

Department of Electrical Engineering and Computer Science  
EECS Senior Design 2021

# **Modular Garden Monitoring System**

## **Team CE12**

Sadie Gladden, Eric Krenz, Zuguang Liu, Alan Trester

April 6, 2021

**Technical Advisor**  
Dr. Zachariah Fuchs

University of Cincinnati  
College of Engineering and Applied Science  
EECE 5031 CompE Senior Design I - 001

## Acknowledgements

We would like to sincerely thank the following individuals:

- **Dr. Carla Purdy** for your continued guidance and wisdom throughout the Senior Design process.
- **Dr. Zach Fuchs** for supporting, advising, teaching, and dealing with us throughout the creation and development of this project.
- **Hive 13 Makerspace** for providing work space and equipment that was vital to the development of our prototype.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Problem/Need . . . . .	4
1.2	Solution . . . . .	4
1.3	Credibility . . . . .	5
1.4	Project Goals and Brief Methodology . . . . .	6
<b>2</b>	<b>Discussion</b>	<b>7</b>
2.1	Project Concept . . . . .	7
2.2	Design Objectives . . . . .	8
2.3	Methodology/Technical Approach . . . . .	10
2.4	Standards . . . . .	12
2.5	Final System Design . . . . .	13
2.6	Testing and System Performance . . . . .	17
2.7	Budget . . . . .	19
2.8	Timeline . . . . .	20
2.9	Problems Encountered and Solutions . . . . .	21
2.10	Future Recommendations . . . . .	22
<b>3</b>	<b>Conclusion</b>	<b>23</b>
<b>4</b>	<b>Appendices</b>	<b>25</b>
4.1	Python Implementation of GUI . . . . .	25
4.2	Submodule Main Control Program . . . . .	34
4.3	Submodule Libraries . . . . .	37

## **Abstract**

The MGMS is an IoT solution to help homeowners, garden enthusiasts, and farmers care for their lawns and gardens in a more informed and effective way while also reducing wasteful water usage. This proof-of-concept system designed over the 2020/2021 fall and spring semesters utilizes an AVR microcontroller and IEEE 802.15.4 wireless communication to provide real-time and historical information about outdoor or indoor environmental conditions while providing customized recommendations for optimal plant care.

# 1 Introduction

The purpose of this report is to divulge in its entirety, the process followed to fulfill all requirements of EECE5031/5032–Senior Design, at the University of Cincinnati’s College of Engineering and Applied Science. The culminating project is a proof-of-concept modular garden monitoring system.

## 1.1 Problem/Need

Lawns and gardens are one of most essential elements for the typical American home. A survey conducted by National Association of Landscape Professionals in 2019 shows that 79 percent of American families value lawns when renting or buying a home, and about one in three Americans garden in their yards multiple times a week[1].

Consequently, there is a constantly high demand of water for use in lawns and gardens. Per the United States Environmental Protection Agency, about 48 gallons of water is devoted for this use per family per day. Across America, nearly 1/3 of all residential water is used for landscaping irrigation totaling an estimated 9 billion gallons per day[2]. In a world undergoing climate change with consistent annual water shortages and wildfires in many parts of the world, wasteful water usage is simply irresponsible and unacceptable.

The issue of wasteful irrigation is not being addressed as actively as it deserves to be. Although younger generations of Americans tend to value lawns and gardens even more than older generations, more than half of young people failed quizzes on proper landscape care and nearly 7 out of 10 young people wish to see further improvement in their lawns[3]. Uninformed (and in turn, irresponsible) lawn care may significantly contribute to the amount of wasteful water usage happening every day.

A 21st-century solution is needed to help new homeowners care for their lawns and gardens in a more informed and effective way while reducing the amount of wasteful water usage that is accounted for by residential lawn care and irrigation.

## 1.2 Solution

Originated from the Internet of Things (IoT) concept, the Modular Garden Monitoring System (MGMS) is a proposed solution that will be able to provide real-time and historical information about environmental conditions. Simply having detailed information on-hand will allow homeowners to make more informed decisions on the types of plants to keep in their gardens as well as when and how much to water them. Internet connectivity can take decision making to the next level by being able to crowd-source gardening recommendations and consider local weather predictions for watering. Further system expansions can introduce features such as automatic watering to take work from homeowner’s shoulders while reducing human error in the garden care process. Finally, a smart design will allow the system to be flexible and applicable in a variety of scenarios varying with garden size and irrigation needs and even between residential and industrial settings. This senior design project will focus on creating a proof-of-concept baseline system to report and access environmental data. The user interface will be built in such a way that will enable smooth addition of the smart features discussed above.

### 1.3 Credibility

**Alan Trester** is an Electrical Engineering student with co-op experience in software, hardware, and manufacturing engineering roles through GE Aviation Systems. He has a strong passion for technology, design, and "making". After graduating he will begin full-time work as an Electrical Engineer with Raytheon Missile & Defense in Tucson, Arizona.

**Eric Krenz** is a Computer Engineering student whose past co-op experience was in hardware, software development, and cybersecurity. He has a passion for engineering, technology, and making the world a better place. Post graduation he will begin his career at Epic Systems in Madison, Wisconsin.

**Sadie Gladden** is a Computer Engineering student with co-op experience in software development, user interface creation, game engine development, computer graphics, and cloud solutions through Siemens PLM and Siemens Healthcare GmbH. She enjoys exploring the relationship between hardware and software and exploring the connection and overlap of technology and medicine.

**Zuguang Liu** is an Electrical Engineering student who has past Co-op experience in industrial system design, embedded system hardware design, and simple machine learning implementation. After finishing a Bachelor's Degree with an Embedded Systems minor, he will continue to pursue a Master's Degree in Electrical Engineering.

Team member responsibilities often overlapped during the development of the MGMS. Overall however, task assignments were chosen based on each members personal engineering strengths and abilities. Figure 1 illustrates each team member's experience and project focus for the development of the MGMS. This image was originally presented during the design planning phase of the project.

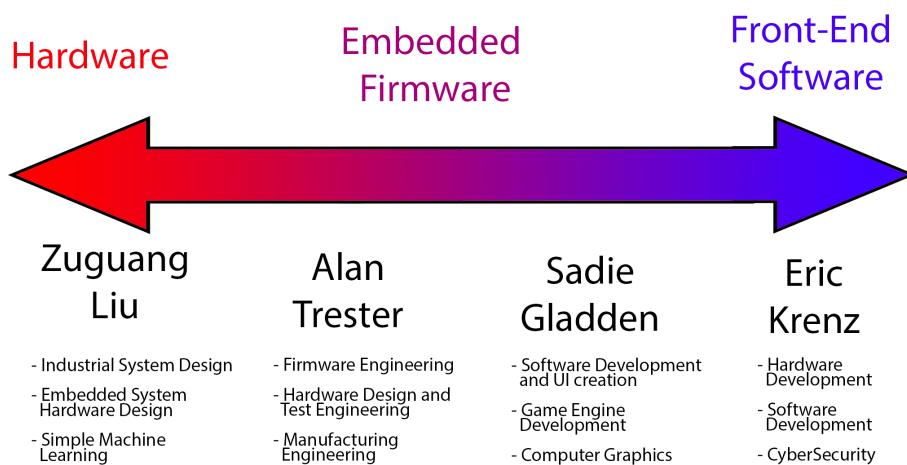


Figure 1: Team members past experience, specializations, and credibility for this project.

**Dr. Zachariah Fuchs** (fuchsze@ucmail.uc.edu) is the professor for Introduction to Mechatronics. He has extensive knowledge on embedded system design, sensor fusion, robotics and control systems. His areas of expertise make him a good source of advice on anything from the top-level architecture of the system to individual component choices and design

## 1.4 Project Goals and Brief Methodology

Our proposed solution is a modular garden monitoring system that will be able to provide real-time and historical information about environmental conditions such as soil moisture, temperature, sunlight, humidity, and so on. This is a depth of information that is not necessarily available on existing products; Simply having this detailed information on-hand will allow homeowners to make more informed decisions on the types of plants to keep in their gardens as well as when and how much to water them. This removes the need for a strong knowledge in horticulture for effective results as well as taking water conservation up a notch. The intended proof-of-concept system will be able to report on and present real-time and historical data on environmental conditions. Project work will focus on the system's wireless communications capabilities, sensor interface, and user interface platform.

The desired characteristics of the MGMS were defined in an attribute table and identified as objectives, constraints, functions, or means. This is presented in Figure 2. These identified attributes are referenced throughout the design planning and testing process.

Attribute Table				
Characteristic	Objective	Constraint	Function	Means
Hardware				
Measures Environmental Conditions			✓	
Accurate	✓			
Expandable and Modular	✓		✓	
Waterproof / Weatherproof		✓		
Inexpensive	✓	✓		
Wireless Communication & Power		✓		✓
Easy to Set Up Outside	✓	✓		
Low Power Consumption		✓		✓
Software				
Easy to Use, Intuitive UI	✓			
Saves Historical Data			✓	
Shows Real-Time Conditions			✓	
Links to Recommended Growing Conditions			✓	
Predicts Weather			✓	
User Configurable	✓		✓	

Figure 2: Attribute Table used during the design decision making process.

This kind of system can be made using a simple embedded system architecture. A basic microcontroller (or family of controllers), which can be chosen to be used throughout the system modules, will be capable of any combination of reading inputs from sensors, sending information, receiving information, and interacting with actuators such as solenoid valves to control water flow. The heart of the system lies in the radio modules, such as Zigbee radios, which allow for full modularity and good wireless range. Finally, the forward-facing system UI can be hosted on a tiny computer such as a Raspberry Pi.

## 2 Discussion

### 2.1 Project Concept

A garden monitoring system such as the one we are proposing is not a novel idea: several products already exist within the consumer and industrial farming markets with similar approaches towards data collection. The Onset HOBO.net system is a web-enabled data-collection solution for industrial farmers. While these systems are very popular and provide good results, with accessible user interfaces and informative data visualization, they are too expensive for consideration by homeowners and don't have the necessary features such as garden suggestions to be applicable in that market [4]. The Edyn Garden Sensor was a consumer-targeted system that aimed to tackle the same problems as the MGMS, unfortunately the product was burdened with limited modularity and expandability as well as a poorly designed app interface [5]. Characteristics of both products are analyzed and, along with interviews and the team's own expectations, are used to set a reasonable objectives baseline for the new system.

A pairwise comparison chart is used to identify priorities among certain attributes that are identified in Figure 2 above. Again, these attributes are considered throughout the design process and are presented in Figure 3.

Pairwise Comparison Chart						
Goals	Accurate	Easy to Use	Inexpensive	Weatherproof	Modular	Score
Accurate	-	0	1	0	1	2
Easy to Use	0	-	1	0	0	1
Inexpensive	0	0	-	0	0	0
Weatherproof	1	1	1	-	1	4
Modular	1	1	1	0	-	3

Figure 3: Pairwise Comparison Chart for the design process of our prototype.

## 2.2 Design Objectives

The end-goal of this project is to develop a proof-of-concept marketable product to functionally address the issues previously discussed in the problem statement: Poor gardening practices and Water conservation. To aid the project development structure, two objective trees are created, referencing identified attributes and priorities from before, for the hardware and software components of the system separately. These charts are presented in Figures 4 and 5 respectively.

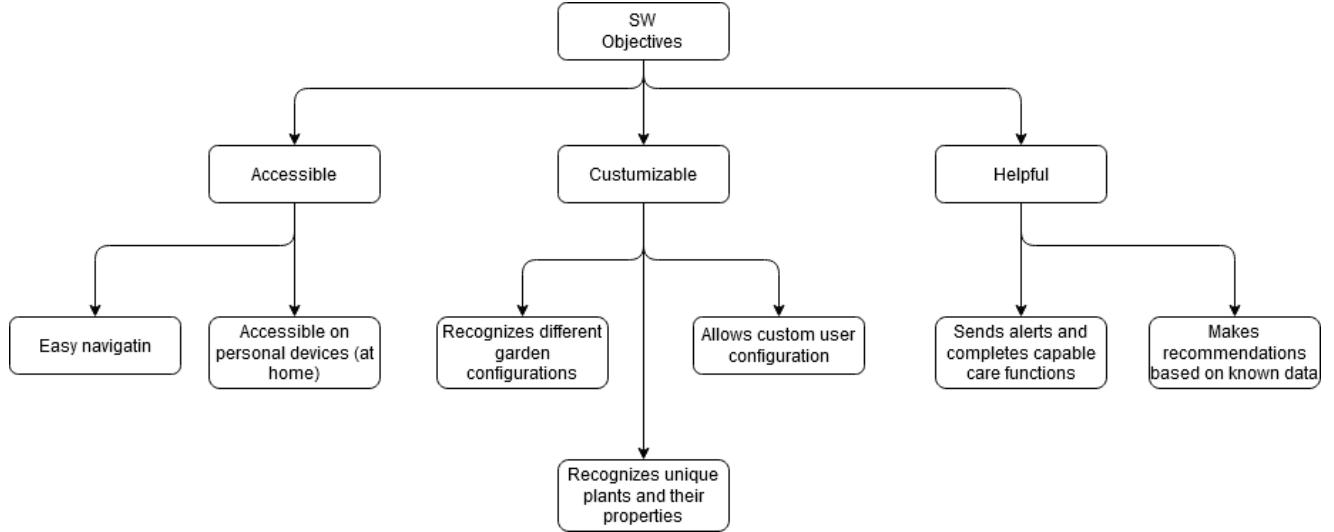


Figure 4: Software Design and Implementation Objective Tree for the design priorities of creating a product that is Accessible, Customizable, and Helpful for the consumer.

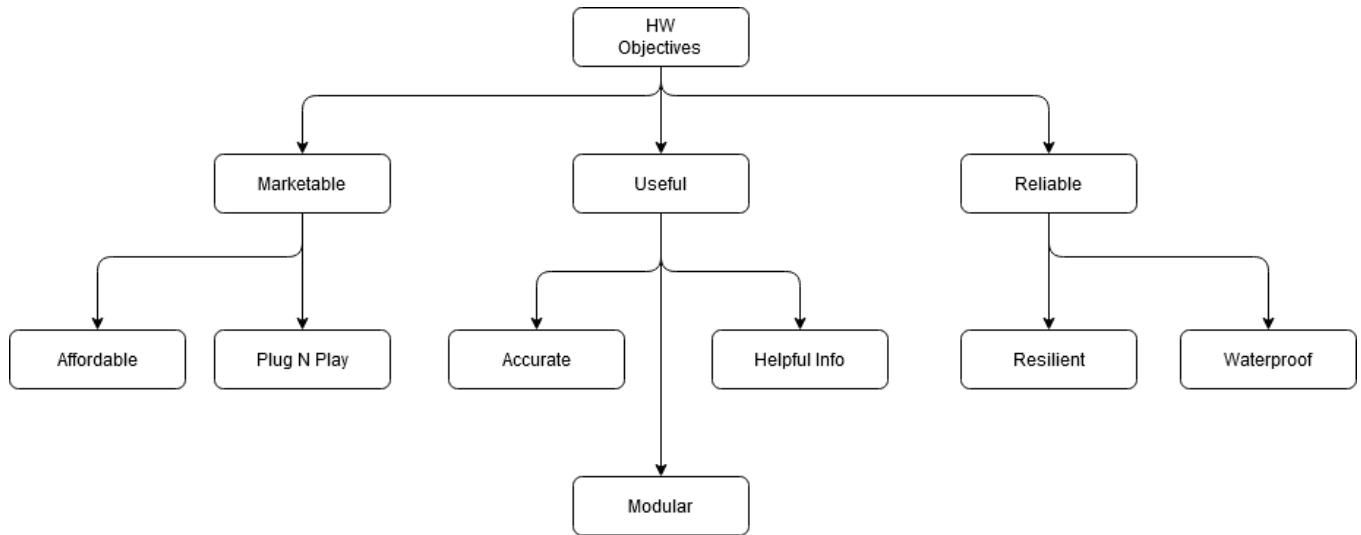


Figure 5: Hardware Design and Implementation Objective Tree for the design properties of being Marketable, Useful, and Reliable for the consumer.

As a consumer-based product, we define a qualitative functionality as the end goal: supporting users to care for their gardens and reasonably complete automated garden care when enabled. Aside from hardware sensor specifications, quantitative requirements are not realistic for this project, especially considering the time and resource limitations associated with the 2021 senior capstone framework. For example, assessing the effectiveness of the MGMS in improving garden yield would require long-term testing in a dedicated space, which would not be possible to complete following a semester timeline and difficult in a virtual collaborative environment.

Initial project requirements will be wholly qualitative outside of hardware requirements and will follow previously identified objectives. For the intended system demonstration in April 2020, system effectiveness will not be tested in lieu of testing for intended system functionality and correct hardware performance. With correct hardware performance, system functionality can be more easily tweaked in software to improve overall system performance once that testing occurs. Because of this, the described level of testing is acceptable for a simple demonstration in April.

Qualitative system attributes such as "ease of use" will be assessed separately during testing in a variety of ways. Timeline and resource permitting, surveys and qualitative analysis can be conducted . This testing will later be defined during the development process.

The intended system functionality is as follows:

- a. Promotes green spaces by lowering the learning curve of home lawn or garden care.
  - Real-time vital statistics
  - User configurable setup
  - Modular to mold to a variety of use-cases
- b. Solves the common problem of garden over-watering to conserves water
  - Control system to keep garden soil moisture at healthy levels
  - Predicts weather patterns and only automatically waters when needed

## 2.3 Methodology/Technical Approach

The definition of the product inherently makes the design an embedded system that requires multi-disciplinary knowledge and skills. Thus, we use the **strategy of design decomposition** to reduce the complexity of the problem to match each team member's expertise. Each hardware device (including sensors, controllers and actuators) will be set-up and tested individually during design prototyping, then combined into a complete system and tested afterwards. The UI software does not depend on hardware as much, so the front-end development is performed separately, while having tasks and deadlines in the same pace as the hardware development, such that the whole system can be defined and prototyped synchronously.

Using a strategy of design decomposition provides several benefits to the project development efforts, the biggest of which is acting as a “cushion” for possible issues that may arise during development and prototyping. Design decomposition means that each system component is evaluated separately, removing dependency on any one component for the final system function. This way, if an issue arises during development, a component or design can be adjusted without affecting the major development of the project as a whole. This is especially important because of the many different sensors and communication technologies being considered for the project. It is likely that sensor accuracy or communication performance may arise as an issue for individual components. Thanks to a design decomposition strategy, these issues will be able to be solved without much consequence.

A baseline system design is created to satisfy the desired project functionality and attributes. This design is shown in Figure 6 as a broken down view showing the three system components that were planned to be developed during the timeline of this project. Figure 7 shows a network topology view of the developed system.

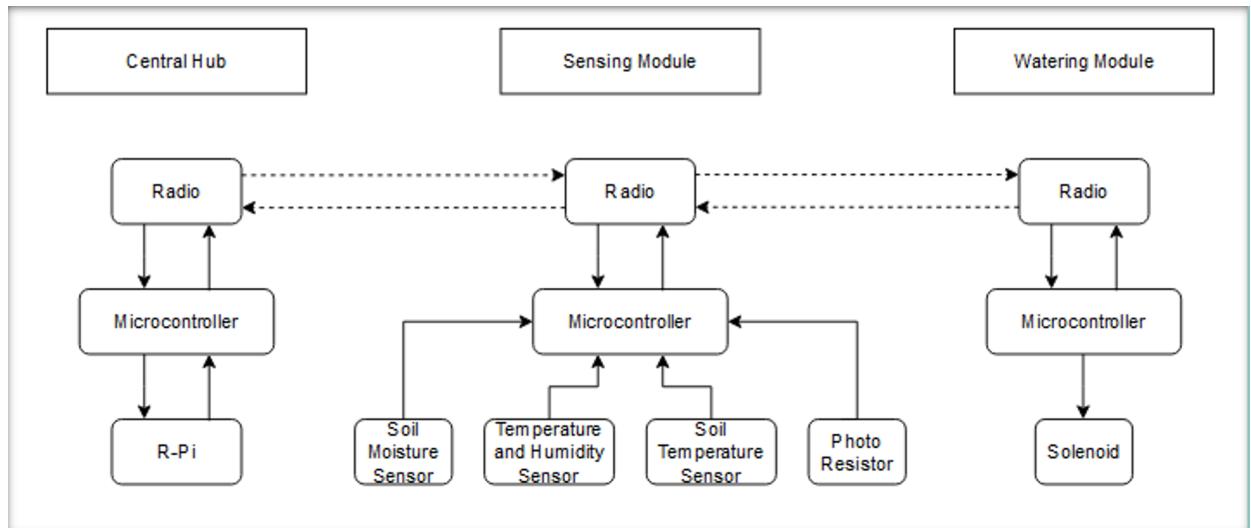


Figure 6: Overall System Design Overview of the prototype.

Diagram.jpg

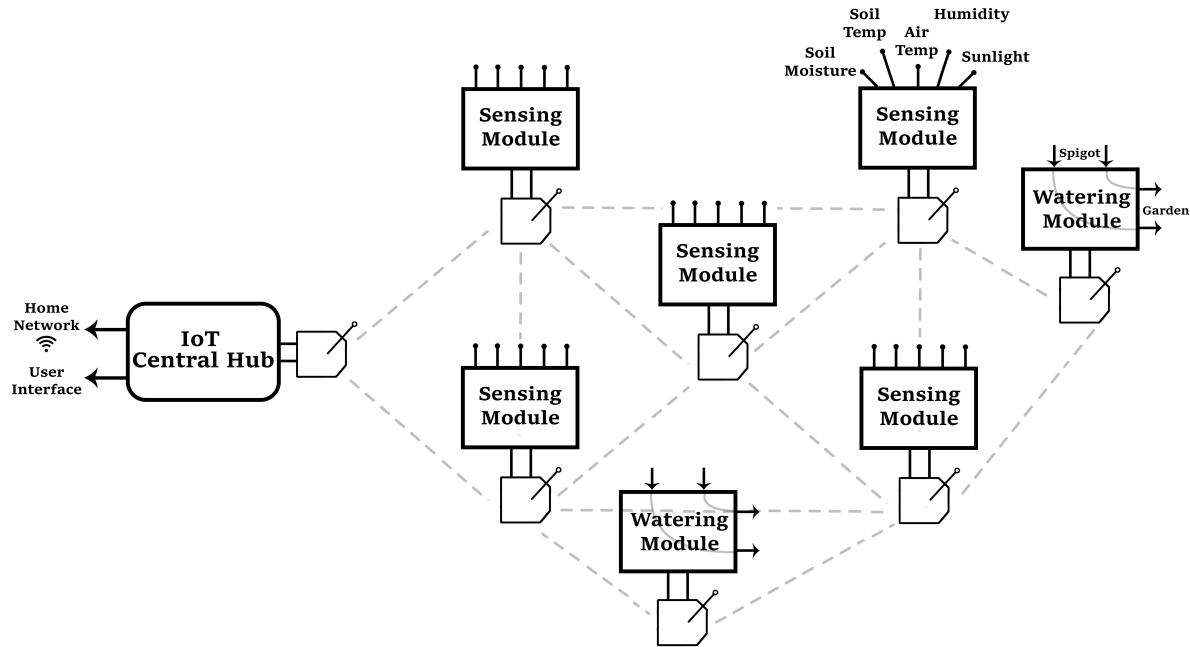


Figure 7: Network Topology view of the Completed System.

The MGMS system utilizes a modular design consisting of a central hub which will wirelessly connect to multiple sensing and watering modules that can be placed around a garden or house. The hub will host the central user interface and allow for customizing different garden setups. The hub software will make decisions based on the user configuration to control connected field modules in order to continuously monitor and water the garden. The user interface will be able to alert the user to garden events and make suggestions based on information available on the internet. The most important feature of the system should be modularity in freedom to interface a variety of system components in many different configurations.

To create a system following the presented design, tools such as a pairwise comparison chart, morphological chart, and decision tables were used to make design decisions in the definition and development phases of this project. The morphological chart, which was used to identify system components for prototyping and production is shown in Figure 8 below.

Morph Chart					
Func.   Means	1	2	3	4	5
Front-End Language	C	C++	Java	<b>Python</b>	-
Embedded Language	C	C++	Assembly	-	-
Wireless Protocol	Bluetooth	<b>Zigbee</b>	Wi-Fi	Cellular	-
Air Temp/Humid Sensor	SHT3X	<b>DHT11</b>	DHT12	-	-
Light Sensor	TSL2591	BH1750	TCS3472	<b>VEML7700</b>	<b>ALS-PT19</b>
User Interface	Touch Screen	<b>Local Website</b>	Windowed Application	-	-
Microcontroller Core	<b>AVR</b>	ARM	MSP	-	-

Figure 8: Morph Chart that shows various design decisions that were made throughout the creation of the MGMS prototype.

## 2.4 Standards

The development of the project conforms to various kinds of professional standards in the embedded system and IoT industry for the sake of security, readability and compatibility.

The product uses I2C bus and protocol for intra-board communication between devices, and uses Zigbee as inter-module wireless protocol. I2C (Inter-Integrated Circuit) is a synchronous serial communication bus invented by Philips Semiconductor (now NXP Semiconductors) [6] and widely used by current IC's in the market. Zigbee is a protocol developed by Zigbee Alliance based on IEEE-802.15.4 standard. IEEE-802.15.4 defines a two-layer architecture for low-data-rate wireless personal area networks (WPAN) [7], while Zigbee enhances it with two software layers [8]. Together they form a mature model to implement IoT concepts.

Additionally, electrical diagrams such as circuit schematic and PCB (printed circuit board) layout will be documented digitally in CAD (computer-aided design) software with standard rules and symbols built in. Common circuit diagram and PCB standards are specified in [9] and [10].

Finally, standards used in the software development, such as syntax and architecture, are based on specific dependencies, and they must be obeyed in order for the source code to successfully build or run. These standards are flexible in the development phase and will be chosen by the team members during project development.

## 2.5 Final System Design

Through the development process, a final system prototype and production design were created. The physical prototype created covered the following system components.

- **Central Hub**

- Wireless-to-Direct Serial Interface
- Raspberry Pi Platform

- **Sensor Hardware Module**

- Direct-Wireless Serial Interface
- Arduino Platform (ATMega328p)
- Hardware Sensor Interface
- Embedded C Software Enabling Full Module Functionality

- **Wireless Communication Framework**

- Serial Communication Datastream Standards
- 802.15.4 Firmware and Standard Configuration

- **User Interface**

- Python-Programmed Serial Interface for Raspberry Pi Connections
- Python-Programmed Graphical User Interface
- Data Visualizaion

Figure 9 shows a photograph of the completed sensor hardware module prototype. In the photograph, it is easy to identify the XBee wireless radio interface, soil sensor interface, and power system.

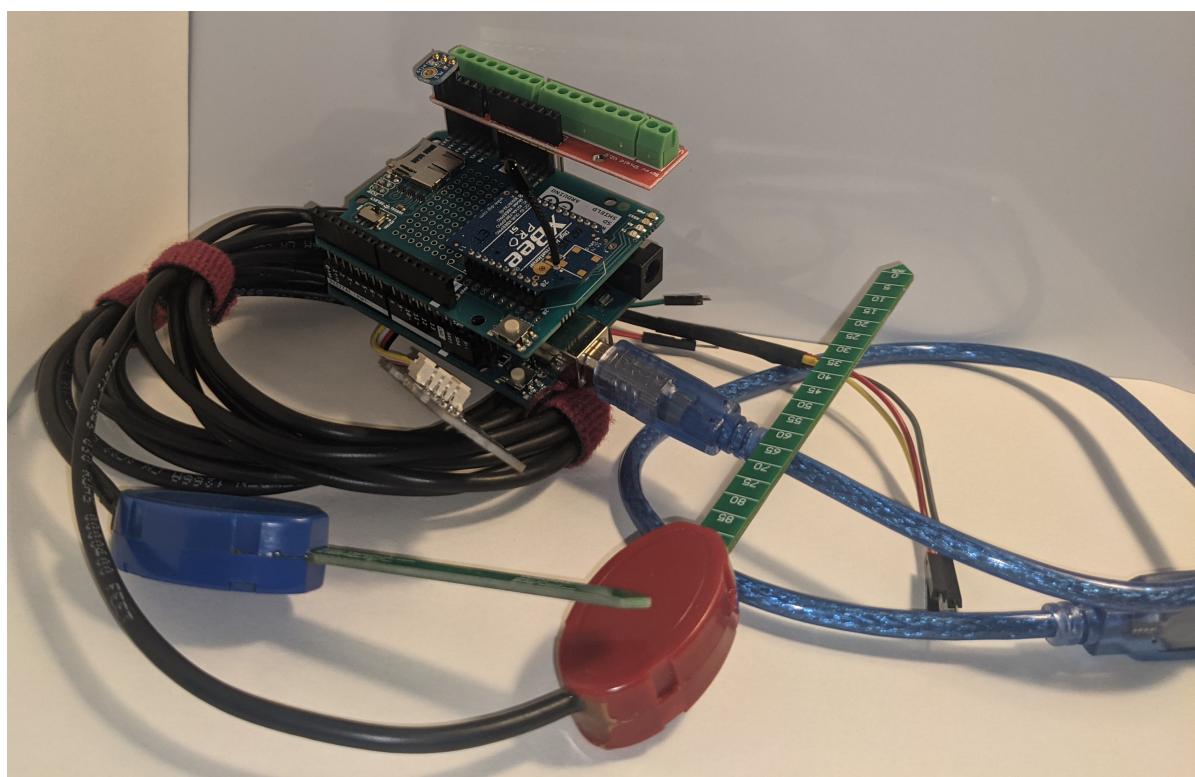


Figure 9: Fully Completed Prototype of the MGMS Hardware Sensor Module.

Complimenting the central hub hardware, a graphical user interface and historical data visualization system were developed. This system allows for user interaction with components interfaced within the wireless network and was partially used for hardware testing during performance evaluation. Figures 10 and 11 show the main UI menu and sample data chart respectively.

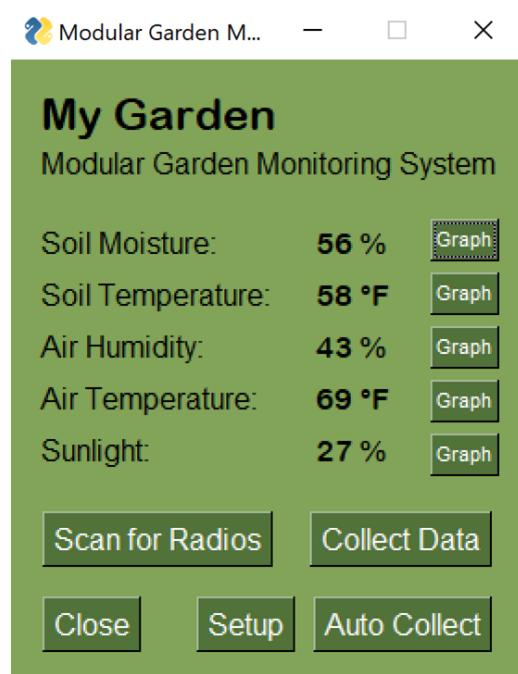


Figure 10: Initial GUI design

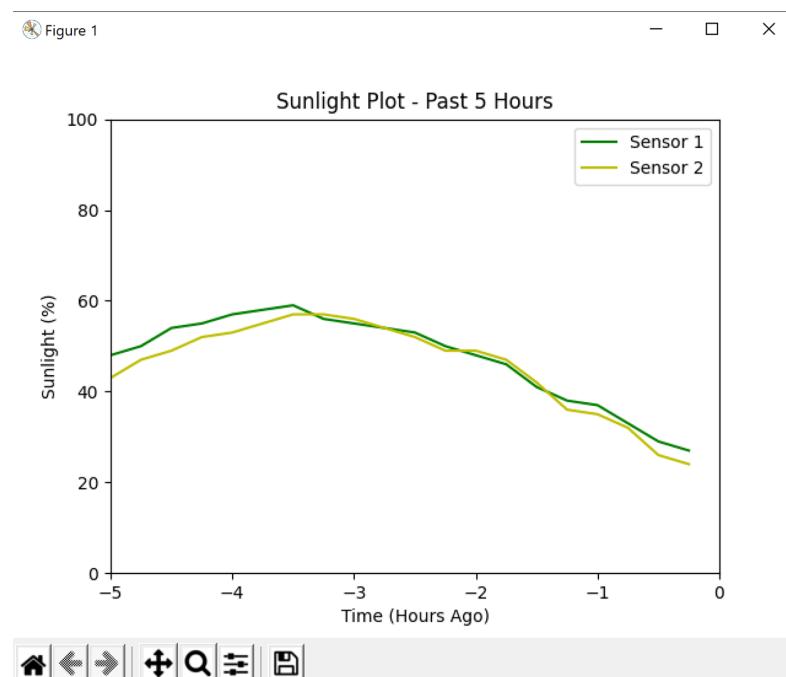


Figure 11: GUI Data Display in Graph Form

Once a working prototype was developed, steps were taken to design a preliminary production design for the MGMS hardware. Due to time and software tool constraints, only production hardware for the MGMS sensor module was designed. By following the developed prototype and decomposing the open-source arduino uno and XBee platforms, a master hardware schematic was created in Altium Designer and is shown in Figure 12. This schematic is then used to design a printed circuit board design to support the system hardware components. This PCB is shown in Figure 13.

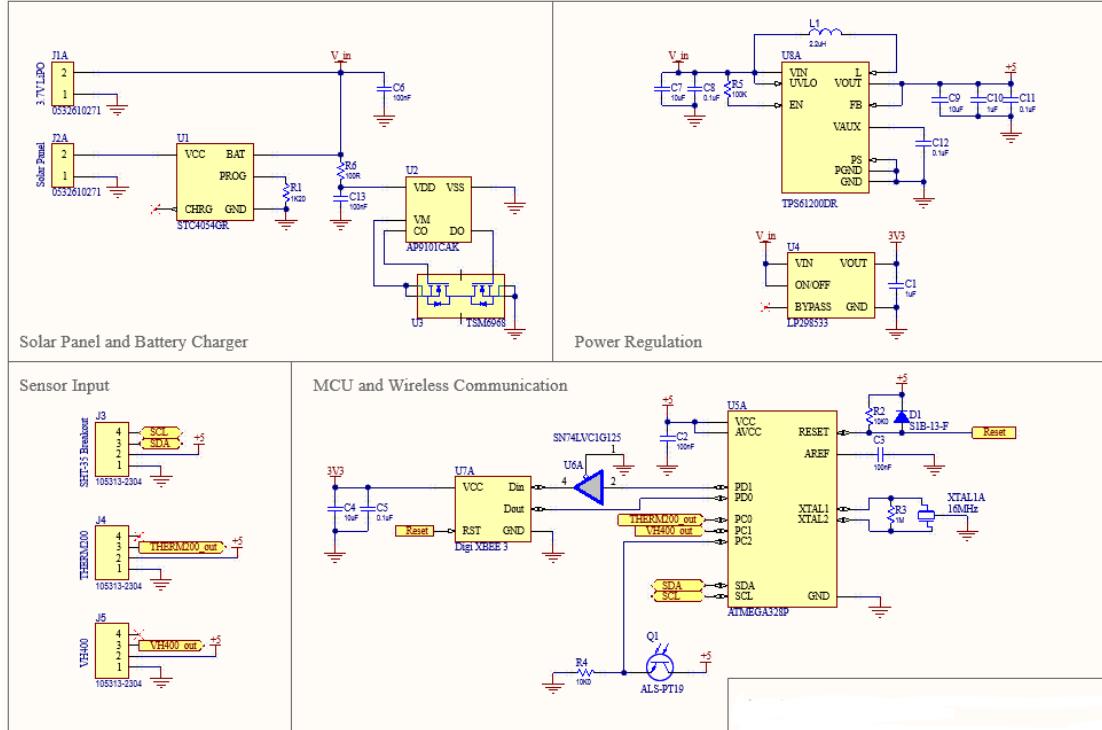


Figure 12: Hardware Sensor Module Schematic.

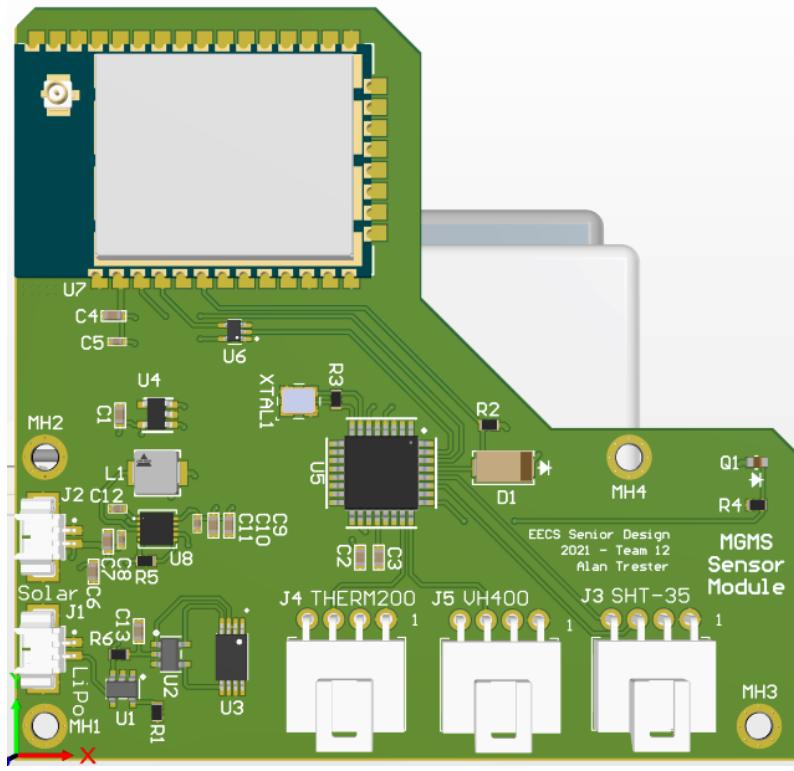


Figure 13: Printed Circuit Board Model for the Hardware Sensor Module.

The printed circuit board was designed in such a way that it could snugly fit into a standard transparent weatherproof container, interfaced with a standard sized solar panel and lithium-ion battery pack. The system's sunlight sensor and radio antenna are cleverly positioned as to not be obstructed by other system components. A final 3d mockup of the entire hardware stack is shown in Figure 14

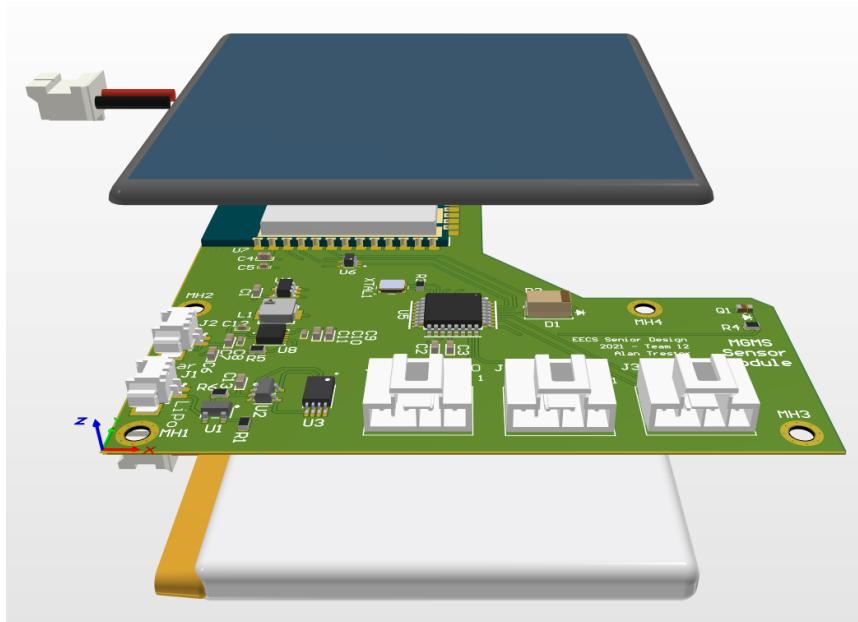


Figure 14: Stack Mockup for the Hardware Sensor Module.

## 2.6 Testing and System Performance

The process of testing and confirming performance of the system's hardware components occurred subsequently with the prototype development. As each individual sensor was interfaced, calibration and speed testing occurred so that embedded software could be developed accordingly. Throughout this process, sensor accuracy and the system response time was confirmed to perform within and exceeding team expectations. Figures 15, ??, and 17 show data used for confirming system performance of the MGMS sensor hardware module.

Soil Moisture Sensor Performance					
Uncalibrated			Calibrated		
Actual Moisture	Reported Moisture	Error	Actual Moisture	Reported Moisture	Error
0.0	0.04	4.00%	0.0	0.00	0.00%
0.1	0.15	5.00%	0.1	0.09	-1.00%
0.2	0.24	4.00%	0.2	0.21	1.00%
0.3	0.33	3.00%	0.3	0.30	0.00%
0.4	0.41	1.00%	0.4	0.38	-2.00%
0.5	0.52	2.00%	0.5	0.51	1.00%
0.6	0.61	1.00%	0.6	0.60	0.00%
0.7	0.69	-1.00%	0.7	0.69	-1.00%

Figure 15: Soil moisture data reported by the VH400 sensor during calibration testing.

Soil Temperature Sensor Performance					
Uncalibrated			Calibrated		
Actual Temp	Reported Temp	Error	Actual Temp	Reported Temp	Error
83.9	85.6	2.03%	81.4	81.7	0.37%
76.2	76.1	-0.13%	75.8	76.0	0.26%
69.0	68.8	-0.29%	72.2	72.3	0.14%
62.4	62.4	0.00%	60.8	60.7	-0.16%
49.2	48.6	-1.22%	48.8	48.6	-0.41%
43.7	43.1	-1.37%	44.6	44.6	0.00%
41.8	40.8	-2.39%	39.3	39.1	-0.51%
40.0	38.4	-4.00%	38.9	38.8	-0.26%

\*all temperatures recorded in °F

Figure 16: Soil temperature data reported by the THERM200 sensor during calibration testing.

Air Temperature Sensor Performance					
Uncalibrated			Calibrated		
Actual Temp	Reported Temp	Error	Actual Temp	Reported Temp	Error
78.3	77.9	-0.51%	78.9	78.7	-0.25%
71.6	70.7	-1.26%	72.4	72.3	-0.14%
68.5	67.8	-1.02%	70.2	70.0	-0.28%
67.7	67.1	-0.89%	69.1	69.1	0.00%
38.2	38.4	0.52%	37.4	37.6	0.53%
37.6	37.9	0.80%	36.9	40.0	8.40%
31.0	31.1	0.32%	31.5	31.5	0.00%
31.3	31.1	-0.64%	30.9	31.1	0.65%

\*all temperatures recorded in °F

Figure 17: Air temperature data reported by the SHT35 sensor during calibration testing.

When designing the software and GUI, like the hardware components, a test driven development method was approached. Tests for the GUI and software were developed and ensured all front-end functionality would fit software capabilities. Figure 18 illustrates key features whose performance was evaluated during the front-end development process.

Test Case/Description	UI Element	PASS/FAIL
Key Feature - GUI must be able to scan for radios when prompted – must be able to be accessed while in auto collect mode	Scan <u>For</u> Radios Button	PASS
Key Feature – GUI must be able to plot accurate data for all sensors when prompted – must be able to be accessed in auto collect mode	Graphing Features	PASS
Key Feature – GUI must be able to get the current sensor data – must be able to be accessed while in auto collect mode	Pull Current Data	PASS
Key Feature – Sets the collection time element	Setup Button	PASS
Key Feature – GUI must be able to pull sensor data automatically given a time element	Auto Collect Data	PASS
Key Feature – Ensure GUI is not inaccessible when in auto collect mode	Auto Collect and Using Other Features	PASS
No Radios Connected – Tests that the GUI reacts correctly when no radios are connected. Pop up shows response to connect radios	Scan <u>For</u> Radios Button	PASS
No Coordinator Connected – Tests that the GUI reacts correctly when no radios are connected.	Scan <u>For</u> Radios Button	PASS
Ensures that the close button closes the GUI and stops the auto collection process	Close Button	PASS

Figure 18: GUI Test Case Chart shows what key features were desired from the software and how those features were tested and expected to perform.

## 2.7 Budget

The expected project budget changed over time as decisions in design and development processes were made. A finalized budget is shown below as a Bill of Materials (BoM) for the system prototype in Figure 19. This bill of materials shows costs for specific prototyping components and not necessarily the tools or work hours used to create the system. An estimated final system cost is provided based on information available from the computer aided design software and printed circuit board retailers.

Final Prototyping BoM					
Description	Mfr. Part Number	Unit Price	Quantity	Retailer	Notes
<b>Soil Moisture Sensor</b>	VH400	\$39.99	1	Vegetronix	3 meter cable
<b>Soil Temp Sensor</b>	THERM200	\$33.99	1	Vegetronix	1.5 meter cable
<b>Temp/Humidity Sensor</b>	SHT35	\$14.99	1	Adafruit	Breakout board
<b>Analog Light Sensor</b>	ALS-PT19	\$7.95	1	Digikey	Chosen over digital sensor
<b>Radio Module</b>	XBP24-AWI-001J	\$19.95	3	Digikey	802.15.4 S1 Pro
<b>Arduino Uno</b>	ATMega328p	\$22.15	3	Mouser	
<b>XBEE-Arduino Interface</b>	B006RATC2E Wireless SD Shield	\$14.99	3	Amazon	Discontinued
<b>Total</b> (Not including prototyping tools or misc. components)					<b>\$268.19</b>
<b>Estimated Production Cost</b> (Altium BoM and PCB retailors)					<b>\$130</b>

Figure 19: Final prototyping Bill of Materials. Includes an estimated final system production cost

## 2.8 Timeline



Throughout the course of the project, it was estimated that each individual person on the team would work approximately 7-10 hours per week. Some weeks will be lighter on work (waiting for parts to be shipped), while others will be more labor intensive (assembly and programming), but overall the estimate of 7-10 hours per week is a fair number. Below is a rough chart documenting the overall time spent working on the project, which supplements the information shown in the Gantt Chart in the previous section.

	<b>Alan Trester</b>	<b>Eric Krenz</b>	<b>Zuguang Liu</b>	<b>Sadie Gladden</b>
<b>Total Hours Planned</b>	310	310	310	310
<b>Total Hours Worked</b>	305	280	295	285
<b>Class and Documentation Time</b>	55	60	60	50
<b>Design and Planning</b>	110	100	100	95
<b>Dev and Prototyping</b>	145	120	135	140
<b>Average per week</b>	9.5	8.75	9.21	8.91

Figure 20: Time Chart showing the allocation of past time spent and future time expected on the project.

Just like the Gantt Chart, this table was continually updated as the project progressed.

## 2.9 Problems Encountered and Solutions

The COVID-19 pandemic in 2020 and 2021 changed many of the ways that world operates. Just as the global pandemic was unprecedented, many of the changes that students at the University of Cincinnati faced have also been unprecedented. Many new challenges, on top of the ones that are typically faced arose during this academic year which changed that way that this senior design project was approached.

The biggest challenge for the team as a result of the global pandemic was the requirement for virtual collaboration as opposed to in-person meetings. Working and collaborating virtually for class sessions as well as group work often hindered progress and communication since instant feedback was not always available as it may have been otherwise. Typically, staying updated on class assignments, keeping track of team progress, and having multiple people collaborate on one project component were uniquely difficult processes. The team adjusted to the new reality in several ways, the most important of which was perhaps by taking advantage of new collaboration and planning tools to complete documentation and project tasks. Tools such as Microsoft Teams, Overleaf, and GIT were leveraged much more heavily than they would have otherwise. Additionally, as a result of the virtual environment, the team did not have access to EECS collaboration spaces and electronic labs, instead working from home offices without the tools, equipment, and real-time team input that otherwise would have been available. Instead, our home offices had to be turned into personal lab spaces by acquiring equipment such as oscilloscopes, multimeters, and 3D printers to use for project development. Lastly, project funding, which is typically provided by EECS, was made unavailable, which further hindered development efforts.

One typical challenge that arose for our team was development with outdated and obsolete technology. Due to personal budget constraints, several system components were chosen due to their existing availability to use in project development. In particular, the XBEE S1 Pro radios which were used to build the MGMS's wireless communication system used an outdated IEEE 802.15.4 firmware that has been replaced in the 2nd, 3rd, and 4th generation XBEE radios with Zigbee and Digimesh features. Working with an outdated technology limited certain system functionalities that would have proved useful for system development. Furthermore, because we are designing the system in such a way that it can be wholly produced, we had to cleverly develop the system prototype in such a way that functionality would be forward-compatible with the newer technology that would likely be used.

## 2.10 Future Recommendations

In the future, there are many possibilities and opportunities to improve upon this prototype. Of course, this is a baseline proof-of-concept system which can be ever-improved with more developed features on top of the system platforms that have been created. Although the platform was designed in a way that could easily accommodate these features, time constraints prevented more sophisticated user interaction and internet connectivity features from being fully realized. Time and equipment constraints also prevented the team from wholly testing the completed prototype. Although the system functioned as expected, it was impossible to measure the hardware and software limits to be documented for further development. Had testing been planned well in advance, this information may have been more attainable and used to create a better final product. The final system can be expanded to be significantly more impressive through designing simple components such as solenoid valve units for watering. It is arguable that it would be possible to create a more well-rounded demonstrable product if less time was spent on the hardware development of the main sensor module.

### **3 Conclusion**

In completing our senior design project we were able to successfully design, prototype, and test the proof-of-concept garden monitoring system to the expectations laid out in the initial design plans. The presented MGMS prototype demonstrates the core system functionality: a thoroughly developed and easy to interface modular wireless communication system, accurate environmental readings, efficient communication, and a robust user interface platform. The time and effort spent developing our core system components means that the development of new features and hardware can be streamlined. With time, a full, marketable, consumer-grade system can be built that effectively promotes green spaces and addresses the water conservation concerns that are brought up at the beginning of this report, filling a very important market space. Considering the previously discussed challenges, solutions, and proposed improvements, our team has a very optimistic outlook on the future of the MGMS system if it were to be further developed.

## References

- [1] N. A. of Landscape Professionals, “New research confirms americans still value lawns and green spaces.” Available at <https://www.businesswire.com/news/home/20190401005679/en/New-Research-Confirms-Americans-Lawns-Green-Spaces>.
- [2] E. P. Agency, “Outdoor water use in the united states.” Available at <https://19january2017snapshot.epa.gov/www3/watersense/pubs/outdoor.html>.
- [3] E. P. Agency, “Drought and watersense.” Available at <https://www.epa.gov/watersense/drought-watersense>.
- [4] Onset, “Field monitoring system onset data loggers.” Available at <https://www.onsetcomp.com/hobonet>.
- [5] Edyn, “Edyn: Welcome to the connected garden.” Available at <https://www.kickstarter.com/projects/edyn/edyn>Welcome-to-the-connected-garden/description>.
- [6] “UM10204 i2c-bus specification and user manual,” vol. 2014, p. 64.
- [7] “IEEE standard for low-rate wireless networks,” pp. 1–709. Conference Name: IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011).
- [8] Z. Alliance, “ZigBee specification.”
- [9] “IEEE standard american national standard canadian standard graphic symbols for electrical and electronics diagrams (including reference designation letters),” pp. i–244. Conference Name: IEEE Std 315-1975 (Reaffirmed 1993).
- [10] “IEEE approved draft standard for electrical characterization of printed circuit board and related interconnects at frequencies up to 50ghz.,” pp. 1–150. Conference Name: P370/D8, July 2020.

## 4 Appendices

### 4.1 Python Implementation of GUI

```
1 #! /usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 # mgms.py
5 # Main module for GUI front-end
6 # Sadie Gladden <gladdesm@mail.uc.edu>
7 # Eric Krenz <krenzew@mail.uc.edu>
8
9
10 import time
11 import traceback
12 import threading
13 from pathlib import Path
14 import matplotlib.pyplot as plt
15 import serial
16 import PySimpleGUI as sg
17
18
19
20 RADIO_LIST = []
21 RADIO_DATA_PULL = []
22 PARSED_RADIO_DATA = []
23 SAMPLE_RATE = 120
24
25
26 # TODO: radio struct? updated with get radios that contains radios name,
27 #        location, DH, DL, etc.
28 class Radio:
29     def __init__(self, MY, SH, SL, DB, NI):
30         self.MY = MY
31         self.SH = SH
32         self.SL = SL
33         self.DB = DB
34         self.NI = NI
35         self.filename = str(self.NI, 'UTF-8')
36
37     def print_data(self):
38         print(
39             "My Radio Info:\nMY: {}\nSH: {}\nSL: {}\nDB: {}\nNI: {}".format(
40                 self.MY, self.SH, self.SL, self.DB, self.NI))
41         return
42
43
44 def compare_radios(a, b):
45     if a.MY == b.MY and a.SH == b.SH and a.SL == b.SL and a.DB == b.DB \
46         and a.NI == b.NI:
47         return 1
48     return 0
49
50
51 def parse_radio_ids(radio_ids):
52     temp_radio_list = []
53     radio_ids = radio_ids.replace(b'OK\r', b'')
54     radio_ids = radio_ids.replace(b'EEww\r', b'')
55     temp = radio_ids.split(b'\r\n')
56     for x in range(len(temp) - 1):
57         radio = temp[x].split(b'\r')
58         if len(radio) == 5:
59             temp_radio = Radio(radio[0], radio[1], radio[2], radio[3],
60                                 radio[4])
61             temp_radio_list.append(temp_radio)
62         else:
63             msg = 'weird radio data'
64     return temp_radio_list
65
66
67 def get_sample_rate():
68     return SAMPLE_RATE
69
70
71 def clear_historical_data():
72     return
73
```

```

74
75 def compare_radio_list(temp_radio_list):
76     new_radio = 0
77     if len(RADIO_LIST) == 0 or len(RADIO_LIST) > len(temp_radio_list):
78         RADIO_LIST.clear()
79         for each in temp_radio_list:
80             RADIO_LIST.append(each)
81             new_radio = new_radio + 1
82     else:
83         for each in temp_radio_list:
84             contains = 0
85             for x in range(len(RADIO_LIST)):
86                 if compare_radios(each, RADIO_LIST[x]) == 1:
87                     contains = contains + 1
88             if contains == 0:
89                 RADIO_LIST.append(each)
90                 new_radio = new_radio + 1
91
92     return new_radio
93
94 def get_radio_names():
95     name = []
96     if len(RADIO_LIST) == 0:
97         return name
98     else:
99         for each in RADIO_LIST:
100            name.append(str(each.NI, 'UTF-8'))
101
102     return name
103
104 def set_sampling_freq(input):
105     # TODO: Link these to buttons or something or make GUI to set?
106     SAMPLE_RATE = input
107
108     return
109
110 def scan_for_radios(serial_port):
111     ser = serial.Serial(serial_port, timeout=3)
112
113     ser.write(b'\r')
114     time.sleep(1.5)
115     # enables command mode (1.5 sec delay is necessary)
116     ser.write(b'+++')
117     time.sleep(1.5)
118     # print('Initiate command mode waiting for OK')
119     line = ser.read_until(b'\r', size=None)
120
121     # command mode calls
122     ser.write(b'ATND\r') # Node Discovery
123     time.sleep(1.5)
124
125     # response
126     # print('trying to get radio list')
127     response = ser.read_until(b'\r\r\r')
128     #print('data from radio {}'.format(response))
129     radio_data = response
130
131     # closes command mode
132     ser.write(b'ATCN\r')
133     time.sleep(1.5)
134     # print('Command ATCN mode waiting for OK')
135     line = ser.read_until(b'\r', size=None)
136
137     # closes serial port
138     ser.close()
139
140     # return radio data to main function
141     return radio_data
142
143
144 def sample_radios(serial_port):
145     # List for storing radio data (to Return)
146     RADIO_DATA_PULL.clear()
147     if len(RADIO_LIST) > 1:
148         for each in RADIO_LIST:
149             # opens serial port
150             ser = serial.Serial(serial_port)
151
152             # enables command mode (1.5 sec delay is necessary)
153             ser.write(b'\r\r')
154             time.sleep(1.5)
155             ser.write(b'+++')
156             time.sleep(1.5)

```

```

157     # print('Initiate command mode waiting for OK')
158     line = ser.read_until(b'\r', size=None)
159     # command mode calls
160     DH = b'ATDH 00' + each.SH + b'\r'
161     ser.write(DH) # Correct DH testing
162     time.sleep(1.5)
163     # print('waiting for OK')
164     line = ser.read_until(b'\r', size=None)
165
166     DL = b'ATDL ' + each.SL + b'\r'
167     ser.write(DL) # Correct DL testing
168     time.sleep(1.5)
169     # print('waiting for OK')
170     line = ser.read_until(b'\r', size=None)
171
172     # closes command mode
173     ser.write(b'ATCN\r')
174     time.sleep(1.5)
175     # print('Command ATCN mode waiting for OK')
176     line = ser.read_until(b'\r', size=None)
177
178     # requests data
179     ser.write(b'\r\r')
180     time.sleep(1.5)
181     ser.write(b'Request Info\r')
182     time.sleep(1.5)
183     # line = ser.readline()
184     line = ser.read_until(b'\r\r', size=None)
185     radio_data = line
186     temp_arr = []
187     temp_arr.append(each.NI)
188     temp_arr.append(radio_data)
189     RADIO_DATA_PULL.append(temp_arr)
190
191     # closes serial port
192     ser.close()
193     # return radio data to main function
194     # NEED RADIO WITH DUMMY DATA FROM ALAN
195 else:
196     radio_data = b'No Radios Set Up\r'
197     RADIO_DATA_PULL.append(radio_data)
198 return RADIO_DATA_PULL
199
200
201 def parse_radio_data(data):
202     # return list of parsed data to main function
203     PARSED_RADIO_DATA.clear()
204     for each in data:
205         temp_arr = []
206         temp_arr.append(str(each[0], 'UTF-8'))
207         temp = each[1].replace(b'OK\r', b'')
208         temp = temp.replace(b'EEww\r', b'')
209         temp = temp.replace(b'\r\r', b'')
210         value = str(temp, 'UTF-8')
211         temp_arr.append(value)
212         PARSED_RADIO_DATA.append(temp_arr)
213 return PARSED_RADIO_DATA
214
215
216 def background_collection(delay, stop, task):
217     next_time = time.time() + delay
218     while True:
219         if stop():
220             break
221         time.sleep(max(0, next_time - time.time()))
222         try:
223             task()
224         except Exception:
225             traceback.print_exc()
226             # in production code you might want to have this instead of course:
227             # logger.exception("Problem while executing repetitive task.")
228             # skip tasks if we are behind schedule:
229         next_time += (time.time() - next_time) // delay * delay + delay
230
231
232 def test_background():
233     #todo
234     print("sampling time:", time.asctime(time.localtime(time.time())))
235     data = sample_radios('COM4')
236     names = get_radio_names()
237     parsed_radio_data = parse_radio_data(data)
238     store_parsed_data(parsed_radio_data)
239

```

```

240
241 def store_parsed_data(data):
242     #TODO
243     return
244
245
246 def clear_radio_list():
247     RADIO_LIST.clear()
248     return
249
250
251 def parse_data(sensor_data):
252     # return list of parsed data to main function
253     return sensor_data.split(',')
254
255
256 def show_gui(parsed_data, sensor1_data, sensor2_data, thread):
257     # green theme for green project
258     sg.theme('Green')
259
260     # Title and description shown at top of GUI
261     description = sg.Column([
262         sg.Text(f'My Garden',
263             font=('Arial Rounded MT Bold', 18),
264             pad=((10, 0), (10, 0))),
265     ],
266     [
267         [
268             sg.Text(f'Modular Garden Monitoring System',
269                 font=('Arial', 12),
270                 pad=((10, 10), (0, 10)))
271         ],
272         key='COL1'
273     ]
274
275     # Data type labels shown in the Left Body of the GUI
276     data_types = sg.Column(
277         [[sg.Text(f'Soil Moisture:', font=('Arial', 12))],
278          [sg.Text(f'Soil Temperature:', font=('Arial', 12))],
279          [sg.Text(f'Air Humidity:', font=('Arial', 12))],
280          [sg.Text(f'Air Temperature:', font=('Arial', 12))],
281          [sg.Text(f'Sunlight:', font=('Arial', 12))]],
282         element_justification='left',
283         key='COL2')
284
285     # Current sensor values shown in the Right Body of the GUI
286     sensor_data = sg.Column(
287         [[sg.Text(f'{parsed_data[0]} %', font=('Arial Rounded MT Bold', 12)),
288          [sg.Text(f'{parsed_data[1]}'+u'\N{DEGREE SIGN}F', font=('Arial Rounded MT Bold', 12))],
289          [sg.Text(f'{parsed_data[2]} %', font=('Arial Rounded MT Bold', 12))],
290          [sg.Text(f'{parsed_data[3]}'+u'\N{DEGREE SIGN}F', font=('Arial Rounded MT Bold', 12))],
291          [sg.Text(f'{parsed_data[4]} %', font=('Arial Rounded MT Bold', 12))]],
292         element_justification='left',
293         key='COL3')
294
295     # Buttons for pop-up windows that include graphs
296     data_buttons = sg.Column(
297         [[sg.Button('Graph', font=('Arial', 8), key='soil_m')],
298          [sg.Button('Graph', font=('Arial', 8), key='soil_t')],
299          [sg.Button('Graph', font=('Arial', 8), key='air_m')],
300          [sg.Button('Graph', font=('Arial', 8), key='air_t')],
301          [sg.Button('Graph', font=('Arial', 8), key='sun')]],
302         element_justification='left',
303         key='COL4')
304
305     # Buttons to close out of gui
306     button_ok = sg.Column(
307         [[sg.Button('Close', font=('Arial', 12), pad=((0, 20), (0, 10)))]],
308         element_justification='right',
309         key='COL7')
310
311     # Button Clears Radios...
312     button_clear = sg.Column([
313         sg.Button(
314             'Clear', font=('Arial', 12), pad=((0, 0), (10, 10)), key='clear')
315     ],
316         element_justification='left',
317         key='COL8')
318
319     # Button to scan for radios
320     button_scan = sg.Column([
321         sg.Button('Scan for Radios',
322             font=('Arial', 12),
323             pad=((0, 10), (10, 10)),
324             key='scan')
325
326

```

```

323     ],
324             element_justification='left',
325             key='COL5')
326
327 # Button to collect data
328 button_collect = sg.Column([
329     sg.Button('Collect Data',
330             font=('Arial', 12),
331             pad=((0, 0), (10, 10)),
332             key='collect')
333 ],
334             element_justification='right',
335             key='COL6')
336
337 button_setup = sg.Column([
338     sg.Button(
339         'Setup', font=('Arial', 12), pad=((0, 0), (0, 10)), key='setup')
340 ],
341             element_justification='right',
342             key='COL9')
343
344 button_auto_collect = sg.Column([
345     sg.Button('Auto Collect',
346             font=('Arial', 12),
347             pad=((0, 0), (0, 10)),
348             key='auto_collect')
349 ],
350             element_justification='right',
351             key='COL10')
352
353 # everything that shows up in the GUI
354 layout = [[description], [data_types, sensor_data, data_buttons],
355             [button_scan, button_collect],
356             [button_ok, button_setup, button_auto_collect]]
357
358 # create and open window
359 window = sg.Window(layout=layout,
360                     title='Modular Garden Monitoring System',
361                     margins=(0, 0),
362                     finalize=True,
363                     element_justification='center',
364                     no_titlebar=False,
365                     grab_anywhere=True)
366
367 # loop keeps window open, executes events, reads values
368 while True:
369     event, values = window.read()
370
371 # if user closes window or clicks close button
372 if event == sg.WIN_CLOSED or event == 'Close':
373     break
374
375 # Popup for scanning for radios
376 if event == 'scan':
377     prev_num_radios = len(RADIO_LIST)
378     try:
379         test_radio_list = scan_for_radios(
380             'COM4') # using windows - /dev/ttyUSB0 for linux
381         radios = parse_radio_ids(
382             test_radio_list
383         ) # takes scanned radios and creates radio objects
384         num_radios = compare_radio_list(
385             radios
386         ) # Compares list of radio objects to MASTER RADIO LIST
387         names = get_radio_names()
388         if len(radios) == 0:
389             sg.Popup(
390                 'Successfully Scanned for Radios.',
391                 f'{len(RADIO_LIST)} Current Radios.\n',
392                 'No New Radios Found',
393                 'Please Ensure Your Radio Is In Range And Powered On',
394                 title='Scan for Radios')
395         elif prev_num_radios > len(radios):
396             lost_radios = prev_num_radios - num_radios
397             sg.Popup('Successfully Scanned for Radios.',
398                     f'Found {num_radios} radios.',
399                     f'Number of Radios Lost: '
400                     f'{lost_radios}',
401                     f'Current Radios: {names}',
402                     title='Scan for Radios')
403         elif prev_num_radios < len(radios):
404             total_radios = prev_num_radios + num_radios
405             sg.Popup('Successfully Scanned for Radios.',

```

```

406                         f'Found {total_radios} radios.',
407                         f'Number of New Radios: '
408                         f'{num_radios}',
409                         f'Current Radios: {names}',
410                         title='Scan for Radios')
411
412         else:
413             sg.Popup(
414                 'Successfully Scanned for Radios.',
415                 f'{len(RADIO_LIST)} Current Radios.\n'
416                 'No New Radios Found',
417                 f'Current Radios: {names}',
418                 title='Scan for Radios')
419
420     except:
421         sg.Popup('Error No Coordinator Connected',
422                 title='Scan for Radios')
423
424 # Popup for collect current data
425 if event == 'collect':
426     if len(RADIO_LIST) == 0:
427         sg.Popup('No Connected Radios.\nPlease Scan for Radios',
428                 title='Current Data')
429     else:
430         data = sample_radios('COM4')
431         names = get_radio_names()
432         parsed_radio_data = parse_radio_data(data)
433         sg.Popup(
434             f'Successfully Collected Current Data From ',
435             f'{len(RADIO_LIST)} Radios.',
436             f'Radio Names: {names}',
437             f'Raw Radio Data: {parsed_radio_data}',
438             title='Current Data')
439
440 # Popup for clear radio list internal use only
441 if event == 'clear':
442     clear_radio_list()
443
444 # Popup to set collection time
445 if event == 'setup':
446     sample_freq = sg.popup_get_text(
447         'Please enter a sampling frequency in minutes', title='Setup')
448     set_sampling_freq(sample_freq)
449
450 # Sets to start auto collecting data
451 if event == 'auto_collect':
452     if len(RADIO_LIST) == 0:
453         sg.Popup('No Connected Radios.\nPlease Scan for Radios',
454                 title='Auto Collect Data')
455     else:
456         if not thread.is_alive():
457             thread.start()
458             sg.Popup(f'Collecting Data Every {SAMPLE_RATE} Seconds',
459                     title='Auto Collect Data')
460         else:
461             sg.Popup('Auto Collection Error',
462                     title='Auto Collect Data')
463
464 # Popup for Soil Moisture
465 if event == 'soil_m':
466     plt.close('all')
467     plot_sensor1 = []
468     plot_sensor2 = []
469     hours_ago = [
470         -5.00, -4.75, -4.5, -4.25, -4.00, -3.75, -3.5, -3.25, -3.00,
471         -2.75, -2.5, -2.25, -2.00, -1.75, -1.5, -1.25, -1.00, -0.75,
472         -0.5, -0.25
473     ]
474     for i in range(20):
475         plot_sensor1.append(sensor1_data[i][1])
476         plot_sensor2.append(sensor2_data[i][1])
477     plt.plot(hours_ago, plot_sensor1, 'g', label="Sensor 1")
478     plt.plot(hours_ago, plot_sensor2, 'y', label="Sensor 2")
479     plt.title('Soil Moisture Plot - Past 5 Hours')
480     plt.xlabel('Time (Hours Ago)')
481     plt.ylabel('Soil Moisture (%)')
482     plt.xlim([-5, 0])
483     plt.ylim([0, 100])
484     plt.legend()
485     plt.show(block=False)
486     # popup_text = '[insert plot w/ matplotlib here]'
487     # TODO add charts to popups in GUI
488     # sg.Popup('This is a pop-up for Soil Moisture',
489     #          popup_text, title='Soil Moisture')

```

```

489 # Popup for Soil Temperature
490 if event == 'soil_t':
491     plt.close('all')
492     plot_sensor1 = []
493     plot_sensor2 = []
494     hours_ago = [
495         -5.00, -4.75, -4.5, -4.25, -4.00, -3.75, -3.5, -3.25, -3.00,
496         -2.75, -2.5, -2.25, -2.00, -1.75, -1.5, -1.25, -1.00, -0.75,
497         -0.5, -0.25
498     ]
499     for i in range(20):
500         plot_sensor1.append(sensor1_data[i][0])
501         plot_sensor2.append(sensor2_data[i][0])
502     plt.plot(hours_ago, plot_sensor1, 'g', label="Sensor 1")
503     plt.plot(hours_ago, plot_sensor2, 'y', label="Sensor 2")
504     plt.title('Soil Temperature Plot - Past 5 Hours')
505     plt.xlabel('Time (Hours Ago)')
506     plt.ylabel(u'Soil Temperature (\N{DEGREE SIGN}F)')
507     plt.xlim([-5, 0])
508     plt.ylim([0, 100])
509     plt.legend()
510     plt.show(block=False)
511     # popup_text = '[insert plot w/ matplotlib here]'
512     # TODO add charts to popups in GUI
513     # sg.Popup('This is a pop-up for Soil Temperature',
514     #          popup_text, title='Soil Temperature')
515
516 # Popup for Air Humidity
517 if event == 'air_m':
518     plt.close('all')
519     plot_sensor1 = []
520     plot_sensor2 = []
521     hours_ago = [
522         -5.00, -4.75, -4.5, -4.25, -4.00, -3.75, -3.5, -3.25, -3.00,
523         -2.75, -2.5, -2.25, -2.00, -1.75, -1.5, -1.25, -1.00, -0.75,
524         -0.5, -0.25
525     ]
526     for i in range(20):
527         plot_sensor1.append(sensor1_data[i][1])
528         plot_sensor2.append(sensor2_data[i][1])
529     plt.plot(hours_ago, plot_sensor1, 'g', label="Sensor 1")
530     plt.plot(hours_ago, plot_sensor2, 'y', label="Sensor 2")
531     plt.title('Air Humidity Plot - Past 5 Hours')
532     plt.xlabel('Time (Hours Ago)')
533     plt.ylabel('Air Humidity (%)')
534     plt.xlim([-5, 0])
535     plt.ylim([0, 100])
536     plt.legend()
537     plt.show(block=False)
538     # popup_text = '[insert plot w/ matplotlib here]'
539     # TODO add charts to popups in GUI
540     # sg.Popup('This is a pop-up for Air Humidity',
541     #          popup_text, title='Air Humidity')
542
543 # Popup for Air Temperature
544 if event == 'air_t':
545     plt.close('all')
546     plot_sensor1 = []
547     plot_sensor2 = []
548     hours_ago = [
549         -5.00, -4.75, -4.5, -4.25, -4.00, -3.75, -3.5, -3.25, -3.00,
550         -2.75, -2.5, -2.25, -2.00, -1.75, -1.5, -1.25, -1.00, -0.75,
551         -0.5, -0.25
552     ]
553     for i in range(20):
554         plot_sensor1.append(sensor1_data[i][2])
555         plot_sensor2.append(sensor2_data[i][2])
556     plt.plot(hours_ago, plot_sensor1, 'g', label="Sensor 1")
557     plt.plot(hours_ago, plot_sensor2, 'y', label="Sensor 2")
558     plt.title('Air Temperature Plot - Past 5 Hours')
559     plt.xlabel('Time (Hours Ago)')
560     plt.ylabel(u'Air Temperature (\N{DEGREE SIGN}F)')
561     plt.xlim([-5, 0])
562     plt.ylim([0, 100])
563     plt.legend()
564     plt.show(block=False)
565     # popup_text = '[insert plot w/ matplotlib here]'
566     # TODO add charts to popups in GUI
567     # sg.Popup('This is a pop-up for Air Temperature',
568     #          popup_text, title='Air Temperature')
569
570 # Popup for Sunlight
571 if event == 'sun':

```

```

572     plt.close('all')
573     plot_sensor1 = []
574     plot_sensor2 = []
575     hours_ago = [
576         -5.00, -4.75, -4.5, -4.25, -4.00, -3.75, -3.5, -3.25, -3.00,
577         -2.75, -2.5, -2.25, -2.00, -1.75, -1.5, -1.25, -1.00, -0.75,
578         -0.5, -0.25
579     ]
580     for i in range(20):
581         plot_sensor1.append(sensor1_data[i][4])
582         plot_sensor2.append(sensor2_data[i][4])
583     plt.plot(hours_ago, plot_sensor1, 'g', label="Sensor 1")
584     plt.plot(hours_ago, plot_sensor2, 'y', label="Sensor 2")
585     plt.title('Sunlight Plot - Past 5 Hours')
586     plt.xlabel('Time (Hours Ago)')
587     plt.ylabel('Sunlight (%)')
588     plt.xlim([-5, 0])
589     plt.ylim([0, 100])
590     plt.legend()
591     plt.show()
592     # popup_text = '[insert plot w/ matplotlib here]'
593     # TODO add charts to popups in GUI
594     # sg.Popup('This is a pop-up for Sunlight',
595     #           popup_text, title='Sunlight')
596
597     # closes window
598     window.close()
599
600
601 # Load given TXT file
602 def load_file(filename):
603     # Reads data from .txt file
604     text_file = open(filename, 'r')
605     lines = text_file.readlines()
606     text_file.close()
607
608     # Creates new empty list
609     file_contents = [] # list values in form of: [P, N, Stressed]
610
611     # Saves "Not Stressed" values into "studyResults" List
612     for j in range(0, 20):
613         if ";" in lines[j]:
614             new_line = lines[j].split(';')
615         else:
616             new_line = lines[j].split()
617         file_contents.append(new_line)
618
619     # Converts all string values in "studyResults" to float
620     file_contents = [list(map(float, sublist)) for sublist in file_contents]
621
622     return file_contents
623
624
625 # Main Function - Start Here
626 def main():
627     # Step 1: Collect Sensor Data
628     # TODO RECIEVE AND PARSE DATA
629
630     # Step 2: Parse Collected Data
631     # Alan has data format ready
632     # dummy data since i don't have the radio
633     # <soil-temp>;<soil-moist>;<air-temp>;<air-hum>;<sunlight> 58;56;69;43;27
634     sensor_data = "56,58,43,69,27"
635     parsed_data = parse_data(sensor_data)
636
637     # Step 3: Visualize Data
638     # TODO make charts with matplotlib
639     # Read contents of 'HW2_data.txt'
640     sensor1 = load_file(str(Path(__file__).resolve().parent / 'sensor1.txt'))
641     sensor2 = load_file(str(Path(__file__).resolve().parent / 'sensor2.txt'))
642     # print(len(sensor1))
643     # print(len(sensor2))
644
645     # Step 4: GUI
646     stop_threads = False
647     th = threading.Thread(
648         target=(lambda x, y, z: background_collection(x, y, z)),
649         args=(SAMPLE_RATE, lambda: stop_threads, test_background))
650     show_gui(parsed_data, sensor1, sensor2, th)
651     time.sleep(1)
652     stop_threads = True
653     if th.is_alive():
654         print('where all threads go to die')

```

```
655     th.join()
656
657
658 if __name__ == '__main__':
659     main()
```

## 4.2 Submodule Main Control Program

```

1 ///////////////////////////////////////////////////////////////////
2 // MGMS Sensor Module main.cpp
3 // Alan Trester <tresteat@mail.uc.edu> ; Zuguang Liu <liu2z2@mail.uc.edu>
4 // March 14, 2021
5 ///////////////////////////////////////////////////////////////////
6
7 #include <avr/io.h>
8 #include <avr/interrupt.h>
9 #include <stdlib.h>           // qsort
10 #include <stdio.h>            // USART
11 #include <string.h>
12 #include <util/delay.h>
13 #include "AnalogSensors.h"
14 #include "SHT35.h"
15
16 ///////////////////////////////////////////////////////////////////
17 // UART Functions & Vars
18
19 volatile char RXBuffer[30] ;
20 // Populated as new data is received in the RXC ISR. Checked in main()
21 volatile uint8_t RXByteCount = 0 ;
22 // Position of next byte to populate in RXBuffer. Used in RXC ISR.
23
24 char TXBuffer[30] ;
25 // Used to transmit data in TX ISR. Set in FullReport()
26 volatile uint8_t TXByteCount = 1 ;
27 // Position of next byte to transmit. Used in TX ISR
28
29 void memset_volatile(volatile void *s, char c, size_t n) ;
30 // memset that accepts volatile variables - Needed to reset volatile buffers
31
32 // Data Collection Functions & Vars
33 float SoilTemp ;           // Soil Temperature.
34 char SoilTempASCII[7] ;
35
36 float SoilMoist ;          // Soil Moisture.
37 char SoilMoistASCII[7] ;
38
39 float AirTemp ;            // Air Temperature.
40 char AirTempASCII[7] ;
41
42 float Humidity ;           // Humidity.
43 char HumidityASCII[7] ;
44
45 void FullReport() ;        // Concatenated with all above
46 char FullReportASCII[28] ;   // 14th byte reserved for \0
47
48 ///////////////////////////////////////////////////////////////////
49 // memset_volatile - memset function implementation that accept volatile
50 // variables as inputs. Found on stackoverflow This is required to be above main
51 // since it accepts volatile inputs
52 void memset_volatile(volatile char *s, char c, size_t n)
53 {
54     volatile char *p = s ;
55     while (n-- > 0)
56     {
57         *p++ = c ;
58     }
59 }
60
61 ///////////////////////////////////////////////////////////////////
62 // Main Function - Control Register Config; Scan RX buffer and react to the
63 // correct message.
64 int main(void)
65 {
66     ADC_init() ;
67     SHT_init() ;
68
69     // Configure USART
70     uint16_t baud = 103 ; // 9600bps (@16MHz)
71     UCSR0B = (1<<TXCIE0) | (1<<RXCIE0) |(1<<TXENO) | (1<<RXENO) ;
72     // Enable transmitter/receiver/Enable TX/RX Interrupts
73     UCSR0C = 0x06 ;      // Frame format = 8 data + 1 stop bit
74     UBRROH = (baud>>8) ; // Configure baud rate
75     UBRROL = (baud) ;
76
77     // Enable all Interrupts
78     sei() ;
79

```

```

80 // Main Program
81 char RXBufferCpy[30] ;      // Copy stores contents of RXBuffer
82 uint8_t RXByteCountCpy ;    // Copy stores contents of RXByteCount
83 bool CheckFlag = false ;   // Double checked flag for strings
84
85 while (1)
86 {
87     strcpy( RXBufferCpy, const_cast<char*>(RXBuffer) ) ;
88     // Copies the contents of the RXBuffer into a non-volatile copy
89     RXByteCountCpy = RXByteCount ;
90
91     if ( strcmp(RXBufferCpy, "Request Info") == 0 )
92     {
93         UpdateSoilTemp(&SoilTemp, SoilTempASCII) ;
94         UpdateSoilMoist(&SoilMoist, SoilMoistASCII) ;
95         UpdateTempHumid(&AirTemp, AirTempASCII, &Humidity,
96                          HumidityASCII) ;
97
98         memset_volatile(RXBuffer, 0, 30) ; // Reset buffer
99         CheckFlag = false ;           // Reset flag
100        FullReport() ;
101    }
102    else if ( RXByteCountCpy == 0 && strcmp(RXBufferCpy, "") != 0 )
103        // If the last character was a \r but its the wrong message
104    {
105        if (CheckFlag)
106        {
107            memset_volatile(RXBuffer, 0, 30); // Reset the buffer
108            CheckFlag = false ;
109        }
110        else
111        {
112            CheckFlag = true ;
113        }
114    }
115 }
116 }
117 }
118
119 ///////////////////////////////////////////////////////////////////
120 // FullReport - Builds the information string to send and begins TX process
121 void FullReport()
122 {
123     memset(FullReportASCII, 0, 28) ; // Reset the string
124
125     strcpy(FullReportASCII, SoilTempASCII) ;
126     strcat(FullReportASCII, ";" );
127     strcat(FullReportASCII, SoilMoistASCII) ;
128     strcat(FullReportASCII, ";" );
129     strcat(FullReportASCII, AirTempASCII) ;
130     strcat(FullReportASCII, ";" );
131     strcat(FullReportASCII, HumidityASCII) ;
132
133     strcpy(TXBuffer, FullReportASCII) ;
134     UDRO = TXBuffer[0] ;           // Transmit the first byte of data
135 }
136
137 ///////////////////////////////////////////////////////////////////
138 // USART_RX_vect - Triggers whenever the USART module indicates that it has
139 // received a new byte of data Saves latest byte onto the end of the RXBuffer.
140 // Changes carriage returns to terminator characters
141 ISR(USART_RX_vect)
142 {
143     RXBuffer[RXByteCount] = UDRO ; // Retrieve the Received Byte
144
145     if( RXBuffer[RXByteCount] == '\r' )
146     {
147         RXBuffer[RXByteCount] = '\0' ; // Termination character
148         RXByteCount = 0 ;           // Reset the Byte Counter
149         //The actual word gets reset in main()
150     }
151     else if( RXByteCount == 29 ) // At the end of our buffer
152     {
153         memset_volatile(RXBuffer, 0, 30) ; // Reset the RXBuffer
154         RXByteCount = 0 ;           // Reset the Byte Counter
155     }
156     else
157     {
158         RXByteCount++ ;
159     }
160 }
161
162 ///////////////////////////////////////////////////////////////////

```

```
163 // USART_TX_vect - Triggers whenever the UART module indicates that an outgoing
164 // transfer has been completed Transmits the next byte of the TXBuffer
165 ISR(USART_TX_vect)
166 {
167     if (TXBuffer[TXByteCount] != '\0')          // Not yet at end of the buffer
168     {
169         UDRO = TXBuffer[TXByteCount] ;           // Send the next Byte
170         TXByteCount++ ;
171     }
172     else
173     {
174         TXByteCount = 1 ;
175     }
176 }
```

## 4.3 Submodule Libraries

```

1  /*
2   * SHT35.h
3   * Custom Library Supporting SHT35 Sensor - Adapted from Adafruit Library
4   * Zuguang Liu <liu2z2@mail.uc.edu> ; Alan Trester <tresteat@mail.uc.edu>
5   */
6
7
8 #ifndef SHT35_H_
9 #define SHT35_H_
10
11 #include <avr/io.h>
12 #include <util/delay.h>
13 #include <stdlib.h>
14 #include "twi_lib.h"
15
16 #define SHT_ADDR 0x45
17 #define NACK_ON_ADDR 2
18
19
20 #define CLK_STRETCH_ENABLED 0
21 #define CLK_STRETCH_DISABLED 3
22
23 #define MODE MPS_05 6
24 #define MODE MPS_1 9
25 #define MODE MPS_2 12
26 #define MODE MPS_4 15
27 #define MODE MPS_10 18
28
29 #define REPEAT_HIGH 0
30 #define REPEAT_MED 1
31 #define REPEAT_LOW 2
32
33 #define CMD_BREAK 0x3093
34 #define CMD_SOFT_RST 0x30A2
35 #define CMD_READ_SREG 0xF32D
36 #define CMD_CLEAR_SREG 0x3041
37 #define CMD_FETCH_DATA 0xE000
38 #define CMD_READ_SERIAL 0x3780
39
40 #define CMD_READ_HIGH_ALERT_LIMIT_SET_VALUE 0XE11F
41 #define CMD_READ_HIGH_ALERT_LIMIT_CLEAR_VALUE 0XE114
42 #define CMD_READ_LOW_ALERT_LIMIT_SET_VALUE 0XE102
43 #define CMD_READ_LOW_ALERT_LIMIT_CLEAR_VALUE 0XE109
44
45 #define CMD_WRITE_HIGH_ALERT_LIMIT_SET_VALUE 0X611D
46 #define CMD_WRITE_HIGH_ALERT_LIMIT_CLEAR_VALUE 0X6116
47 #define CMD_WRITE_LOW_ALERT_LIMIT_SET_VALUE 0X6100
48 #define CMD_WRITE_LOW_ALERT_LIMIT_CLEAR_VALUE 0X610B
49
50 #define HI REP WI STRCH 0x2C06
51 #define MD REP WI STRCH 0x2COD
52 #define LO REP WI STRCH 0x2C10
53 #define HI REP WO STRCH 0x2400
54 #define MD REP WO STRCH 0x240B
55 #define LO REP WO STRCH 0x2416
56
57 #define CMD_HEATER_ON 0x306D
58 #define CMD_HEATER_OFF 0x3066
59
60 ret_code_t SHT_init() ;
61 //ret_code_t SHT_measure(uint16_t* temp, uint16_t* hum) ;
62 //ret_code_t SHT_check(uint16_t* status) ;
63 void UpdateTempHumid(float* RealTemp, char* TempASCII, float* RealHumid,
64                      char* HumidASCII) ;
65
66 #endif /* SHT35_H_ */

```

```

1 /*
2  * SHT35.cpp
3  * Custom Library Supporting SHT35 Sensor - Adapted from Adafruit Library
4  * Zuguang Liu <liu2z2@mail.uc.edu> ; Alan Trester <tresteat@mail.uc.edu>
5  */
6
7 #include "SHT35.h"
8
9 static ret_code_t SHT_send_cmd(uint16_t cmd){
10     uint8_t data[2] = {cmd >> 8, cmd & 0xFF};
11     return tw_master_transmit(SHT_ADDR, data, 2, false);
12 }

```

```

13
14
15 static float get_temp(uint16_t temp) {
16     return (temp / 65535.00) * 175 - 45;
17 }
18
19 static float get_hum(uint16_t hum) {
20     return (hum / 65535.0) * 100.0;
21 }
22
23 static ret_code_t SHT_measure(uint16_t* temp, uint16_t* hum){
24     ret_code_t error_code = SUCCESS;
25     uint8_t data[6];
26
27     error_code = SHT_send_cmd(HI_REP_WI_STRCH);
28     if (error_code != SUCCESS) return error_code;
29
30     _delay_ms (1);
31
32     error_code = tw_master_receive(SHT_ADDR, data, sizeof(data));
33     if (error_code != SUCCESS) return error_code;
34
35     *temp = (uint16_t) data[0] << 8 | data[1];
36     *hum = (uint16_t) data[3] << 8 | data[4];
37
38     return error_code;
39 }
40
41 static ret_code_t SHT_check(uint16_t* status){
42     ret_code_t error_code = SUCCESS;
43     uint8_t data[3] = {0};
44
45     error_code = SHT_send_cmd(CMD_SOFT_RST);
46     if (error_code != SUCCESS) return error_code;
47
48     error_code = SHT_send_cmd(CMD_READ_SREG);
49     if (error_code != SUCCESS) return error_code;
50
51     _delay_ms (20);
52
53     error_code = tw_master_receive(SHT_ADDR, data, sizeof(data));
54     if (error_code != SUCCESS) return error_code;
55
56     *status = data[0] << 8 | data[1] ;
57
58     return error_code;
59 }
60
61 ret_code_t SHT_init(){
62     ret_code_t error_code = SUCCESS;
63     tw_init(TW_FREQ_250K, true); // set I2C Frequency, enable internal pull-up
64
65     error_code = SHT_send_cmd(CMD_SOFT_RST);
66     if (error_code != SUCCESS) return error_code;
67
68 //    error_code = SHT_send_cmd(CMD_CLEAR_SREG);
69 //    if (error_code != SUCCESS) return error_code;
70
71     return error_code;
72 }
73
74 void UpdateTempHumid(float* RealTemp, char* TempASCII, float* RealHumid,
75                       char* HumidASCII)
76 {
77     uint16_t RawTemp ;
78     uint16_t RawHumid ;
79
80     SHT_measure(&RawTemp, &RawHumid) ;
81
82     *RealTemp = get_temp(RawTemp) ;
83     *RealHumid = get_hum(RawHumid) ;
84
85     dtostrf( *RealTemp, 6, 2, TempASCII) ;
86     dtostrf( *RealHumid, 6, 2, HumidASCII) ;
87 }
```

```

1  /*
2   * SHT35.h
3   * Custom Library Supporting I2C for MGMS - Adapted from TEP SOVICHEA
4   * Zuguang Liu <liu2z2@mail.uc.edu> ; Alan Trester <tresteat@mail.uc.edu>
5   */
6
7
8
9 #ifndef TWI_LIB_H_
10#define TWI_LIB_H_
11
12#include <avr/io.h>
13#include <util/twi.h>
14#include <stdbool.h>
15
16#define SUCCESS 0
17
18#define TW_SCL_PIN PORTC5
19#define TW_SDA_PIN PORTC4
20
21#define TW_SLA_W(ADDR) ((ADDR << 1) | TW_WRITE)
22#define TW_SLA_R(ADDR) ((ADDR << 1) | TW_READ)
23#define TW_READ_ACK 1
24#define TW_READ_NACK 0
25
26typedef uint16_t ret_code_t;
27
28typedef enum {
29    TW_FREQ_100K,
30    TW_FREQ_250K,
31    TW_FREQ_400K
32} twi_freq_mode_t;
33
34void tw_init(twi_freq_mode_t twi_freq_mode, bool pullup_en);
35ret_code_t tw_master_transmit(uint8_t slave_addr, uint8_t* p_data, uint8_t len,
36                             bool repeat_start);
37ret_code_t tw_master_receive(uint8_t slave_addr, uint8_t* p_data, uint8_t len);
38
39#endif /* TWI_LIB_H_ */

```

```

1  /*
2   * SHT35.cpp
3   * Custom Library Supporting I2C for MGMS - Adapted from TEP SOVICHEA
4   * Zuguang Liu <liu2z2@mail.uc.edu> ; Alan Trester <tresteat@mail.uc.edu>
5   */
6
7#include "twi_lib.h"
8
9static ret_code_t tw_start(void)
10{
11    /* Send START condition */
12    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTA);
13
14    /* Wait for TWINT flag to set */
15    while (!(TWCR & (1 << TWINT)));
16
17    /* Check error */
18    if (TW_STATUS != TW_START && TW_STATUS != TW REP START)
19    {
20        return TW_STATUS;
21    }
22
23    return SUCCESS;
24}
25
26
27static void tw_stop(void)
28{
29    /* Send STOP condition */
30    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
31}
32
33
34static ret_code_t tw_write_sla(uint8_t sla)
35{
36    /* Transmit slave address with read/write flag */
37    TWDR = sla;
38    TWCR = (1 << TWINT) | (1 << TWEN);
39
40    /* Wait for TWINT flag to set */
41    while (!(TWCR & (1 << TWINT)));
42    if (TW_STATUS != TW_MT_SLA_ACK && TW_STATUS != TW_MR_SLA_ACK)
43    {

```

```

44     return TW_STATUS;
45 }
46
47     return SUCCESS;
48 }
49
50
51 static ret_code_t tw_write(uint8_t data)
52 {
53     /* Transmit 1 byte*/
54     TWDR = data;
55     TWCR = (1 << TWINT) | (1 << TWEN);
56
57     /* Wait for TWINT flag to set */
58     while (!(TWCR & (1 << TWINT)));
59     if (TW_STATUS != TW_MT_DATA_ACK)
60     {
61         return TW_STATUS;
62     }
63
64     return SUCCESS;
65 }
66
67
68 static uint8_t tw_read(bool read_ack)
69 {
70     if (read_ack)
71     {
72         TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
73         while (!(TWCR & (1 << TWINT)));
74         if (TW_STATUS != TW_MR_DATA_ACK)
75         {
76             return TW_STATUS;
77         }
78     }
79     else
80     {
81         TWCR = (1 << TWINT) | (1 << TWEN);
82         while (!(TWCR & (1 << TWINT)));
83         if (TW_STATUS != TW_MR_DATA_NACK)
84         {
85             return TW_STATUS;
86         }
87     }
88     uint8_t data = TWDR;
89     return data;
90 }
91
92
93 void tw_init(twi_freq_mode_t twi_freq_mode, bool pullup_en)
94 {
95     DDRC |= (1 << TW_SDA_PIN) | (1 << TW_SCL_PIN);
96     if (pullup_en)
97     {
98         PORTC |= (1 << TW_SDA_PIN) | (1 << TW_SCL_PIN);
99     }
100    else
101    {
102        PORTC &= ~((1 << TW_SDA_PIN) | (1 << TW_SCL_PIN));
103    }
104    DDRC &= ~((1 << TW_SDA_PIN) | (1 << TW_SCL_PIN));
105
106    switch (twi_freq_mode)
107    {
108        case TW_FREQ_100K:
109            /* Set bit rate register 72 and prescaler to 1 resulting in
110            SCL_freq = 16MHz/(16 + 2*72*1) = 100KHz */
111            TWBR = 72;
112            break;
113
114        case TW_FREQ_250K:
115            /* Set bit rate register 24 and prescaler to 1 resulting in
116            SCL_freq = 16MHz/(16 + 2*24*1) = 250KHz */
117            TWBR = 24;
118            break;
119
120        case TW_FREQ_400K:
121            /* Set bit rate register 12 and prescaler to 1 resulting in
122            SCL_freq = 16MHz/(16 + 2*12*1) = 400KHz */
123            TWBR = 12;
124            break;
125
126        default: break;

```

```

127     }
128 }
129
130
131 ret_code_t tw_master_transmit(uint8_t slave_addr, uint8_t* p_data, uint8_t len,
132                             bool repeat_start)
133 {
134     ret_code_t error_code;
135
136     /* Send START condition */
137     error_code = tw_start();
138     if (error_code != SUCCESS)
139     {
140         return error_code;
141     }
142
143     /* Send slave address with WRITE flag */
144     error_code = tw_write_sla(TW_SLA_W(slave_addr));
145     if (error_code != SUCCESS)
146     {
147         return error_code;
148     }
149
150     /* Send data byte in single or burst mode */
151     for (int i = 0; i < len; ++i)
152     {
153         error_code = tw_write(p_data[i]);
154         if (error_code != SUCCESS)
155         {
156             return error_code;
157         }
158     }
159
160     if (!repeat_start)
161     {
162         /* Send STOP condition */
163         tw_stop();
164     }
165
166     return SUCCESS;
167 }
168
169
170 ret_code_t tw_master_receive(uint8_t slave_addr, uint8_t* p_data, uint8_t len)
171 {
172     ret_code_t error_code;
173
174     /* Send START condition */
175     error_code = tw_start();
176     if (error_code != SUCCESS)
177     {
178         return error_code;
179     }
180 //    return error_code;
181
182     /* Write slave address with READ flag */
183     error_code = tw_write_sla(TW_SLA_R(slave_addr));
184     if (error_code != SUCCESS)
185     {
186         return error_code;
187     }
188
189     /* Read single or multiple data byte and send ack */
190     for (int i = 0; i < len-1; ++i)
191     {
192         p_data[i] = tw_read(TW_READ_ACK);
193     }
194     p_data[len-1] = tw_read(TW_READ_NACK);
195
196     /* Send STOP condition */
197     tw_stop();
198
199     return SUCCESS;
200 }
```

```

1  /*
2   * AnalogSensors.h
3   * Custom Library Supporting ADC Reads for Soil Temp, Soil Moisture, Sunlight
4   * Alan Trester <tresteat@mail.uc.edu> ; Zuguang Liu <liu2z2@mail.uc.edu>
5   * 3/14/2021
6  */
7
8
9 #ifndef ANALOGSENSORS_H_
10#define ANALOGSENSORS_H_
11
12#include <avr/io.h>
13#include <stdlib.h>
14
15void ADC_init() ;
16// Initializes the analog pins to correctly do measurements
17void UpdateSoilTemp(float* TempMode, char* ASCII) ;
18// Reads Soil Temp values from ADC, finds the most likely temperature
19void UpdateSoilMoist(float* MoistMode, char* ASCII) ;
20// Reads Soil Moisture values from ADC, finds the most likely moisture
21
22#endif /* ANALOGSENSORS_H_ */

1 /*
2 * AnalogSensors.cpp
3 * Custom Library Supporting ADC Reads for Soil Temp, Soil Moisture, Sunlight
4 * Alan Trester <tresteat@mail.uc.edu> ; Zuguang Liu <liu2z2@mail.uc.edu>
5 * 3/14/2021
6 */
7
8#include "AnalogSensors.h"
9
10////////////////////////////////////////////////////////////////
11// Necessary for qsort to work. Compares the values for 2 float values
12static int CompareFloat(const void* a, const void* b)
13{
14    if ( *(float*)a < *(float*)b ) return -1 ;
15    if ( *(float*)a == *(float*)b ) return 0 ;
16    if ( *(float*)a > *(float*)b ) return 1 ;
17}
18
19////////////////////////////////////////////////////////////////
20// Initializes the analog pins to correctly do measurements
21void ADC_init()
22{
23    ADMUX = (0<<REFS1)|(1<<REFS0)|(0<<ADLAR)|(0<<MUX3)|(0<<MUX2)|(0<<MUX1)|
24        (0<<MUX0);
25    // Select Analog Port 0 and Internal Reference Voltage;
26    ADCSRA = (1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(0<<ADPS1)|(0<<ADPS0);
27    // Enable A/D, Enable Interrupt, Set A/D Prescalar
28    DIDR0 = (1<<ADCOD) | (1<<ADC1D);
29    // Disable Input Buffers
30}
31
32////////////////////////////////////////////////////////////////
33// Reads Soil Temp values from ADC, reports the most likely temperature
34void UpdateSoilTemp(float* TempMode, char* ASCII)
35{
36    ADMUX = (0<<REFS1)|(1<<REFS0)|(0<<ADLAR)|(0<<MUX3)|(0<<MUX2)|(0<<MUX1)|
37        (0<<MUX0); // Select Analog Port 0 and Internal Reference Voltage
38
39    uint8_t i ; // for loop counter
40    uint16_t DataH ; // ADC High Bits
41    uint16_t DataL ; // ADC Low Bits
42    float DataFloat ; // Needed for int -> float cast
43    float RawValues[10] ; // Samples for mode function
44
45    uint8_t Counts[10] ; // Occurrence of each value in rawData
46    uint8_t MaxCount ; // Array location of the highest count
47
48    // Take 10 measurements
49    for (i = 0 ; i<10 ; i++)
50    {
51        ADCSRA |= (1<<ADSC) ; // Begin ADC conversion on Channel 0
52        while( ADCSRA & (1<<ADSC)) ; // Wait for ADC conversion to complete
53
54        DataL = ADCL ;
55        DataH = ADCH << 8 ;
56        DataH = DataH + DataL ;
57
58        DataFloat = DataH ; // Cast to Float
59        RawValues[i] = ( (DataFloat / 1024) *5 ) * 75.006 ) - 40 ;
60        // Equation from THERM200 Datasheet

```

```

61 }
62
63 // Begin finding mode of the measured data
64
65 qsort(RawValues, 10, sizeof(float), CompareFloat) ; // Ascending sort
66 // Count occurrence of each value
67 Counts[0] = 1 ; // Count the first value
68 for (i = 1 ; i<10 ; i++) // Count the rest
69 {
70     if ( RawValues[i] == RawValues[i-1] )
71     {
72         Counts[i] = Counts[i-1] + 1 ;
73     }
74     else
75     {
76         Counts[i] = 1 ; // restart the count
77     }
78 }
79
80 // Finds the largest count
81 MaxCount = 0 ; // Count the first value
82 for (i = 1 ; i<10 ; i++) // Count the rest
83 {
84     if ( Counts[i] > Counts[MaxCount] ) MaxCount = i ;
85     // Update the MaxCount as necessary
86 }
87
88 *TempMode = RawValues[MaxCount] ;
89 dtostrf( RawValues[MaxCount], 6, 2, ASCII) ;
90 }
91
92 ///////////////////////////////////////////////////////////////////
93 // Reads Soil Moisture values from ADC, reports the most likely moisture
94 void UpdateSoilMoist(float* MoistMode, char* ASCII)
95 {
96     ADMUX = (0<<REFS1)|(1<<REFS0)|(0<<ADLAR)|(0<<MUX3)|(0<<MUX2)|(0<<MUX1)|
97             (1<<MUX0) ;
98     // Select Analog Port 1 and Internal Reference Voltage;
99
100    uint8_t i ;
101    uint16_t DataH ;
102    uint16_t DataL ;
103    float Voltage ; // Voltage for piecewise approximation
104    float RawValues[10] ; // Samples for mode function
105
106    uint8_t Counts[10] ; // Occurrence of each value in RawData
107    uint8_t MaxCount ; // Address of the highest count
108
109    // Take 10 measurements
110    for (i = 0 ; i<10 ; i++)
111    {
112        ADCSRA |= (1<<ADSC) ; // Begin ADC conversion on Channel 0
113        while( ADCSRA & (1<<ADSC)) ; // Wait for ADC conversion to complete
114
115        DataL = ADCL ;
116        DataH = ADCH << 8 ;
117        DataH = DataH + DataL ;
118
119        Voltage = DataH ; // Cast to Float
120        Voltage = (Voltage / 1024) * 5 ; // Convert to Voltage
121
122        // Piecewise approximation for Voltage -> VWC from VH400 Datasheet
123        if (Voltage < 1.1) RawValues[i] = (10 * Voltage) - 1 ;
124        else if (Voltage < 1.3) RawValues[i] = (25 * Voltage) - 17.5 ;
125        else if (Voltage < 1.8) RawValues[i] = (48.08 * Voltage) - 47.5 ;
126        else if (Voltage < 2.2) RawValues[i] = (26.32 * Voltage) - 7.89 ;
127        else RawValues[i] = (62.5 * Voltage) - 7.89 ;
128    }
129
130    // Begin finding mode of the measured data
131
132    qsort(RawValues, 10, sizeof(float), CompareFloat) ;// Sorts ascendingly
133
134    // Count occurrence of each value
135    Counts[0] = 1 ;
136    for (i = 1 ; i<10 ; i++)
137    {
138        if ( RawValues[i] == RawValues[i-1] )
139        {
140            Counts[i] = Counts[i-1] + 1 ;
141        }
142        else
143        {

```

```
144     Counts[i] = 1 ;
145 }
146
147 // Finds the largest count
148 MaxCount = 0 ;
149 for (i = 1 ; i<10 ; i++)
150 {
151     if ( Counts[i] > Counts[MaxCount] ) MaxCount = i ;
152 }
153
154
155 *MoistMode = RawValues[MaxCount] ;
156 dtostrf( RawValues[MaxCount], 6, 2, ASCII) ;
157 }
```