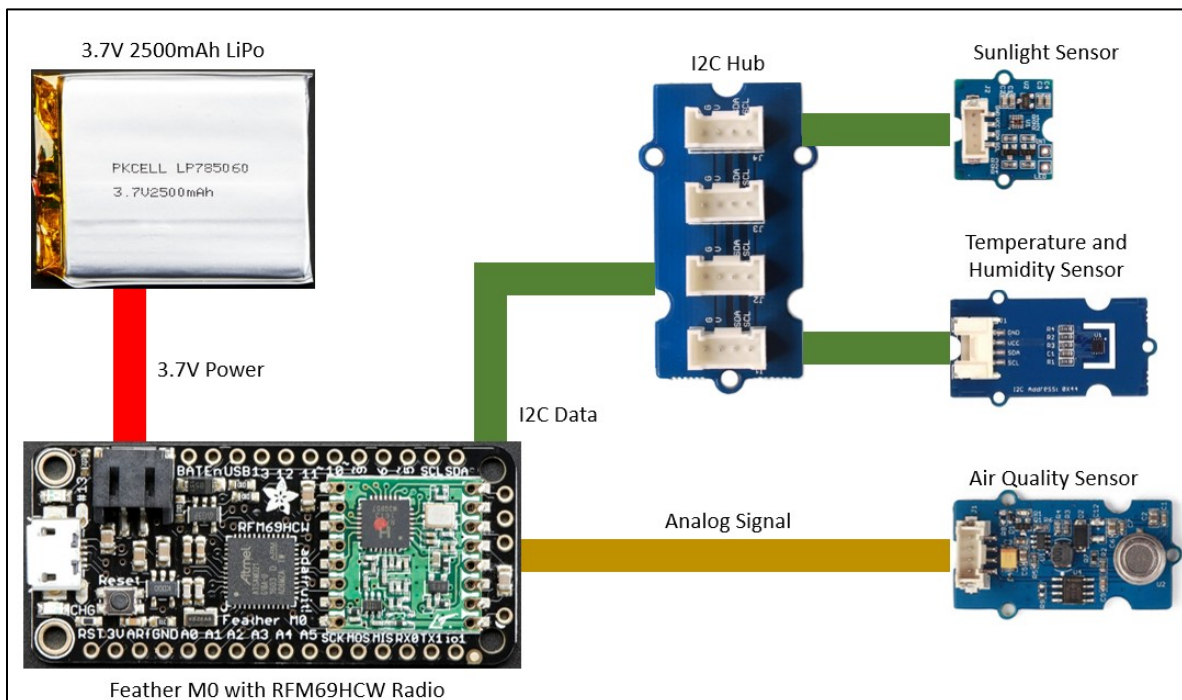


Figure 5. Node System Diagram

The humidity sensor and temperature sensor were grouped together on the Grove Temperature & Humidity Sensor breakout board with a SHT31 sensor IC. According to the SHT31 datasheets, the SHT31 sensor had a $\pm 2\%$ error on humidity over 5%-100% relative humidity and a temperature sensor with a range from $-40\text{ }^{\circ}\text{C}$ to $125\text{ }^{\circ}\text{C}$, and an accuracy of $\pm 0.3\text{ }^{\circ}\text{C}$ [Seedstudio.com-Temperature, 2017]. For air-quality detection, a Grove Air Quality sensor v1.3 was used. This sensor was suitable for monitoring general air quality indicator gases, such as carbon monoxide, alcohol, acetone, formaldehyde, ozone, and other slightly toxic gases [Seedstudio.com-Air, 2017]. It worked in a low power mode, just barely keeping power on the electrochemical gas cell to keep precision, saving energy in the battery. The Grove Sunlight Sensor from the same Grove family was used for its multi-channel light sensing capability. This module could detect UV light, visible light, and infrared light through its multi-channel interface. [Seedstudio.com-Sunlight, 2017].

To get data back from the sensors, the I2C Temperature and Humidity sensor board as well as the Sunlight sensor would be connected via an I2C hub to the microcontroller. The Air Quality sensor reported an analog signal proportional to the concentration of harmful gases in the air, and so it would be connected directly to the microcontroller via a pin with ADC capabilities.

2.1.2 Enclosure



Figure 6. Node Enclosure from Bud Industries

[<https://www.digikey.com/product-detail/en/bud-industries/PN-1323-CMB/377-1890-ND/2674154>]

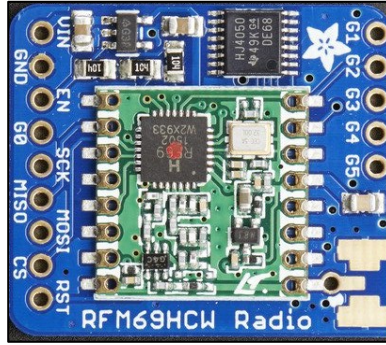
The sensors chosen dictated the criteria needed for the Node enclosure. The electronics needed to be sealed in a weatherproof box [Digikey.com-Box, 2018], shown above in Figure 6, protected from the outdoor conditions in the field while allowing light and gases to filter in and out of the box. The box also must allow radio signals to travel back to the gateway. To allow the sunlight sensor to work properly, the top of the Node box needed to be transparent. The humidity and temperature sensors, as well as the air-quality sensors, relied on air samples. To obtain real-time air samples, a special breathable PTFE Gore vent [Digikey.com-Vent, 2018] was used which allows air exchange between the box and the environment without letting any liquids through. The Radio antenna protruded through the top of the clear top to enable radio connection with the Nodes, and was sealed by a rubber O ring on the outside of the box to make it weathertight.

For simple deploy and retrieve actions by a Rover, additional structures were added to the main electronic enclosure. To secure the Node in the field, four zinc-coated 12-in long rods were inserted into holes on the lips of the box in the four corners. Farmers could adjust the depth of

the rods that screw into the soil based on their specific conditions. On the top of the rods, above the clear top of the box, two 1-in diameter, 6-in long PVC pipes were mounted as handles, one on each end of the enclosure, for the Rover to carry the Node. The sections of PVC pipe were painted neon pink to allow object and color recognition via the camera on the Rover.

2.1.3 Radio Network

This project continued to use a radio mesh network to connect the Nodes. As the previous team had designed it, multiple Nodes were to be connected to one server through paths that may require relays of other Nodes that are closer to the server. The past report mentioned that the team had trouble with the Zigbees' constant resets of the individual Nodes' identity. This problem caused the system to be unable to recognize the Nodes in the software, making reliable communication impossible. Thus, instead of using the Zigbee mesh network kit inherited from the previous team, we decided to use the Adafruit Feather M0 RFM69HCW Packet Radio - 433MHz (Feather M0). Although the RFM69HCW radio cannot transmit audio or video, it is perfect for transmissions of small data packets like the environmental sensor data with much lower power consumption than the Zigbee radio. The RFM69HCW radio module is also compatible with the existing Arduino libraries, making the setup of the programming less time-consuming.



*Figure 7. FRM69HCW Packet Radio – 433MHz
[<https://www.adafruit.com/product/3071>]*

The Adafruit Feather M0 RFM69HCW Packet Radio - 433MHz board pictured above in Figure 7 is a better alternative to the Zigbee and Beaglebone Black combination. Feather M0 has an ATSAM D21G18 ARM Cortex M0 processor, with 48MHz clock frequency, 3.3V logic, and hardware I2C capability [Industries-Feather M0, 2018], satisfying all the criteria for the Node microcontroller. Additionally, the processor has 32K of RAM and 256K of Flash memory, plenty for sensor nodes and a 0.0099uW active power, compared to the excessive 512MB RAM and 4G Flash memory and 2W active power requirement on the Beaglebone Black [Beagleboard.org, 2018]. The power consumption reduction would allow us to use a smaller battery for the Node while still promising much better battery life as the Feather M0 + RFM69HCW uses only 0.5% as much power as the Beaglebone+Zigbee design. The Feather M0 would be a more robust, more power efficient, and simpler controller solution for the radio on the Nodes.

2.1.4 Power Consumption

For the nodes to be practical, they need to last at least one year before needing their batteries replaced. At the same time, the Node should report environmental data at least once per hour so that a fairly real-time model of the field conditions could be made. To fit within the size constraints of the Node enclosure, we have a maximum battery capacity of around 3000mAh. On

hand however, we inherited 2500mAh LiPo batteries from the previous team, so we used those as our starting point. To reach an operational lifetime of at least one year, we assumed the usable capacity of the 2500mAh battery to be only about 1800mAh to account for any drainage from temperature variations and to account for self-discharge of the LiPo as well. If we were using other battery chemistries, the self-discharge rate would be lower, but the capacity per cubic unit of size would be lower. For the 1800mAh to last for a full year the Node could drain a maximum of 4.93mAh per day, although lower power consumption would be even better.

2.2 Gateway

The gateway is a relatively simple component of the system. Its job was to listen to the radio network and forward any sensor data packets from Nodes to the server and send commands from the server across the mesh network to the Rover. The hardware for the Gateway builds off of the same Feather M0 with the RFM69HCW radio which allows it to listen and send data on the mesh network. In addition, the gateway also has an Ethernet shield, shown in Figure 8, which connects the microcontroller to an Ethernet connection from which the gateway can contact the server to send and receive data. To keep the system simple, we housed the gateway electronics in the same waterproof enclosure that the Nodes use.

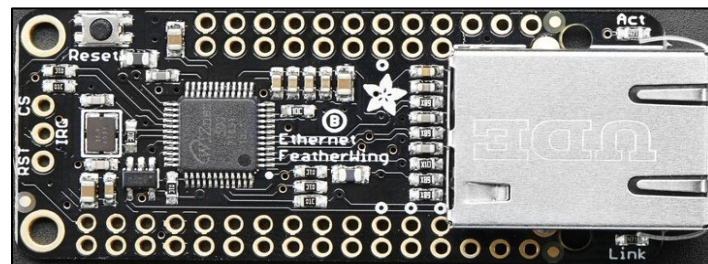


Figure 8. Feather Ethernet Shield

[<https://www.adafruit.com/product/3201>]

2.3 Rover

2.3.1 Hardware

Since we inherited the Wild Thumper drive base, pictured in Figure 9, from the previous team, the Rover hardware mainly consists of the intake for Nodes as well as a holder for the electronic components. Material wise, we inherited a continuous-rotation servo and plywood for construction. We also purchased two HiTec HB-485 servos with a stall torque at 6V of 83.3 oz-in.



Figure 9. Dagu Wild Thumper 4WD Chassis

[<https://www.pololu.com/product/1566>]

The intake of the Rover must be able to lift up a Node from the ground, hold it in a position a few inches off the ground while driving around, and place it down to the ground safely. We explored many mechanisms when we designed the lift. First, we considered a linear system that would allow the Node to travel vertically. Given one continuous rotation servo, we gave up on the idea of building a scissor lift or a linear slide. Those two mechanisms would

surely fulfill the requirements, but they would also demand more power and tuning. With plywood as our main construction material for lower cost and manufacturing time, we were not confident that we would be able to craft a scissor lift or a linear slide that would work efficiently. Another option for linear mechanism was a pulley system. A hook would be attached to cables that run vertically on a surface in the front of the Rover; the servo would drive two connected drums to operate the directions of the hook. With this plan, we would still run into the issue of building a sufficient system by hand with plywood. Although it is more doable than the scissor lift and linear slide, it can also run into the problem of being too front heavy and tip over the Rover.

We also looked into possibilities of building a four-bar lift. We knew that the plywood would work fine with a four-bar mechanism since we have had previous experience with it in RBE 2001. Another advantage of using a four-bar mechanism was that it was easy to modify the link lengths slightly to alter the amount of stroke vs the lifting force of the mechanism. We also knew that the HS-485 servos would be a good fit for a four-bar linkage since they could provide 180 degrees of rotation to the crank linkage with a decent amount of torque to lift the nodes. In the end we decided to implement a four-bar for the node lifting mechanism.

2.3.2 Software

The Rover software consisted of two parts: the communication link between the Feather M0 board and the Raspberry Pi (RPi) and autonomous functionality on the RPi.

The com-link between Feather M0 (in Arduino) and the RPi (in Python) was via a Serial USB cable plugged into both devices. The Feather M0 was in a board stack with a GPS, a servo controller that manages the lift of the Rover, and a radio transceiver on board. The Feather M0 would send a packet to the Rover with the current location, and current command data fetched

from the server to the RPi. The RPi would send packets commanding a change of lift servo position, a forced update of the current command from the server (to insure nothing changed about the current waypoint), and a command to fetch the next command from the server when the current one was complete. These data packets, detailed below in Table 1, would follow the AT protocol. An AT protocol is formatted as ‘ATXX[=...]’, where ‘XX’ would be a two-letter code which indicates the type of command and the =... is an optional addition for specifying additional data to pass with the command.

Table 1. AT Commands Implemented

Command	Sent by	Description
ATUD\r\n	RPi	Check the server for the latest command to follow
ATAV\r\n	RPi	Mark the current command as finished on the server and fetch the next one
ATLM=<Lift_Position>\r\n	RPi	Set the lift position of the Rover. <0-255>
ATGT=<target_latitude>,<target_longitude>,<bearing_to_target>,<latitude>,<longitude>,<current_bearing>,<distance_to_target>,<bearing_error_to_target>,<target_cmd_action>\r\n	Feather M0	Sends the navigational data to the RPi on the Rover once every 100ms. <target_cmd_action> is from 0-15. 0 is drive, 1 is place node, 2 is pickup node, 3-15 is reserved for future use. <lat>,<lng>,<degrees>,<lat>,<lng>,<degrees>,<meters>,<degrees>,<integer>

2.3.3 Node Placement and Recovery

The Rover use a four-bar mechanism as discussed above to pick and place the Nodes in the field. Placing the node is a simple task; the Rover drives to the GPS location specified by the farmer and places the Node at that location. Picking up the Node however is much more complex as the GPS accuracy is not good enough to position in exactly the same location as

when the Rover placed the Node; the best GPS accuracy is about 1m of accuracy. To combat this, the Rover must align itself on the Node to pick it up once it is close. We used a camera on the Rover connected to the RPi to do color recognition and object tracking to target the Node once the Rover is within 3m of the recorded Node location. Using the camera, once the Node is lined up with, the Rover picked it up using the same four-bar mechanism and carry the Node back to the farmer.

2.4 Software and Web Server

The overall diagram of the software system can be seen below. As mentioned in previous sections all communications with the Rover and Nodes passes through the Gateway which has a connection to the Server. The server maintains connections to the gateways, the databases, and the user facing web applications and coordinates all their actions together as shown in Figure 10.

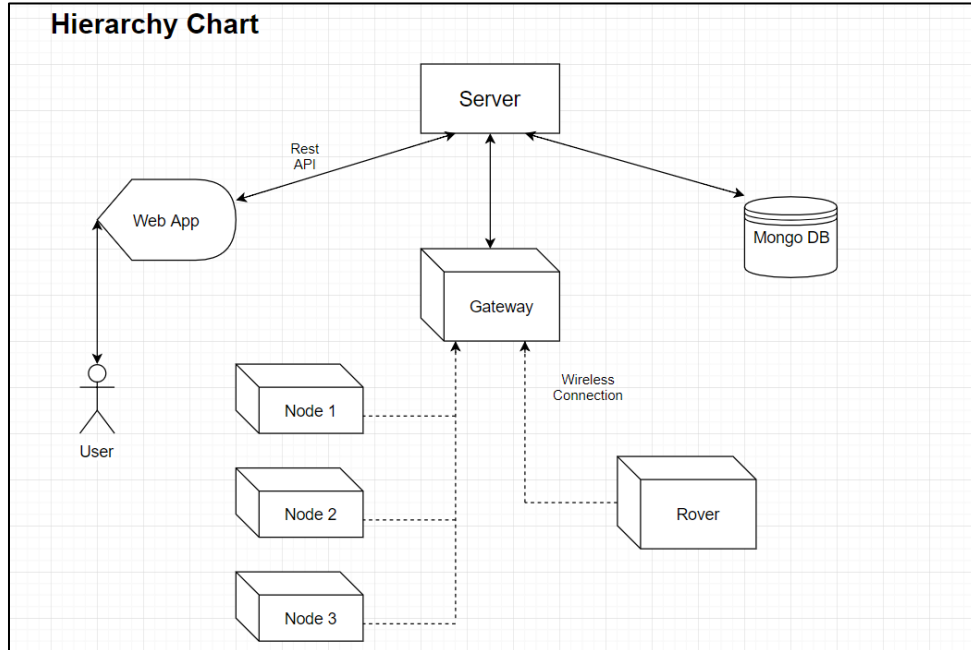


Figure 10. Software Structure Diagram

2.4.1 Server

The web server was a JavaEE web server. This kind of server allows web applications to be deployed as a WAR (Web ARchive) file that contains both application code and HTML, CSS, and JavaScript content. The web server served the HTML, CSS and JavaScript on one URL path, and the Application Programming Interface, (API), which allows the web pages and the Gateway to talk with the application logic, was served from a different URL path. The web application was developed in Java which is a commonly used server programming language.

The API for the server provided functions for the user to login, create an account, and logout, to retrieve sensor data for time periods, to find the Node's GPS location and edit its nickname, and for a Node to post sensor data. Additionally, the API provided functions for the user to create Rover commands, to view the current list of commands for a Rover, to create a Rover, and for the Rover to retrieve the current command to execute.

2.4.2 Database

The logic in the web application relies on the data being persisted somewhere, and that somewhere is a database. In the previous year of the MQP, the team used directly accessed files to implement a custom database. However, performance suffered when much data was added. For the continuation of the project we used MongoDB for the database. Using a commercial database allows us to focus on the application rather than worrying about the complex code needed to implement a highly functioning database that performs well under load. MongoDB is a document-based NoSQL database which allows unstructured data. This database was chosen because it is highly flexible allowing rapid schema changes to the data as well as multiple indexes to be built on the documents allowing fast, expressive queries to be performed on the data. MongoDB also has well-documented drivers available for Java which makes interfacing

with the database easier and faster. The database was hosted on the same computer as the local web server for this project, although nothing prevents running the processes on separate machines when higher load capacities are needed.

2.4.3 Web Interface

As mentioned in 1.2, we planned on rewriting the Human Interface of this project in the form of a local web server while borrowing some of the previous team's idea on its structure. Since most people are used to the form of a website, we decided to use it as the means to display any data from the Nodes. The farmers would have clear visual demonstrations of the environmental conditions so that they could make executive decisions. The Human Interface would simplify the tasks and minimize manual labor to operate the SWARM system thus increasing its efficiency for the farmer.

To study the change of conditions in one isolated area, the farmer would read time-stamped data packets from a single node within a certain window of time. There would be a graph of the collected environmental variables within a desired time window. If the farmer would like to look at all the fields as a whole and troubleshoot the areas that have worse environmental conditions, they may explore the Map tab, where the server would take the data point and its GPS location to reflect the field on a geographic map similar to Google Maps.

Not only should the farmers read data from the Human Interface, but they can also take actions through it via the Rover. To issue a command to the Rover the farmer would simply click on a location on the map to place a waypoint. Once a waypoint was created the farmer could change the desired action at that waypoint, (drive, pickup Node, or place Node), or add additional waypoints by simply clicking on the map again. In this way the farmer could program the Rover with very simple actions and an intuitive workflow.

3. Results

As stated in the methodology section, there were two hardware components of this project: the Nodes and the Rover; the software components included the server, the User interface, the Node software, and the Rover software. All parts were completed successfully during the course of the 15-week project.

For demonstration purposes, three Nodes were created along with one Gateway. To cover the entire farm, the farmer would install a Gateway on the outside wall of the facility that would be connected to an Ethernet cable. The Nodes would then be placed at User-designated positions autonomously by the Rover. With the current sensor setup, the Nodes could record UV light, IR light, visible light, humidity, temperature, and air quality data and send the data to the server periodically. The farmer could monitor all the data sent back by the Nodes through the User interface in the forms of graphs and data points shown on a web interface. To operate the autonomous Rover, the User would simply click on the map to place waypoints and select the command to perform at each waypoint such as ‘drop off’, ‘pick up’, and ‘drive to’. This SWARM system would allow farmers to collect environmental data from their field without having to hire additional staff or even step outside of the facility to manage the data collection and system maintenance. A comparison of the system we developed versus the existing solutions can be found in Appendix A.

3.1 Nodes and Gateway

There were two types of electronics packages in the project: the Nodes and the Gateway. Nodes were designed to be distributed in the fields to collect environmental data, and the single

Gateway on the wall of the facility would receive data from the Nodes and upload it through the Ethernet to the server.

3.1.1 Node Sensors

The first step we performed when inheriting the sensors from the previous MQP team was to test them. We started by downloading the sensor libraries from SeeedStudio, the sensor board manufacturer, and running them. We ran into problems with the Air Quality sensor because the original library for it was built for the ADC peripherals on AVR microcontrollers. Our Feather M0 was a Cortex microcontroller, so the ADC peripherals and the access methods were different. After fixing the relevant code to allow the library to run on the Cortex M0 controller we were able to get results from all the sensors. We did some basic testing to see how quickly the sensors would respond to external stimuli and found that the sensors were accurate with good response times as shown below printed out using the Arduino serial plotter as shown in Figure 11.



Figure 11. Node Sensor Test

Once the Node sensors were installed into the Node enclosure with the PTFE semipermeable membrane, we needed to make sure they still worked and responded quickly to external changes to temperature and humidity. The figure below shows one of our tests. For the test shown below we let the sensors warm up inside at room temperature which is the flatter section of the green humidity and orange temperature lines on the left side. After we were satisfied that the numbers were accurate, for this test humidity and temperature were within +/- 5% of relative humidity and +/-1°C of actual values as reported by local weather stations within a mile on the Weather Underground (www.wunderground.com), we moved the Node outside into cold, rainy weather. We watched the plot shown below in Figure 12 to see how long it took for the sensor values to stabilize. It took roughly 30 minutes for the temperature and humidity to change to the actual values outdoors. After the values stabilized outdoors, we brought the Node back inside. This time it took almost an hour for the sensor values to report the indoor values

again. From looking at the plot, the temperature and humidity reach the final point at the same time which means that the sensors do not have time delay, and instead the delay of change is determined by the rate of diffusion across the PTFE membrane. The outdoor->indoor rate of change was about twice as slow as the indoor->outdoor change due to air movement. Since it was raining outdoors with a little wind, that moved more air past the membrane and agitated the air near the membrane allowing faster diffusion. Indoors, there was less air movement with did not allow the PTFE membrane to diffuse the indoor air into the enclosure as fast.

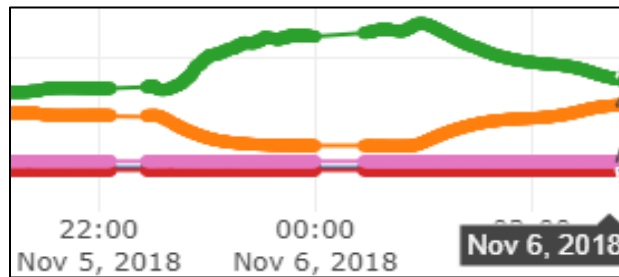


Figure 12. Node Sensor Response Times

Besides the test mentioned above, we ran a few other similar tests to verify the results. The results of our other tests showed similar results. In all our tests the sensors reported humidity accuracies of $\pm 5\%$ relative humidity and temperature accuracies of $\pm 1^{\circ}\text{C}$. This was deemed appropriate for environmental data as super high sensing tolerances below 2% relative humidity and temperature accuracies of $\pm 0.5^{\circ}\text{C}$ without calibration steps and would require much more expensive sensors.

3.1.2 Radio

For the radio, we used the RFM69HCW packet radio that came with the Feather M0 boards. We used the default examples for the radio from Adafruit as the starting point for using

the radios. The libraries provided for the radio worked well without issues. The radios themselves did not have ids built in, so all data packets contained an 8-byte, 64-bit id header that identified what radio should listen to the packet. This filtering for the 64-bit address was done on the device, not the radio, and worked well. In all our testing, we never had a data mix up or a packet being handled by the wrong device.

To ensure that the radios in the Nodes would have adequate range in the field, we performed range testing. We did not have any farms nearby, so we performed testing in an urban setting to get a baseline. We determined that the maximum distance between two Nodes in an urban environment was 0.32km, with the receiver 1m off the ground. After scaling the urban line of sight value based on relativized testing data, the maximum expected line of sight distance would be 1.6km +/- 0.3km on farms. To do a sanity check, we compared our expected range against a calculated value based on the Friis transmission range equation shown below. For reliable reception of data packets, we needed greater than -89dBm of receive power. Based on the equation we got 1.8km of range which matched well with our expected range. For the full range calculations, see Appendix B.

$$P_r = \frac{P_t * G_t * G_r * \lambda}{(4\pi R)^2}$$

P_r = Power at receiver
P_t = Power of transmitter
G_r = Receiving antenna gain
G_t = Transmitting antenna gain
λ = Wavelength in meters
R = Distance between antennas

3.1.3 Enclosure

The Node's enclosure contained two parts: a waterproof box that housed all the electronic components and handles that allowed the Rover to transport the Node. A final Node is shown deployed in Figure 13.



Figure 13. Node Outdoors During Testing

The waterproof box was bought off-the-shelf from Digikey as mentioned in Methodology. It was 4.528" L by 3.543" W by 2.165" H, with a solid light grey polycarbonate body and a clear lid. Along the inside walls of the body, there were ribs that allow insertion of thin dividers. To organize the sensors, the microcontroller, and the battery, we laser cut 3.5" W by 2" H acrylic sheets to make the dividers, where we mounted all the electrical components. We used 3M's 0.2" thick clear acrylic indoor tape to glue the sensors and microcontroller to the board. The tape was used instead of screws because the soldering joints that protruded the surface of the board were fragile to pressure exerted by screws; the tape solved the problem by

providing a cushion for the board so that the soldering joints could be pressed into the thickness of the tape, when the board sat on top of the surface of the tape. The clear lid of the box would let in natural lighting for the light sensor to work properly. The breathable PTFE semipermeable membrane from Gore enabled air from outside the box to slowly permeate into the box without letting water into the box.

When the box was screwed closed with the waterproof seal in place, the PTFE membrane installed, and the antenna in place, just like the Node would be deployed in the field, we performed a submerge-test to ensure that the enclosure was completely watertight, so the moisture would not damage the electronics inside. We placed the enclosure under a standard kitchen faucet in a large pot pictured in Figure 14; water from the faucet was turned to half capacity to mimic pouring rain. Then the enclosure was submerged in water in the bowl for 10 minutes with the top of the lid 1.5” below the surface of the water. We opened the box after 10 minutes to check on the leakage by swabbing the internals of the box with a tissue. With the tissue remaining dry, it proved that the enclosure was watertight.



Figure 14. Waterproof Submersion Testing the Node

After we completed the housing for the electronics, we added hardware so that the Node could be easily picked up, carried, and dropped off by the Rover. Four $\frac{3}{8}$ " diameter zinc-coated 12" rods were attached to the four corners of the box, with two 6" sections of 1.25" diameter PVC pipes across the short sides of the box on the top of these rods as handles. The PVC pipes were spray-painted neon pink to aid in computer vision identification for the rover. Originally, we planned to secure the Node in the field with the rods as mentioned in Methodology; these rods would protrude from the bottom to act as feet that would be inserted into the soil by the Rover. However, when we tested this prototype setup on the grass by hand with $\frac{3}{4}$ " feet, we found out that they were extremely difficult to push into the soil even by hand and did not prevent tipping too well. Therefore, we changed the strategy for preventing the Node from tipping over in the field; instead of using the vertical feet, we laser cut acrylic sheets to make

horizontal feet that enlarged the footprint of the Node as seen in the following photo, Figure 15. This new Node structure demanded less force from the Rover's lift to push downward, and it standardized the height of the handles, making the pick-up easier for the Rover.

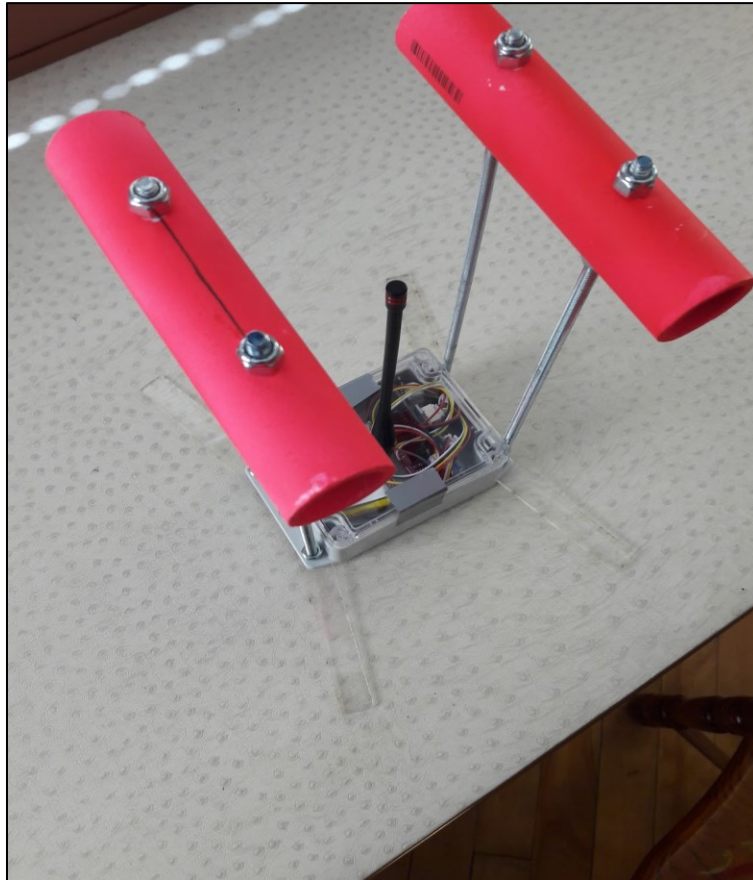


Figure 15. Assembled Node with Acrylic Feet

The Gateway's enclosure was different from the Node's because it would not interact with the Rover physically. The Gateway instead would be mounted on a wall as demonstrated in Figure 16 and fed an Ethernet cable with data and Power Over Ethernet (POE). Like the Nodes in the field, the Gateway radio antenna was mounted vertically so that the antennas would be able to talk to each other. Since the Gateway did not contain any environmental sensors, the air

vent was unnecessary. Instead, a waterproof Ethernet cable gland was threaded through the box to provide power and Ethernet connection to the Gateway.



Figure 16. Gateway Mounted on the Wall

3.1.4 Power

The Gateway's power was provided by the Ethernet cable via a passive Power Over Ethernet (POE) setup, but the Nodes in the field needed to be battery powered. We inherited 2 types of batteries from the previous team: 2500mA LiPo batteries that could be directly connected to the microcontroller, and higher capacity 3100mAh LiFe battery cells that required a separate holder. We started with the LiPo cells as our starting point because they could plug

directly into our Node control board. We knew that we had limited battery capacity, so we used power saving modes on the radio and microcontroller during the periods when the microcontroller was not required to transmit data. During testing, the data packets were transmitted every minute, with the microcontroller at low-power mode in between. However, when deployed in a field, the period could be increased to one hour between transmits. We performed current measuring tests with the lab bench oscilloscopes, but we were unable to obtain reasonable sleep power usage data because the currents that the Node used during sleep were in the single digit micro-amp range and were too small to be accurately measured by the lab equipment. Therefore, we calculated power consumption of each component in the Node with their datasheets and used the calculation to determine which battery solution we would adopt as our final solution. As shown in Appendix C

Based on these results, the 2500mA LiPo battery would be able to operate the Node for 10 year with data transmission every 15 minutes and normal self-discharge rates for the LiPo battery. We chose to use the 2500mA because they were overall cheaper, easier to assemble, and powerful enough for our purposes. This also meant that within the first 10 years, the Users would not need to take apart Nodes to recharge the batteries.

3.1.5 Cost of Nodes and Gateway

The Nodes we built would cost \$101.97 per unit as we built them, see Appendix D for breakdown. This price was inflated because we did not buy sensor ICs, but rather mounted sensor ICs and cables. Should the Node be built in a production setting, with all the components mounted to a single custom printed circuit board, the cost would be expected to be well less than half of the current cost, which would put the hardware cost around \$50 before the benefits of buying components in bulk.

The cost of a single Gateway was \$91.15 for which the breakdown could be found in Appendix E. The only components not included in that cost would be a long Ethernet cable that the farmer would run themselves. This cost would be much closer to a final production cost, as it did not contain mounted sensors that had high markups. The expected final cost for a production gateway with custom PCBs would be around \$80.

The total cost of our 3 Nodes + Gateway as we built them was \$397.06 which is a huge cost savings over the existing solutions such as the DigiBale solution for 3 Sensor Units and 1 Gateway for \$1000. At expected quantity manufacturing costs, our solution would end up around the \$230 mark for the hardware of 3 Nodes and a Gateway; at a 50% margin, the system could be sold for \$500, which was still half of the DigiBale solution while maintaining more functionality. This could be a large cost savings for farmers who would want smart environmental sensors for their field. A detailed comparison of our system cost and features versus existing ones can be found in Appendix A.

3.2 Rover

The Rover was the only mobile component of this project. It carried out User's commands and transported the Nodes. For the convenience of the User, the Rover was fully autonomous and was controlled by being given GPS waypoints and action commands such as 'pick up', 'drive to', and 'drop off'. While the Nodes and Gateway were made weatherproof, the Rover was only able to be operated in times without precipitation on relatively dry ground, as we did not spend time or effort trying to waterproof the Rover. The 4WD base we inherited was not waterproof, so we did not make the rest of the system waterproof either.

The Rover was able to perform all the tasks it needed to: picking up, carrying, and dropping off one Node; recognizing a Node using its onboard camera; and communicating with the server through its radio and the gateway to retrieve new commands, update its status for the User to track, and follow the commands to manipulate the Nodes. The Rover worked decently, though the suspension caused some issues. The suspension for the Wild Thumper chassis had springs that forced the wheels down so that it could go over rough terrain. Unfortunately, this meant that during point turns, where one side of the drive went forwards and the other side went backwards, the front corner wheel would tilt inwards and cause the turning resistance to increase drastically which decreased the turning accuracy and the minimum amount of power needed to turn. We were able to overcome this issue in software by applying more power while turning and avoiding point turns as much as possible.

3.2.1 Lift Mechanism

The lift mechanism on the Rover was a four-bar made in birch plywood driven by two servos. The full mechanism can be seen in action in Figure 17.

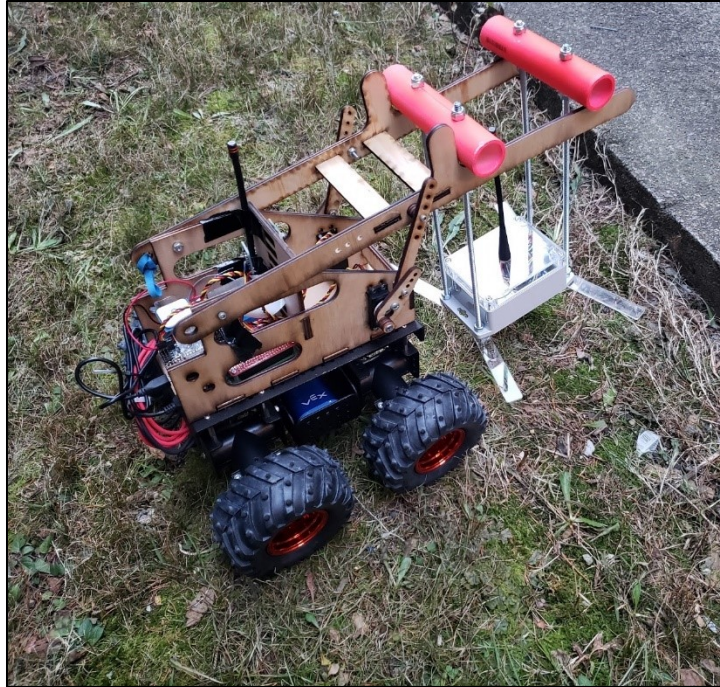


Figure 17. Rover Picking up a Node

We used 0.2" birch plywood for the lift because it was accessible and easy to rapidly prototype. The lift was first designed in CAD, then laser cut. The two HS 485 servos were attached to the crank and provided enough force to lift up a Node 0.5" off the ground as can be seen in Figure 18.



Figure 18. Node Lifted 1/2" off the Ground

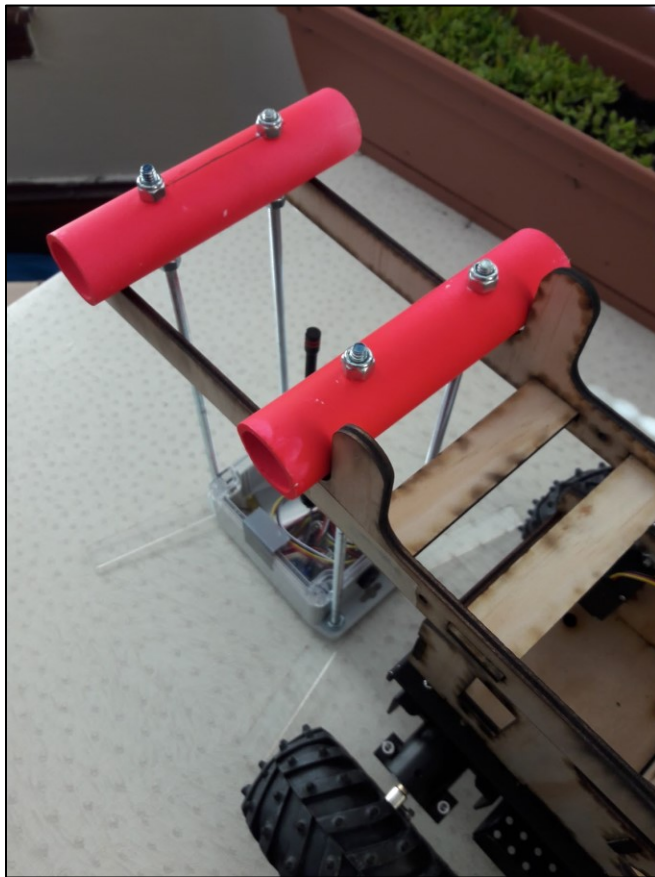


Figure 19. Lift Engaging the Node's Handle

Braces were added between the two sides of the lift to ensure stability. The driven link acted as the arm that held the Node's handle. The L shaped part in the front of the driven link was used to hold the handle on the Node when the arms were lifted. The L shape helped correct the position of the Node when it was being picked up and stopped it from sliding down while the Rover was driving as can be seen in Figure 19. When the Rover was carrying the Node, the four-bar linkage was designed so that the crank link on the servo would be collinear with the linkage connecting it to the lift arms. This setup enabled the servos to exert almost no torque at all while carrying the Node, reducing servo wear. This design also allowed us to preserve more battery power because the servos did not need to produce constant torque.

3.2.2 Electronics

The Rover electronics contained two parts: the Raspberry Pi (RPi) cortex and the Feather M0 microcontroller stack. The reason why there were two systems on the Rover was that we wanted to take advantage of the processing power of the RPi but needed an Arduino interface for the radio to allow the Rover to talk to the Gateway. The Feather M0 board was used for the Arduino environment; it controlled the radio module that received command and sent out GPS data, the servo controller that drove the lift, the GPS module that pinpointed the Rover's location, and the compass that detected the heading. The RPi directly controlled the robot's drive base and the camera for visual recognition. The complete electronics system can be seen in Figure 20.

The servos that were used to drive the lift were HS 485 180-degree rotation servos. The drive motors were included in the Wild Thumper drive base and contained individual motor controllers. For power supply, the Rover used one 3000mA Vex 7.2V battery for all the electronic components including motors, servos, and control boards.

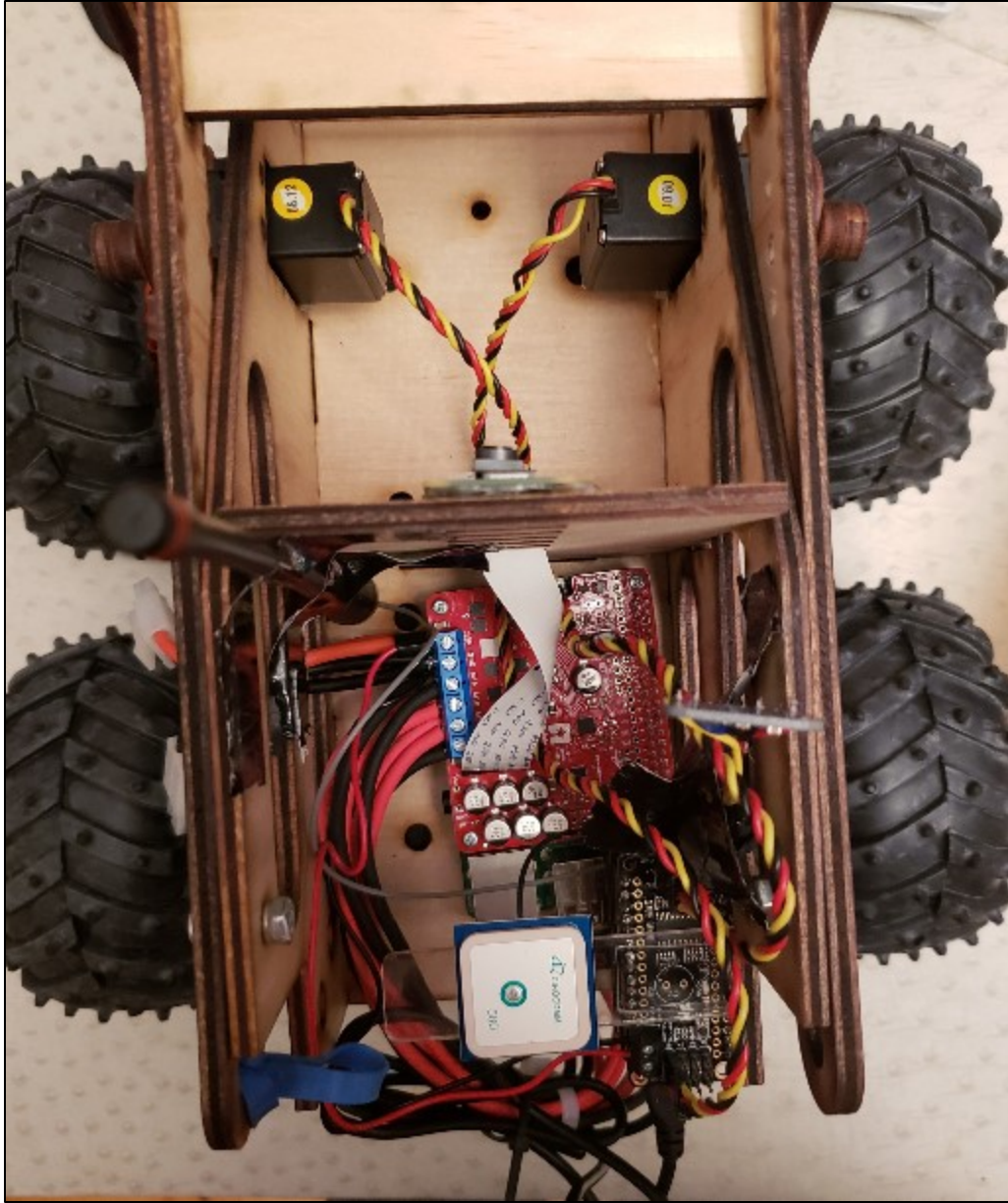


Figure 20. Rover Electronics

3.2.3 GPS

Originally, we used the Adafruit GPS wing for the Feather M0 as our main GPS sensor, but it was unable to perform reliably. The GPS receiver returned the wrong position and caused the Rover to appear like it was jumping around several hundred feet every few seconds. To

improve the localization ability of the GPS, we bought a 32db High Gain CIROCOMM active antenna for better GPS connection and a separate LSM303 compass board from Adafruit to give accurate instantaneous heading data so that the Rover could follow a heading even with temporary GPS errors. However, because we could only test the Rover in the city of Worcester, which was a busy urban area, the GPS readings were still not sufficient to pinpoint of the location of the Rover. With the active antenna, the Rover location still jumped around about 100 feet and occasionally would have constant offsets so the reported position was 50 feet from where the Rover actually was. The accuracy of the GPS receiver was sub 8.2 feet, so our problems stemmed from poorly received signals. If we were to purchase a better GPS module, we would have to buy a differential GPS unit to get the accuracy to sub 1' with increased receive sensitivity, which would set us back at least \$200. Without a guarantee that the differential GPS unit would give us accurate reading, we decided that we would work with our current GPS.

3.2.4 Cost of Rover

The total cost of the Rover was \$500.77, as seen in Appendix F. However, the cost here was not indicative of what a Rover in production would cost. The cost to make the Rover more robust and waterproof, as would be needed if it were to be sold to farmers, would likely be in the couple hundred to thousands of dollars as waterproofing actuators could be expensive.

Additionally, more functionality might be added to the Rover if it were to be sold, which would drive the cost up as well. The additional functionality needed by the farmers could include the ability to carry multiple Nodes at once, the ability to right itself if it fell over, and the inclusion of a Lidar to help the Rover avoid obstacles in its path.

3.3 Server and Web Application

The server used for the project was GlassFish 4.0.1. GlassFish was the official open source JavaEE server that was developed and maintained by Oracle. The JavaEE technologies we used when developing our web application for the server were Servlets for handling HTTP requests, Web Archive (WAR) packages for deploying our static web page content along with our Servlets, and Enterprise Java Beans (EJBs) to provide managed contexts that provide lifecycle support to enable clean initialization and closing of database connections.

Instead of using JAX-RS to build our REST API endpoints for the server, we used plain Servlets. Since we were only dealing with about a dozen API URLs, we decided to work with plain Servlets as they required less configuration to handle the HTTP requests. To actually perform processing on the data we got from the API, we had data transfer object (DTO) classes that represented the Nodes, sensor values from a Node for a specific point in time, Rovers, Rover commands, and Users. We also created controllers that allowed us to pass in DTOs and perform actions such as logging the current sensor values for a Node, creating a new User, or updating an existing command for the Rover. These controllers also acted as the interface between the DTOs and the database, and the controllers also hid the details of how the database worked inside the controllers.

The other part of the web application was the User interface formed by static HTML, CSS, and JavaScript files. As part of the WAR standards, these files could be deployed to the server alongside the Java files. We chose to do this for the project because it allowed us to keep our web files and application files together in the same project, enabling simpler version control, quicker changes during testing and a final application with all its dependencies in a single WAR

file which could be deployed to any JavaEE compliant server including common ones such as TomcatEE, JBoss, Wildfly, Jetty and others.

For security in a production environment, the server should be secured with an SSL certificate so that the data such as passwords, sensor data, and Rover commands are not sent in plain text. The GlassFish server came with a self-signed SSL certificate which prevented such problems, but browsers do not recognize the self-signed certificates as valid by default. For browsers to recognize the certificate by default, an SSL certificate must be acquired from a recognized Certificate Authority (CA).

3.4 Database

The database we used was MongoDB. MongoDB is a document-based NoSQL database. The document based, schema-less nature of MongoDB fit well with IoT applications as preset schemas did not need to make a collection of documents; a collection would be similar to a table in SQL. This meant that documents could have varying contents within a collection which enabled simpler models for things like Rover commands where different data could be added depending on the type of command in question without needing to change anything about the collection.

We created 6 collections for our application within MongoDB: Users, Nodes, Nodes History, Rovers, Rover Commands and Counters. The User collection was the collection that held the information related to a User. A document for a User contained their username, a hashed version of their password, a User's temporary token that was stored in a cookie to authenticate a User on the system, and an expiration time for the temporary token so that the User must login once per day for security. The Nodes collection held the owner (which was a User) of the Node,

the id of the Node, the password for that Node, the name of the Node, the sensor values for the Node, and the location of the Node. The Node collection, besides having a primary index on the Node id, contained an index on the owner of the Node so that a User could quickly pull Nodes belonging to them without needing to scan all the documents in the collection to identify the matching documents. The Node History collection was similar to the Nodes collection, except that it also contained an indexed field denoting the time that the document was inserted so that the time series data for a Node's sensor values could be pulled easily. The Rovers collection was very similar to the Nodes collection containing the primary index of the Rover id, the password, the indexed owner of the Rover, the current Rover location, and the current command the Rover is executing. The Rover Commands collection was a little different. It contained documents with a time based randomly generated primary key that identified a specific Rover command. It also contained an indexed field of the Rover id of the Rover that would execute that command, the details of the command, (drive, pickup Node, drop off Node), and the location at which to perform the command. To create a list of the commands that a Rover will perform, the commands for a specific Rover id were pulled and then sorted by the random time-based primary key such that commands inserted first were the first commands the Rover will perform. This structure followed the natural FIFO command issuance order that people commonly used to program sets of actions. When the Rover completed a command, that command was removed from the database. The final collection was the Counter collection. This collection held counters for generating unique Rover and Node ids to use in the system so that there were never conflicts between two Nodes or Rovers using the system.

The Java web application created the collections and indexes automatically as they were needed, but in order to connect the web application to MongoDB, a connection must be set up on

MongoDB. The process for doing so could be found in Appendix G. Once the steps in Appendix G were performed, the web application would be able to automatically find and connect to a MongoDB instance running on the same machine. To connect to a remote MongoDB server, or to alter the connection settings, the MongoProvider java class would need to be edited to include the new connection settings and the application recompiled.

3.5 User Interface

The User interface consisted of a series of HTML web pages with CSS styling along with JavaScript code to handle the logic of the interface. The HTML and CSS framework for the User interface was built off of the Bootstrap 4 CSS library from Twitter. Bootstrap allowed us faster creation of the User interface since it had templates for buttons, lists, menus, popups and other common UI features prebuilt and styled that we could modify to suit our needs. The large map-based part of our UI came from a JavaScript library called Leaflet which was a commonly used, open source map creator. Leaflet out of the box handled basic User interactions such as zooming, panning, and rotating the map which reduced our workload. Leaflet came with extensions that allowed us to handle click events on the map which we used to place Rover commands as well as functions for putting interactive popup markers on the map. We took advantage of these features heavily to build a responsive map UI that allowed the User to click on markers for where Nodes or Rovers were to interact with, instead of having to search through lists. To enable communications with the server, we used jQuery's HTTP request library to perform REST requests and process the returned data to display on the web page.

3.5.1 User Login

To allow Users to login, we created the login screen shown below in Figure 21. It was fairly basic to keep it easy to use. The top field was for the username, and the bottom field was

for the password. When the User clicked submit, jQuery was used to send a HTTP POST containing the username and password of the User, and the server set a cookie to keep the User logged in for 24 hours in the response if the User entered a valid username and password combination. If the User entered the wrong username and password combination, the system would not let the User login and would give an alert as shown below in Figure 22. Once the User had entered the correct username and password combination, the system would automatically send the User to the screen where they could view the Nodes.

To create a new User, the new User would navigate to the “/swarm/newUser.html” path of the website. It looks identical to the User Login screen except that the text header says “Create Account” instead of “Please Sign In” and the button says “Create” instead of “Login”.

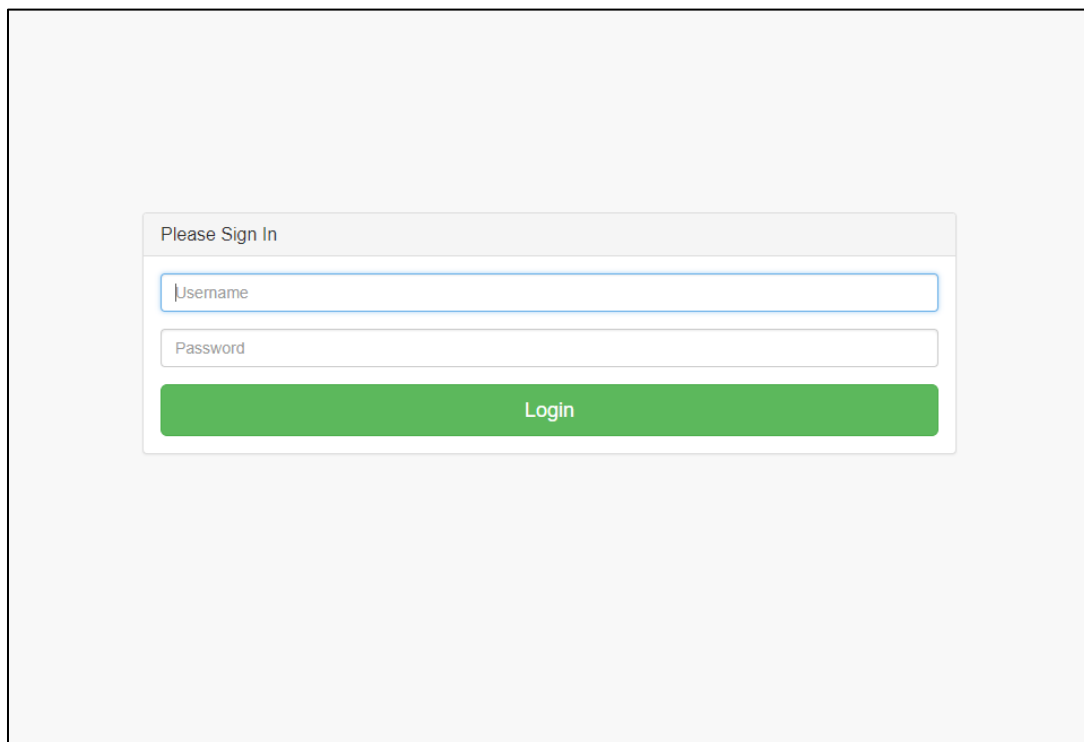
The image shows a user login form centered on a light gray background. The form is enclosed in a white border and has a light gray header bar at the top with the text "Please Sign In". Below the header, there are two input fields: the first is labeled "Username" and the second is labeled "Password". At the bottom of the form is a prominent green button with the text "Login" in white.

Figure 21. User Login Screen

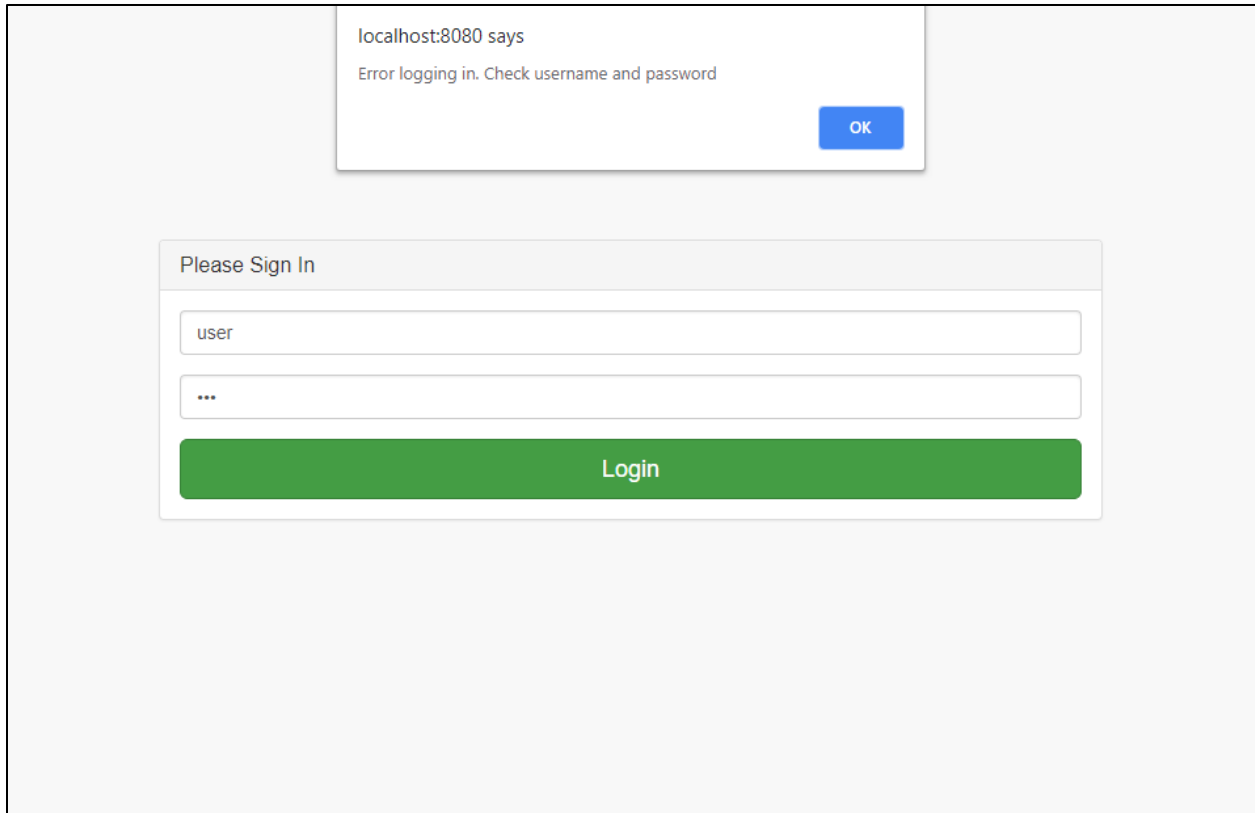


Figure 22. User Login Screen with Incorrect Username and Password Combination

3.5.2 Node Data Visualization

The core part of the Node data visualization system was the map view pictured below in Figure 23. The map view was the main way of interacting with the Nodes on the system, and it was shown below. At the left of the screen was the menu to select what screen to be on, Nodes or Rovers, and the selected choice was darkened. In the center of the screen the map showed the current location of the Nodes. If a marker for a Node was clicked, a popup would show up displaying the Node id and name. If the contents of the popup were clicked, the Node Modal would show up. The alternative way to open up the Node Modal would be to click on its id from the Node list table above the map.

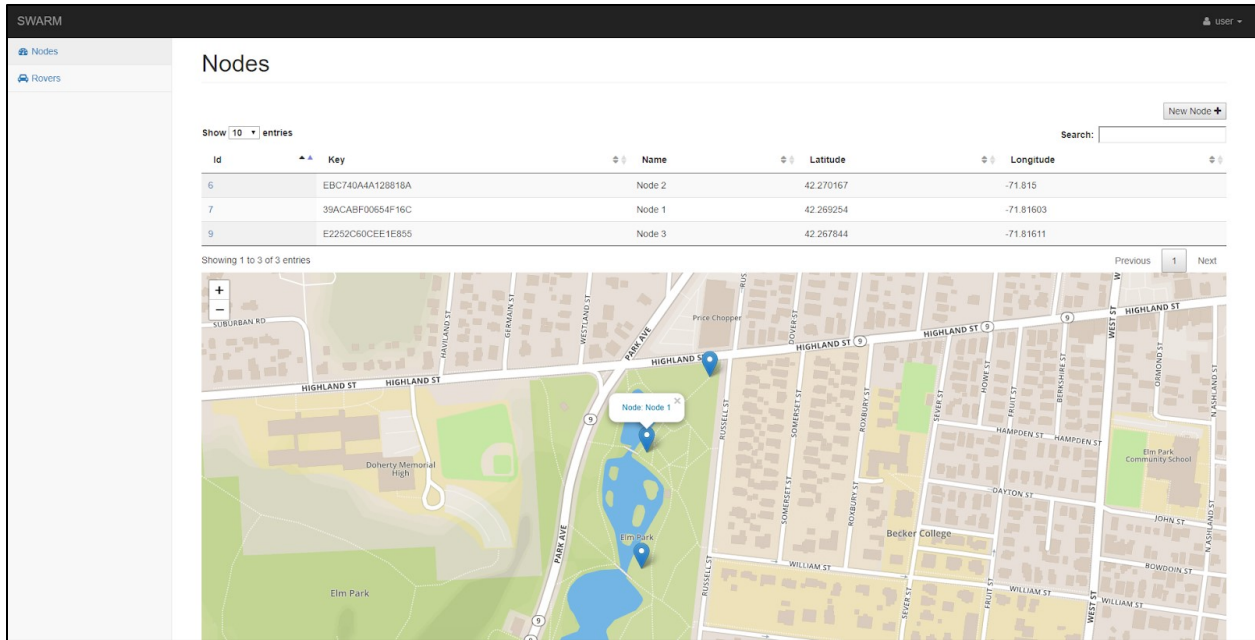


Figure 23. Node Map Screen with 3 Nodes Deployed

The Node Modal shown below in Figure 24 allowed the User to change details about a Node. From the Modal, the Node's key/password, name, and location could be changed or updated. The Modal also showed the latest sensor data from the Node as well as the sensor history of the Node. Shown in Figure 25 the Node's sensor data could be easily retrieved by selecting a time range. Using the JavaScript Plotly library, we plotted the sensor data. This library had the advantage of providing built in functionality that allowed the User to view individual sensor data channels by clicking on the sensor data that they would like to be inspect. Additionally, we added functionality so that when a User moused over the data, the values would directly show on the screen.

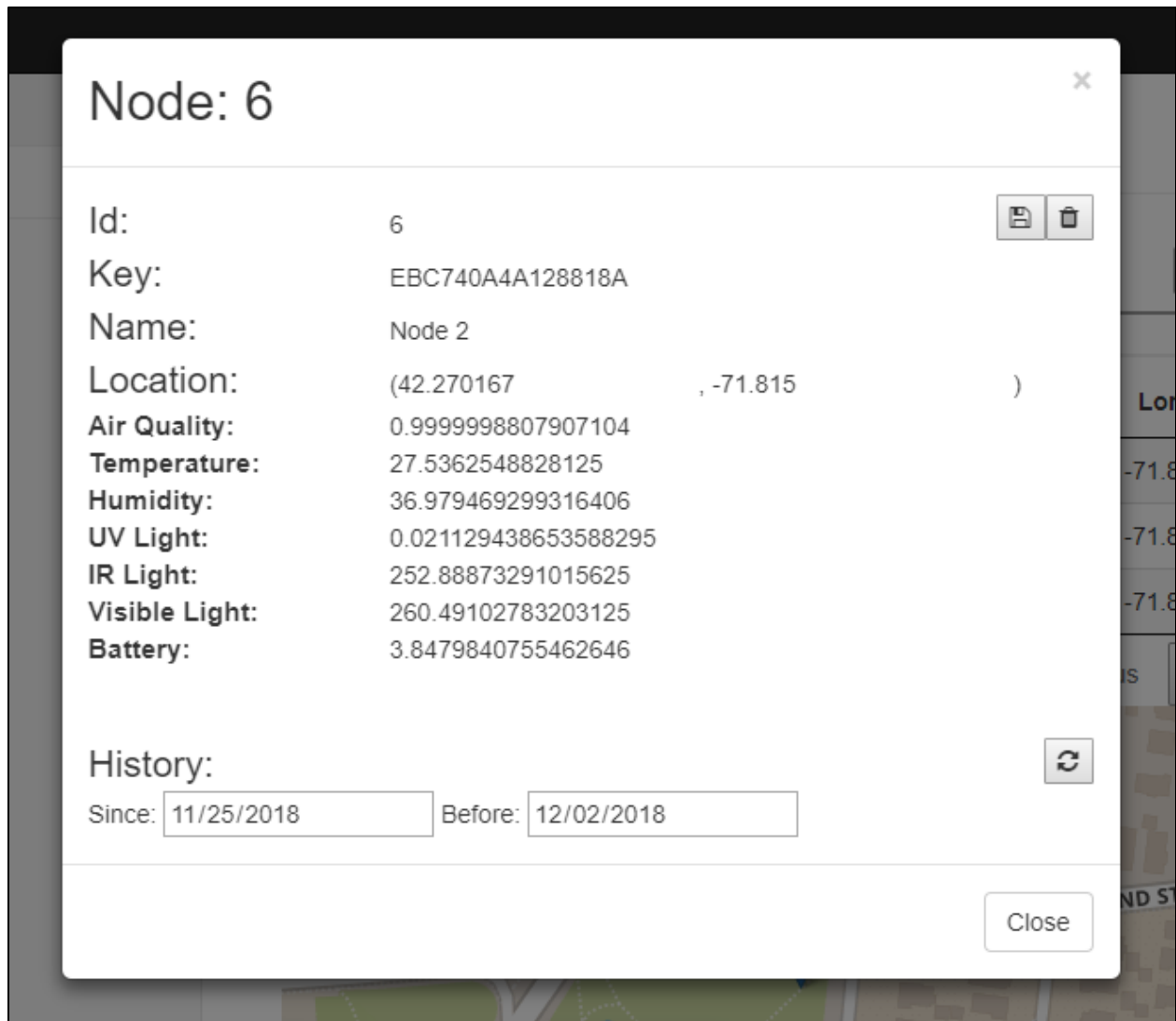


Figure 24. Node Modal that Allows Editing Node Parameters

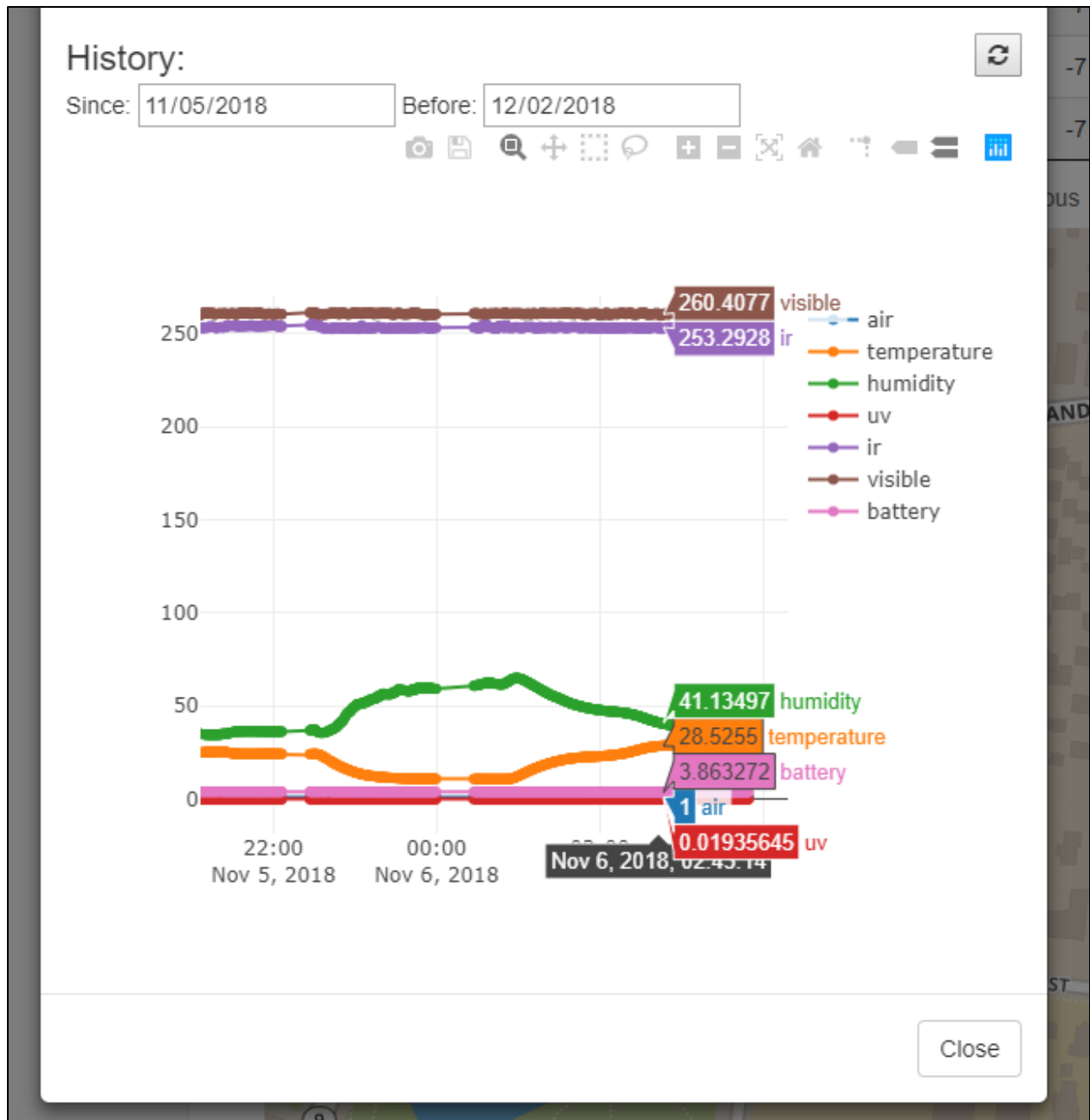


Figure 25. Node Modal Sensor History Data View

3.5.3 Rover Commands

We designed the method for interacting with the Rovers to be as similar as possible to the Node screen so that it should be easier for Users to learn the software. The Rover control

interface is shown below in Figure 26. It had the same general layout as the Node control interface. On the left side was the menu to select from the Node view or the Rover view; the center showed the map that displays the current Rovers that were deployed. Above the map was a table, just like with the Nodes, that showed the deployed Rovers in a table format. By clicking the Rover id in the table or by clicking the marker for the Rover on the map, a Rover Modal would pop up just like for the Nodes. The Modal shown in Figure 27 looked just like the one for the Node and allowed the User to change the Rover's name, key/password, and deployment location.

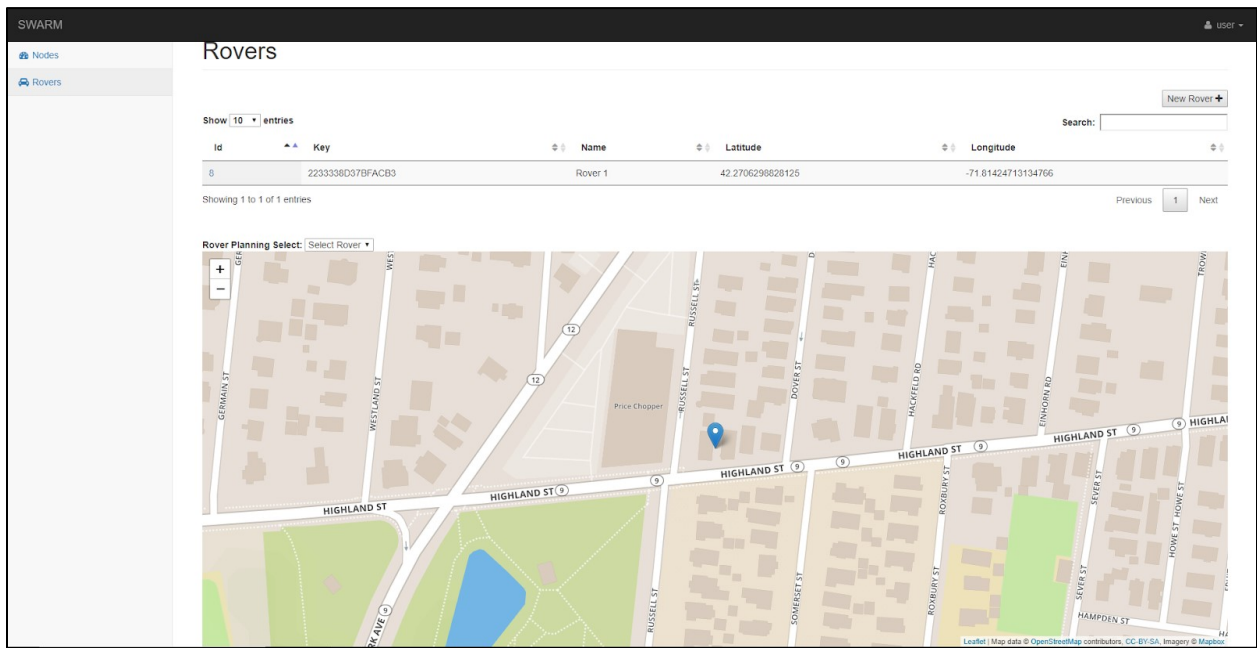


Figure 26. Rover Map Scree with 1 Rover Deployed

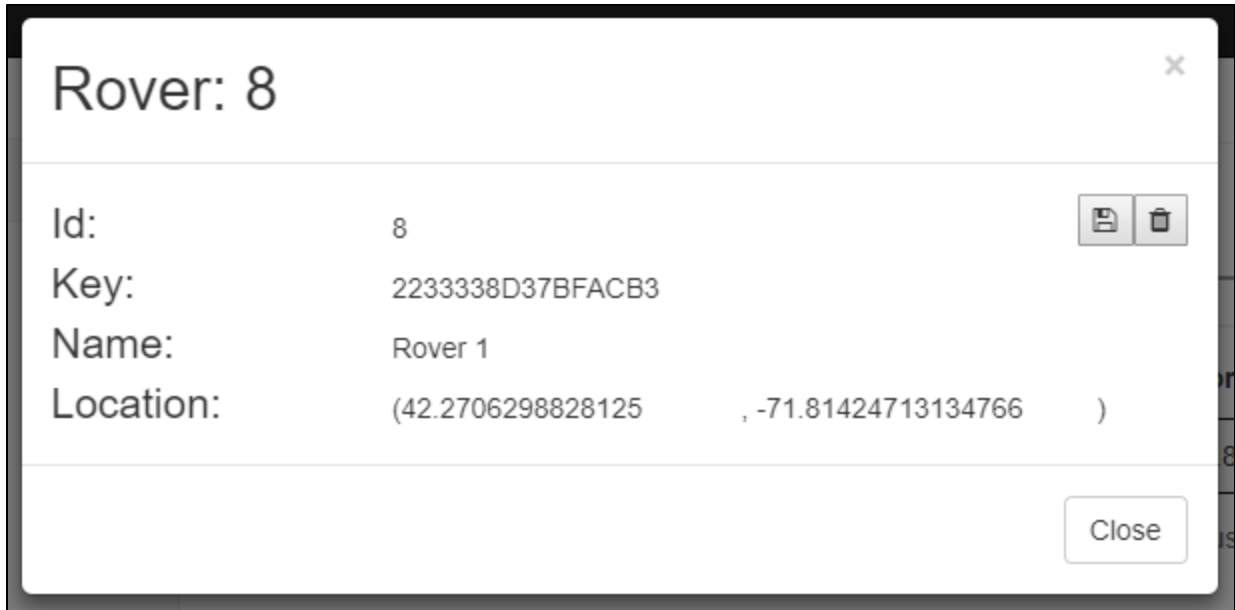


Figure 27. Rover Modal Details

The additional functionality that the Rover map interface provides, was the functionality to program commands for the Rover to follow to pick up, place, and drive to points and Nodes. This was provided by a dropdown selector at the top left corner of the map shown in Figure 28. When the User clicked the dropdown selector, the list of all deployed Rovers would show up. By clicking on one of the Rover names, the map switched into programming mode. To switch back into the view of all the deployed Rover's at once, the page could be refreshed, or the 'Select Rover' option could be selected in the dropdown menu. Once the map was in Rover programming mode, the Rover's commands would show on the map as shown in Figure 29.

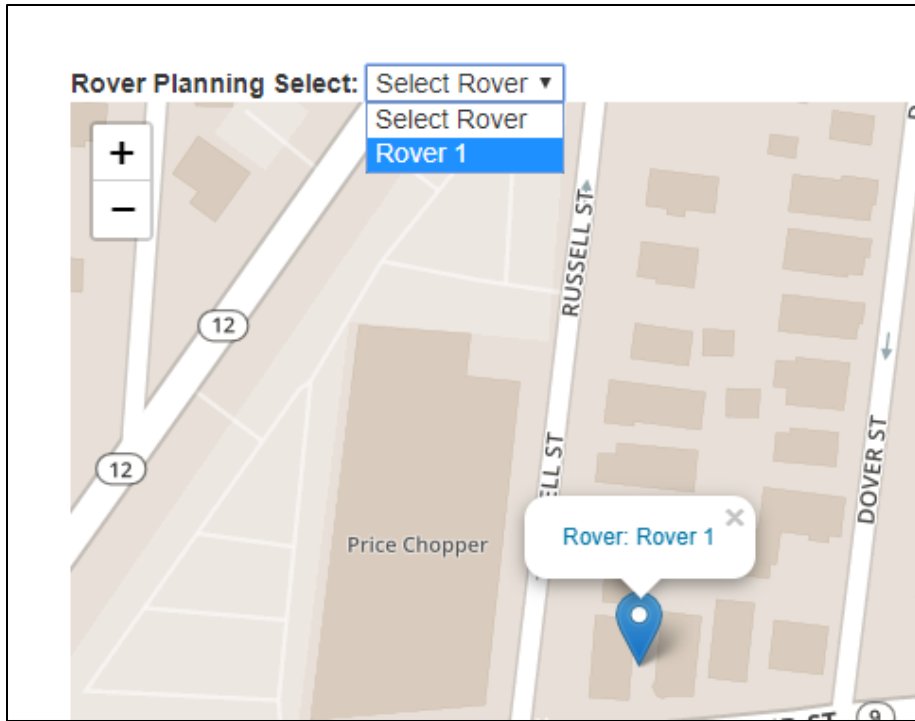


Figure 28. Rover Programing View Selector

The Rover programming screen shown in Figure 29 below displayed the current Rover commands. The commands were shown as markers on the map with colored circles underneath the marker to denote the command: blue for drive, red for drop off, and green for pickup. By clicking on a marker for command, a popup would show up as shown below. In the popup the type of command could be changed or deleted by clicking any of the buttons. The order of the command could be determined by the number in the top left corner of the popup in the bracket where lower commands came first with command 0 being first. As the Rover completed commands, these markers would be removed from the map so that the User did not need to keep track of deleting each command manually.

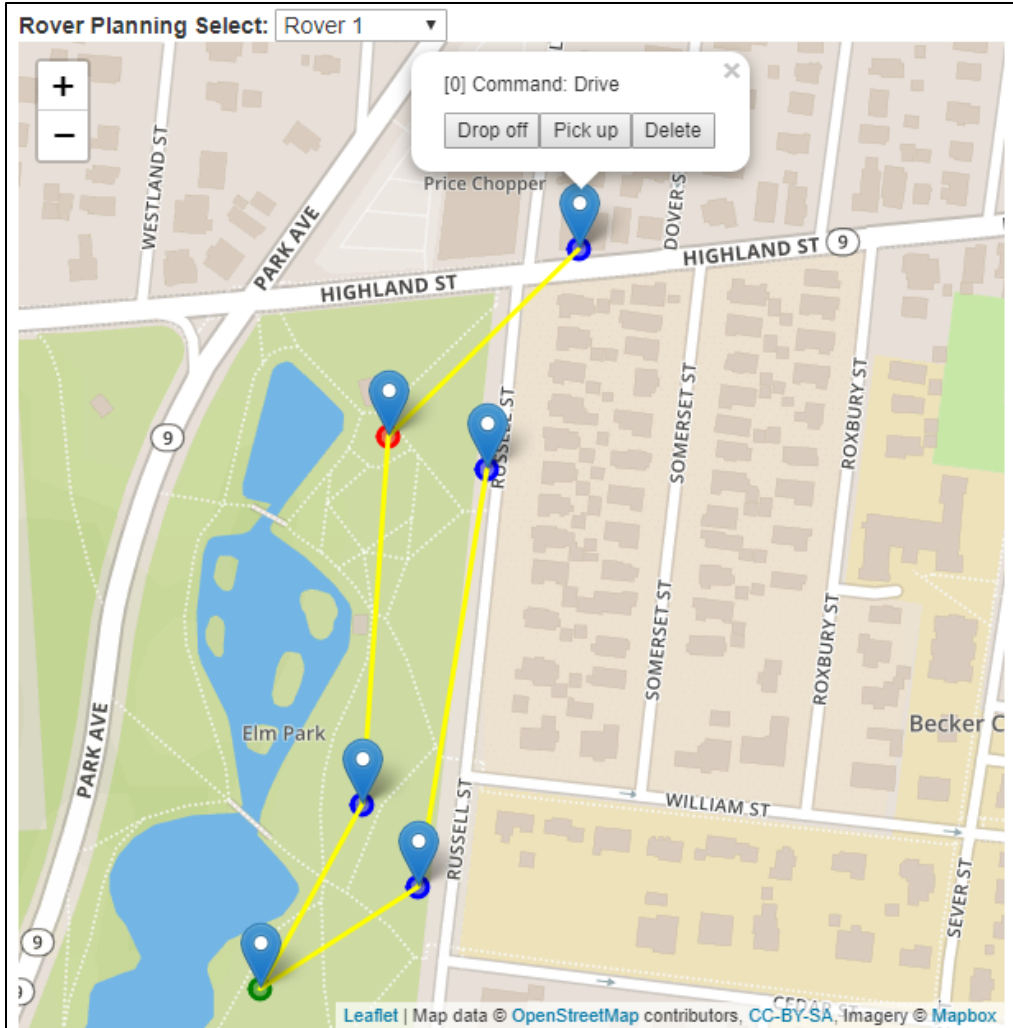


Figure 29. Rover Command View and Popup

4. Discussion and Recommendations

Overall, we were satisfied with the outcomes of the project. We designed and built three Nodes. Each Node was capable for operating for 10 years on battery power while reporting temperature within $\pm 1^{\circ}\text{C}$, relative humidity within $\pm 5\%$, air quality, environmental IR, UV, and visible light levels every 15 minutes. The Nodes were waterproof with a transparent top, yet allowed gas to permeate across a PTFE membrane permitting accurate environmental data. The Nodes successfully were able to transmit their data to a waterproof embedded Gateway that could be mounted outside on a wall. The Gateway enabled the Nodes to successfully send their data to a central server to collect the data as well as communicate with the Rovers to give them commands. The Rover worked and were able to identify Nodes with their camera from up to 10' away and autonomously pick the Node up and place it. The Rover was also able to follow headings and drive towards targets accurately so had the GPS provided more consistent position data, would have been able to navigate large distances to pick up and retrieve Nodes from a field. Our server and user interface were able to connect to all the Nodes and Rovers and update data in near real time. The User could login to view the history of environmental data gathered by the Node as well as create new Nodes or delete existing Nodes on a map. The Users could also use a map interface to program the deployed Rovers to drive to waypoints and pickup or place Nodes.

If we were to have more time or budget, however, there were four areas that we would like to improve. First of all, the Node's circuitry would be designed to be more suitable for mass production. Secondly, the Rover's drive base could perform better. Third, we would like to try different solutions for the GPS function on the Rover. Lastly, the User Interface could be upgraded to add more features. In case another team would like to further work on this project, we listed some recommendations below.

4.1 Node Circuitry

Currently, the Nodes' electronics and sensors were scattered across several boards for development purposes. The microcontroller, sensors, and other parts were either bought separately or inherited from the previous team. Because each board had its own markup, the total markup for the Node was much higher than the actual value of the components. Meanwhile, assembling all the different boards would be too much labor during production. While the Node we developed was good for a prototype, using the prototype design with separate boards would be too expensive and labor intensive for anything made for production.

If one were to mass-produce Nodes, it would be more efficient to design a customized PCB board that contains all the necessary components, including the microcontroller, radio, sensors, and battery connector all on the same board. By putting everything on the same board using discrete IC components, the cost of the overall Node would be much lower. Another advantage of custom PCB boards would be the ability to reduce the size of the Node making it easier for the Rover to transport them and reduce the cost of enclosures. We would recommend using a board layout program like KiCAD or Eagle to design the layout and outsource the manufacturing of the PCB to services like Oshpark.

4.2 Rover Drive Base

We used the Wild Thumper 4 Wheel Drive for the Rover's drive base because the previous team obtained this particular drive base with their budget. This drive base could drive on uneven terrain with its independent suspension for each wheel. However, the motors were unable to operate under lower power for more precise maneuvers such as slowing down to approach the Node and especially for turning. Turning was a big issue for this drive base; some wheels would lose traction while turning because the suspension was too soft causing some

wheels to tilt in while the Rover rode up higher on others. We overcame these difficulties in software by having the robot use more power when attempting to approach the Node and accepting it might need to correct a lot more. With a better drive base, the Rover would be able to more accurately drive to the Nodes and pick them up and the reliability would be much higher. If the Rover were getting redesigned for all-weather use, a waterproof drive base would be a necessity as well as the Wild Thumper drive base was not waterproof.

4.3 Rover Navigation

Our Rover relied on accurate GPS signals to follow the User designated waypoints for navigation. Accurate GPS signals were difficult to obtain in Worcester. After we installed an active antenna for our GPS unit rated to sub 2.5m accuracy, we still received inconsistent positioning data with the reported position jumping around due to interferences in the city. While we believed that the inaccuracy would not be as severe on a farm, we would still recommend a differential GPS unit for the Rover and a static base station installed on the farm. The differential GPS and base station combination would ensure that the Rover would receive correction information from the static base station and help pinpoint the Rover's location within a few centimeters.

Although the User could use the map on the interface to visualize the Rover's route, they could not predict all potential obstacles en route. To make sure that the Rover would be safe during travel, additional sensors like ultrasonic and lidar sensors could be implemented to help Rover navigate along the way. Potentially, the Rover could incorporate Simultaneous Localization and Mapping (SLAM) to map out the farm and use that information to help it navigate between the waypoints even better.

4.4 User Interface

The server and database performed well for the task at hand. Both the database and the JavaEE server were deployed on an 8GB, quad-core machine which was easily able to keep up with data being added at the rate of several dozen sensor data points per second. The only functionality that was not parallelizable in the servers was the generation of unique ids for Rovers and Nodes, which was not really an issue because there should not be millions of new unique ids created each second for deploying new Nodes and Rovers. The rest of the functionality provided by the system was parallelizable so that it can be horizontally scaled to handle more load. Despite the backend working well, the User Interface on the front end could use more work in the future.

Currently, the User Interface (UI) did not employ smart HTML document object model (DOM) manipulation. When updates to data displayed were available, the UI code did not only update the DOM that needed to be updated. Instead, it replaced the entire DOM tree from the root that the change originated from. The extra DOM updating could be helped by using larger web frameworks like Angular JS, React JS, or Vue JS that implement smarter DOM manipulations based on data to DOM bindings.

Additional functionality, such as heat-maps overlaid on the map gathered from the Nodes, would improve the UI. The data is already present, it just would need work to display it in such a manner on the map. Similar features could be added for animated history overlays of temperature, humidity, light, and air quality on the map. While the added functionality would not be strictly necessary for use, it would make usage easier and more intuitive for many Users.

5. Citations

Aleksandrova, M. (2018). *IoT in Agriculture: Five Technology Uses for Smart Farming and Challenges to Consider - DZone IoT*. [online] dzone.com. Available at: <https://dzone.com/articles/iot-in-agriculture-five-technology-uses-for-smart> [Accessed 23 Oct. 2018].

Beagleboard.org. (2018). *BeagleBoard.org - black*. [online] Available at: <https://beagleboard.org/black> [Accessed 23 Oct. 2018].

Daniels, J. (2018). *From strawberries to apples, a wave of agriculture robotics may ease the farm labor crunch*. [online] CNBC. Available at: <https://www.cnbc.com/2018/03/08/wave-of-agriculture-robotics-holds-potential-to-ease-farm-labor-crunch.html> [Accessed 23 Oct. 2018].

Digibale.com. (2018). *DigiBale Ltd - People First, Then Technology*. [online] Available at: <https://www.digibale.com/shop/products/farm-automation-starter-kit-pre-order/> [Accessed 23 Oct. 2018].

Seedstudio.com. (2017). *Grove - Air quality sensor v1.3 - Sensor - Seed Studio*. [online] Available at: <https://www.seedstudio.com/Grove-Air-quality-sensor-v1-3-p-2439.html> [Accessed 23 Oct. 2018].

Seedstudio.com. (2017). *Grove - Sunlight Sensor - Sensor - Seed Studio*. [online] Available at: <https://www.seedstudio.com/Grove-Sunlight-Sensor-p-2530.html> [Accessed 23 Oct. 2018].

Seedstudio.com. (2017). *Grove - Temperature&Humidity Sensor (SHT31) - Sensors - Seed Studio*. [online] Available at: <https://www.seedstudio.com/Grove-Temperature-Humidity-Sensor-SHT3-p-2655.html> [Accessed 23 Oct. 2018].

Industries, A. (2018). *Adafruit Feather M0 RFM69HCW Packet Radio - 433MHz*. [online] Adafruit.com. Available at: <https://www.adafruit.com/product/3177> [Accessed 23 Oct. 2018].

Jefferson, N., Raspe, P., Tran, N. and Ciaraldi, M. (2017). *Agricultural Swarm Robotics with Distributed Sensing*.

Digikey.com. (2018). *PN-1323-CMB Bud Industries | Boxes, Enclosures, Racks | DigiKey*. [online] Available at: <https://www.digikey.com/product-detail/en/bud-industries/PN-1323-CMB/377-1890-ND/2674154> [Accessed 23 Oct. 2018].

Post, S. (2018). *What is IoT in Agriculture? Farmers Aren't Quite Sure Despite \$4bn US Opportunity - report - AgFunderNews*. [online] AgFunderNews. Available at:

<https://agfundernews.com/iot-agriculture-farmers-arent-quite-sure-despite-4bn-us-opportunity.html/> [Accessed 23 Oct. 2018].

Cfbf.us. (2017). *Search for Solutions: California Farmers Continue to Struggle with Employee Shortages*. [online] Available at: <http://www.cfbf.us/wp-content/uploads/2017/10/CFBF-Ag-Labor-Availability-Report-2017.pdf> [Accessed 23 Oct. 2018].

Seibel, M. (2017). *Barriers to Deploying Internet-of-Things (IoT) on the Farm*. [online] Hacker Noon. Available at: <https://hackernoon.com/barriers-to-deploying-internet-of-things-iot-on-the-farm-eca1bc46b2> [Accessed 23 Oct. 2018].

Digikey.com. (2018). *VENT-PS1YBK-N8001 Amphenol LTW | Boxes, Enclosures, Racks | DigiKey*. [online] Available at: <https://www.digikey.com/product-detail/en/amphenol-ltw/VENT-PS1YBK-N8001/1754-1230-ND/7898285> [Accessed 23 Oct. 2018].

Wikipedia.com (2018) Systems Integrator. [Online]. Available at: https://en.wikipedia.org/wiki/Systems_integrator [Accessed 10 Nov 2018]

Appendices

Appendix A – Smart Agriculture Solutions Comparisons

	Our System (minus Rover)	DigiBale Agriculture Starting Kit	Libelium Fasal Smart Agriculture Basic Solution	Arable Sensor Node
System Description	3 Nodes, 1 Gateway	3 Nodes, 1 Gateway	1 Node	1 Node
Connection	Ethernet to Gateway	Ethernet to Gateway	Cellular to Node	Cellular to Node
Sensors	<ul style="list-style-type: none"> • Temperature • Humidity • Air Quality • UV Light • IR Light • Visible Light • Battery 	<ul style="list-style-type: none"> • Temperature • Soil Moisture • Battery • Radio signal strength 	<ul style="list-style-type: none"> • Temperature • Humidity • Pressure • Soil Moisture • Leaf Wetness 	<ul style="list-style-type: none"> • Temperature • Humidity • Pressure • Solar Radiation • Evapotranspiration • Precipitation • Battery
Software Package	Yes	No	No	Yes
Cost	\$500	\$1000	\$2000	Unknown

Appendix B - Wireless Range Calculations

Friis Transmission Equation:

$$P_r = \frac{P_t * G_t * G_r * \lambda}{(4\pi R)^2}$$

P_r = Power at receiver
 P_t = Power of transmitter
 G_r = Receiving antenna gain
 G_t = Transmitting antenna gain
 λ = Wavelength in meters
 R = Distance between antennas

Node to Gateway Parameters:

P_r = -89dBm
 P_t = 17dBm
 G_r = 3dBi
 G_t = 3di
 λ = 0.33m
 R = Distance between antennas

R is ~8km. However, P_r does not include any path loss or interference losses. Based on previous RF knowledge, and Fresnel zone occlusions, the total path and interference losses can be around 15dBm over a 5km link. So, recalculating with P_r at -74dBm to account for path loss.

$$\mathbf{R = 1.8km}$$

Appendix C – Node Power Consumption

Component	Sleep	Awake	Transmitting
Cortex M0 Board	4uA	4.2mA	4.2mA
RFM69HCW Radio	0.1uA	1.25mA	94mA
3.3V Regulator	2uA	10uA	10uA
Temperature/Humidity Sensor	1.7uA	1.7uA	1.7uA
Air Quality Sensor	1.3uA	1.3uA	1.3uA
Light Sensor	1.4uA	1.4uA	1.4uA
Total Power Consumption	10.5uA	5.46mA	98.23mA
Contextualized Power Draw	0.252mAh/24hrs	0.000003mAh/ transmit	0.00022mAh/ transmit

Calculating power consumption for 1 transmit every 15 minutes:

At 1 transmit every 15 minutes there are 96 transmits/day. The power consumption is **0.273408mAh/day** at that rate.

Calculating lifetime on 2500mAh battery:

$$2500\text{mAh} / (0.273408\text{mAh/day}) = 9150 \text{ days} = \sim 25 \text{ years best case}$$

Based on 50% self-discharge of LiPo batteries over long time periods plus a safety factor, the **actual lifetime is likely around 10 years.**

Appendix D – Node Cost

Item	Cost
Cortex M0 Feather with RFM69HCW Radio	\$ 25.00
Protoboard Feather	\$ 6.00
Stacking Female Headers	\$ 1.00
UFL connector	\$ 0.50
3dBi UFL 433MHz Antenna	\$ 9.00
Battery	\$ 3.00
I2C Hub	\$ 3.00
Air Quality Sensor	\$ 10.00
Temperature and Humidity Sensor	\$ 12.00
Sunlight Sensor	\$ 10.00
Enclosure	\$ 12.60
PTFE Vent	\$ 2.52
Threaded Rods	\$ 4.00
PVC Pipe	\$ 0.25
Lock Nuts	\$ 2.00
Acrylic	\$ 0.10
Total	\$ 100.97

Appendix E – Gateway Cost

Item	Cost
Cortex M0 Feather with RFM69HCW Radio	\$ 25.00
Ethernet Feather	\$ 20.00
UFL connector	\$ 0.50
3dBi UFL 433MHz Antenna	\$ 9.00
POE Adapter	\$ 8.00
Short Ethernet Cable	\$ 2.00
Ethernet Cable Gland	\$ 9.00
5V Power Supply	\$ 5.00
Enclosure	\$ 12.60
Acrylic	\$ 0.05
Total	\$ 91.15

Appendix F – Rover Cost

Item	Cost
Wild Thumper 4WD	\$ 175.00
VEX 7.2V 3000mA battery	\$ 30.00
Vex charger	\$ 19.00
Cortex M0 Feather with RFM69HCW Radio	\$ 25.00
Protoboard Feather	\$ 6.00
Stacking Female Headers	\$ 1.00
UFL connector	\$ 0.50
3dBi UFL 433MHz Antenna	\$ 9.00
Feather GPS	\$ 40.00
Active GPS Antenna	\$ 9.67
Feather Servo Controller	\$ 10.00
Triple-Axis Compass	\$ 15.00
Raspberry Pi	\$ 38.00
RPi Camera	\$ 30.00
RPi Motor Driver	\$ 52.00
USB Cable	\$ 3.00
HS 485 Servos	\$ 32.00
Wood	\$ 5.00
Wood Glue	\$ 0.50
Acrylic	\$ 0.10
Total	\$ 500.77

Appendix G – MongoDB Setup

1. Install Mongo DB 4.0
2. Run the following command to start Mongo in the non-authenticated mode

```
start /d "<Mongo_4.0_Install_Path>\bin" mongod.exe --dbpath
"<Database_Folder_Path>"
```

3. Run mongo.exe from the terminal
4. Type the following commands into the mongo shell to create a user admin

```
>use admin
>db.createUser({user:"userAdmin", pwd: "uAPass",
roles:[{role: "userAdminAnyDatabase", db:"admin"}]})
```

5. Shutdown mongo and mongod
6. For Windows create the following batch script and run it to start mongod in auth mode

```
start /d "<Mongo_4.0_Install_Path>\bin" mongod.exe --auth -
-dbpath "<Database_Folder_Path>"
Pause
```

7. Start mongo.exe again and type the commands below to login as a user admin

```
>use admin
>db.auth("userAdmin","uAPass")
```

8. Run the following commands to add a read/write user to operate on the application database

```
>use swarm
>db.createUser({user:"swarmApp", pwd: "swAppPass",
roles:[{role: "readWrite", db:"swarm"},{role: "read",
db:"reporting"}]})
```

9. To run the mongo.exe application as the new user run the following command in the terminal. At this point, the web application will be able to connect to it

```
mongo --port 27017 -u "swarmApp" -p "swAppPass" --
authenticationDatabase "swarm"
```

Appendix H - Node Software Setup

1. Download and install the Arduino IDE (<https://www.arduino.cc/en/Main/Software>)
2. Copy the libraries in 'node/libraries' directory from the zipped source code file to the Arduino libraries folder
3. Download the 'Feather M0' board definition from Adafruit via the Arduino Board Manager (Tools->Board->Board Manager)
4. Open the 'NodeTx.ino' file in the 'node' directory in Arduino
5. Edit the const ID and KEY byte arrays defined at the top of the file to match the Node id and password stored on the server.
6. Upload the program to the Node
7. After the program finishes uploading the Node is fully setup

Appendix I - Gateway Software Setup

1. Download and install the Arduino IDE (<https://www.arduino.cc/en/Main/Software>)
2. Copy the libraries in 'gateway/libraries' directory from the zipped source code file to the Arduino libraries folder
3. Download the 'Feather M0' board definition from Adafruit via the Arduino Board Manager (Tools->Board->Board Manager)
4. Open the 'GatewayRx.ino' file in the 'gateway' directory in Arduino
5. Edit the const HOST char array defined at the top of the file to match the location of the target server.
6. Upload the program to the Gateway
7. After the program finishes uploading the Gateway is fully setup

Appendix J - Rover Software Setup

Rover Arduino

1. Download and install the Arduino IDE (<https://www.arduino.cc/en/Main/Software>)
2. Copy the libraries in 'rover/libraries' directory from the zipped source code file to the Arduino libraries folder
3. Download the 'Feather M0' board definition from Adafruit via the Arduino Board Manager (Tools->Board->Board Manager)
4. Open the 'RoverCom.ino' file in the 'gateway' directory in Arduino
5. Edit the const ID and KEY byte arrays defined at the top of the file to match the Rover id and password stored on the server.
6. Upload the program to the Rover Arduino
7. After the program finishes uploading the Rover Arduino is fully setup

Rover Raspberry Pi

1. Install Python 3
2. Install the 'PySerial' and 'dual_g2_hpmd_rpi' modules for Python using pip3
3. Install OpenCv 2 with Python bindings
4. Copy the files in 'rover/rpi' to a folder on the Raspberry Pi
5. To check if the Rover can find a Node using vision, run 'search.py'
6. To check if the Rover's drive is working, run 'drive.py'
7. To check if the Rover's camera is working, run 'snap.py'
8. To run the Rover software and have it talk to the server, run 'main.py'

Appendix K - Server Software Setup

1. Setup MongoDB according to the instructions in Appendix G
2. Install a JavaEE webserver such as GlassFish, although after 2019 Payara is probably a better bet for support.
3. Deploy the 'SwarmMQP.war' file in the 'server' folder of the zipped source code file to the webserver.
4. The application should be up and running.