Auto Garden Bed - Group 9

Team members:

Nicholas Chitty - Electrical Engineer
Brendan College - Computer Engineer
Scott Peirce - Optical Engineer
Justin Pham-Trinh - Electrical Engineer

UCF Senior Design Fall 2022 - Spring 2023

Contents

List of Figures

List of Tables

1 Executive Summary

New gardeners typically struggle getting their garden started due to a lack of tending to their plants. This project seeks to solve many of the problems that new gardeners have through sensing and control. The main issues with plant growth relate to soil composition, soil moisture, temperatue, and sun light. This project seeks to use optics to measure the soil moisture and composition; then an MCU will capture this data and control solar shades to control sunlight and solenoids to control watering. A web component will be included to check the weather as well as notify the user of impending weather events that could affect their plants adversively (frost or heat wave). The entire system will be powered with solar panels that are capable of tracking the sun through the sky and can act as blinds over the plants.

This project all starts with scoping out the project. The team has immediately compiling a list of must-have requirements and some things the team would like to accomplish as "nice to haves". The team started this process by looking at all the similar projects that have already been done and looked at all the ways the team can expand on the work they have already accomplished. For example, the team liked the weather aspects of a project for getting rain information; a problem the team were thinking about was how to get the system in as much of a "set it and forget it" state as possible as it pertained to frost. The solution is to integrate with a weather service online and send notifications when there is a frost or freeze advisory.

After assembling the list of requirements, the team set out to create a high-level functional block diagram for each of the subsystems. This helps the team see where the different systems integrate for the future as well as breaking out all the different components that may need to be purchased.

The ultimate novelty in this project is all of the sensing that will be done through spectroscopy. The team has found a plethora of research on the topic and has started familiarizing themselves with the limitations and capabilities of the available technology. Ideally, the team would like to find a scalable solution to the sensing in which the optical sensing could be attached to a drone or satellite to survey fields for farming.

2 Project Description

At the core, this project is a microcontroller project that anybody with a slight understanding of digital communications could accomplish. Our team in the background of our project will cover the novelties of this project that separates us from any "Garduino" project that can be found on numerous blogs. In our Goals and Objectives, we will cover how we will accomplish the novelties as well as the specifics as to why our project is better and more marketable.

2.1 Project Background

Home gardening is a valuable hobby that helps people get outdoors, create something beautiful, and earn tangible rewards. Unfortunately, plants are sensitive organisms that require consistent attention. Plant life also involves complex relationships between the organisms and their environment. These two problems can discourage potential gardeners. This team proposes to build a system that addresses these issues through mechanical automation and user notification.

In 2022 there is great potential for realizing these advantages. Remote sensing, wireless communication, API integration, and closed-system power and water control are both available and economical. Many "DIY smart-gardens" and "Garduino" projects have been published to the internet. In the agriculture industry, there are commercially available technologies with high performance systems for water distribution, network communication, and remote sensing. This project is intended to advance the field by producing a system that can maintain a suitable environment for plant growth by autonomously sensing and modifying the conditions of the garden bed. In addition, it will feature a notification system that allows the plants to "speak" by prompting the user to make accommodations when bad weather is projected, such as a cold snap.

Especially worth noting is that this project will feature on-the-rise technology in the form of a Near Infrared Spectrophotometer. Smart agricultural systems need to determine several variables, including moisture level, nutrient content, and acidity. There is a wide variety of techniques for sensing soil moisture, but by far the cheapest and most used is electrical conductivity. This involves pressing electrical nodes into the ground and up against the wet soil matrix, which inevitably leads to corrosion. DIY and commercial systems require other discrete sensors and even chemical analysis to characterize the state of the soil. Near Infrared Spectroscopy is an alternative method of soil sensing that offers many advantages over these traditional technologies. It works by stimulating a response from weak molecular bonds in the soil, isolating the frequencies of that response, and comparing the signal strengths to that of a known sample. In addition to being a noninvasive, corrosion-immune source of information about soil moisture, the NIR Spectrometer gathers data about the chemical contents of the soil. This means that the same scan will detect the presence and quantity of soil nutrients and water acidity as well. A high speed, high precision spectrometer

costs between $5,000 - $10,000, but this application requires only rudimentary sensing capabilities. It may even be possible to acquire the parts for less than $500, and if this is the case, then it would be a significant step towards increasing the adoption of state-of-the-art sensing technologies.

The proposed system integrates sensing with a power control scheme. The garden bed will sit under a gantry, which will serve as the foundation for a solar panel array. If deemed efficient, the solar panel will have a swivel mount to maximize energy captured. This power will be stored in a battery and used to control a solenoid that will release water into the garden bed when prompted by the sensing system. It will also power the microcontroller, NIR Spectrometer, Solar panel swivel mount, and wireless communication.

2.1.1 Motivation

The idea came from seeing the "Garduino" style projects all over hobbyist forums and websites but the idea really took hold in that each member of the team saw an opportunity to explore a new facet of engineering they held an interest in. This project provided the team an opportunity to apply our knowledge on power systems and delivery, controls, digital signal processing, and optical sensing. These are all areas that the team wanted to demonstrate a high level of understanding in and grow at the synthesis level.

2.2 Goals and Objectives

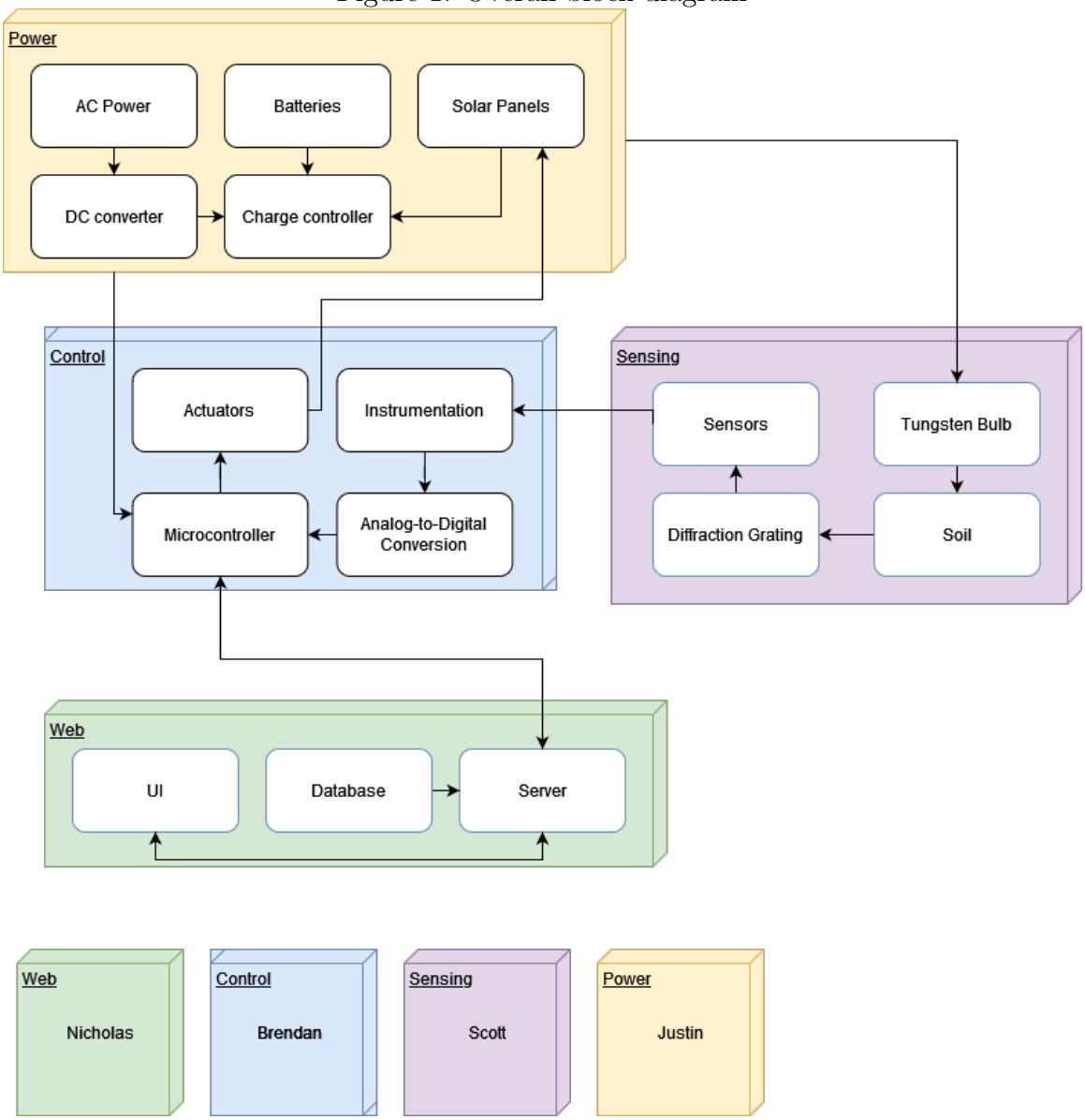
The primary goal of this project is to have a plant bed. This plant bed should be able to actively measure soil moisture content, soil OH group content, and soil acidity. Based on these measurements the plant bed will communicate with a web interface to serve this data to users as well as allow users to control the watering of their plant bed remotely. The plant bed will be powered via solar panels to charge batteries in order to provide a truly "set it and forget it" gardening experience.

Function of the Project Per the motivation of this project, the plant bed should serve to ease new gardeners into the hobby by removing the monotony of watering and allowing users to focus on researching the proper parameters for their plants.

2.3 Specifications and Requirements

This project idea requires the integration of power, systems, computer, optical, and web engineering to achieve a final product of a self-sustaining plant bed. The design in Figure 1 shows four different subsystems the team has designated as power, control, sensing, and web.

Figure 1: Overall block diagram



Control The control subsystem is the brains of the entire operation. This subsystem has to accomplish four distinct tasks:

1. Actuate mechanical components (linear rail, solenoid valve)
2. Convert analog sensor data to digital data
3. Send plant bed telemetry to web subsystem
4. Receive commands from web subsystem and modify system accordingly

In order to accomplish the first task, the chosen microcontroller (MCU) must be capable of driving the currents for these components and also support pulse-width modulation (PWM) for interacting with the motor controller. The second task requires that the MCU have an analog-to-digital converter. The third and fourth tasks necessitate WiFi connectivity as well as firmware support for either HTTP requests or WebSockets. These requirements for the control subsystem weigh heavily in the discussion of which MCU to choose found in Section 3.3.1.

Power The power system's function is self-explanatory, supply power to the entire system. This will be accomplished in two ways. First, using a DC barrel-plug to the wall, the system could be powered this way. The second way, is via the solar panels, batteries, and charge controller. Both manners of supplying power to the system must be regulated. Discussed in Section 3.3.2 are the different ways that the charge controller can efficiently switch between battery power and the panels to increase battery health and charge level.

Sensing The sensing subsystem is where the majority of the novelty of this project lies. A lot of problems with electronic sensing is decay in the wet environs of plant beds. To improve upon this problem and hopefully offer a more sustainable and expandable approach, the team is going to try to accomplish sensing optically via infrared spectroscopy. Through the use of spectroscopy the team will be able to collect a wider range of data than just soil moisture content which includes soil OH group content, temperatue, and acidity.

Web The web component of this project accomplishes three things; data analysis, reporting, and user specified control of components. The data analysis is occurring on the web because of the greater availability of compute power and greater ease of programmability. For reporting, the web will use databases to store data ad infinitum and serve this data in the form of graphs or "live" metrics. The web component also will be able to take a user's input to send commands back to the control system to actuate different parts such as the solenoids or to ask for a more recent data reading.

2.3.1 Engineering Specifications

The team compiled a set of specifications for each of the subsystems that would need to be implemented to accomplish the requirements from above. Table 1 shows breaks down these specifications by the corresponding subystem.

Table 1: Engineering Specifications

Subsystem	Metric	Specification
Control	Input voltage	2.8-5.5 V
	Power consumption	≤1.50W nominal power draw
	Shutdown power draw	≤100 uW
	Processor speed	≥20 MHz
	ADC resolution	≥8-bit
	ADC sampling rate	≥1 ksps
	Transmit power	≥10 dBm
	Receive sensitivity	≥-50 dBm
Power	Power	20-30W
	Battery Capacity	8Ah/95Wh
	Battery Nominal Voltage	9.6-12.8V
	Charging Time	<4 hours
	Regulator Efficiency	>65%
Sensing	Spectra	400nm - 1700nm
	Resolution	<10nm
	Output voltage range	0-1.8V
Web	Up-time	95±.1%
	Storage Capacity	>16 GB
Miscellaneous	Dimensions	1 m^3
	Weight[1]	<50lb

Control The specifications in the control portion of the table are focused on integrating the various subsystems in this product. Input voltage must be flexible to take either 3.3 V or 5 V as an input, plus or minus 0.5 V to account for possible fluctuations in power. The controls subsystem must not consume a large amount of power under nominal conditions to increase battery life and decrease battery size and capacity (and as a result, cost). Shutdown power draw must also be minimal, especially when the battery is completely drained. If the controls subsystem continues to draw from the battery past its rated capacity, the product may become a danger to the user. The processor must be fast enough to handle multiple operations concurrently and in a timely manner, while still being able to drive the network stack and wireless elements. The analog-to-digital converter must have a large enough resolution to accurately take measurements from the sensing subsystem, and must be able to sample sufficiently fast enough to take those measurements in fast enough succession to where they will be useful for our product. Transmit power must be ample enough to operate on a normal consumer WiFi network, and the receiver must be adequately sensitive enough to operate outside, away from the user's wireless access point.

[1]The weight of the system includes a full soil load

Power The power specifications were made throughout the design process. For example, the power output was an unknown until all the subsystems had a firm idea of their power requirements. The maximum instantaneous power draw of the entire system is in the 20-30W range, hence where that specification is drawn from. The battery capacity and charging time are derived from the duration the team decided the system should last on battery power and the input power from the solar panels.

Sensing The spectra was chosen

Web The web engineering specifications were hard to put into words. In general, the team wanted a reliable service quantified by the up-time of the web service. The service also needs to be able to store data in perpetuity for at least a single plantbed. Due to the efficiency of data storage and compression as well as the frequency of scans, the team felt that 16GB was effective enough to showcase the full system for now.

Miscellaneous The current miscellaneous specifications regard the physical qualities of the system. The dimensions of a meter cubed are necessary for the depth needed for various plants as well as solar panels. The weight is heavily correlated to weight of batteries, control and sensing components, as well as the soil and water hosing system.

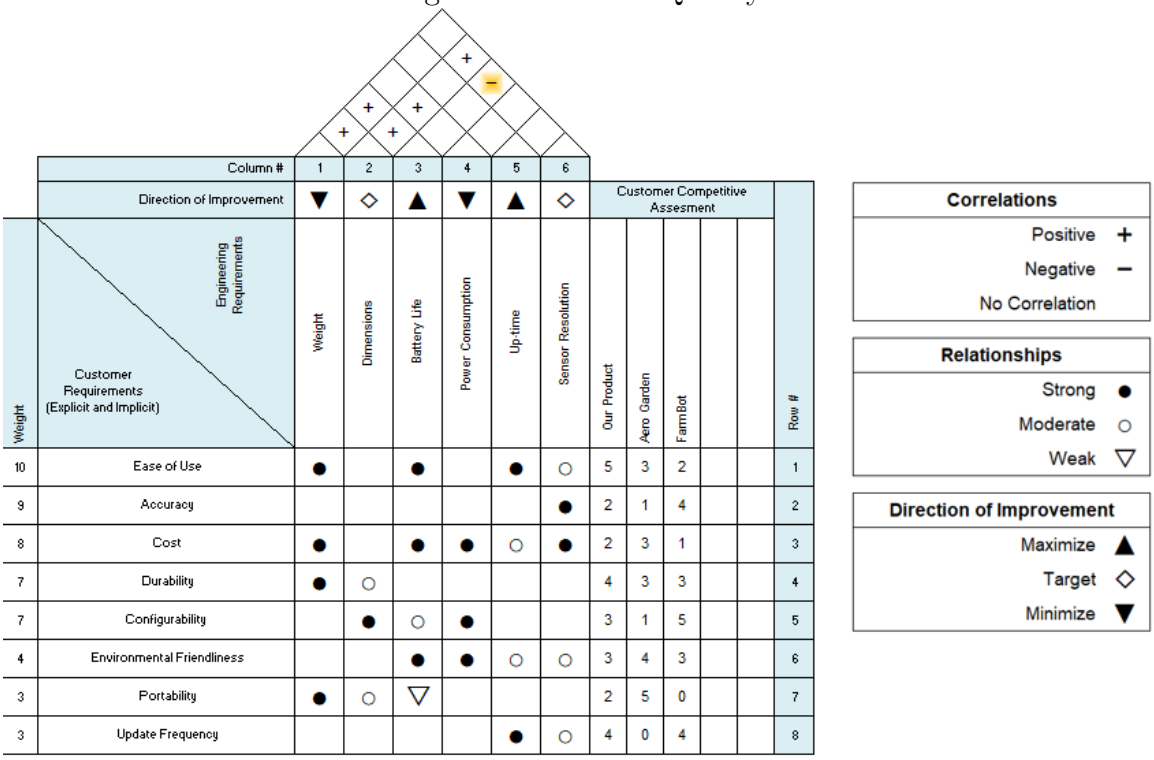
2.3.2 Marketing Requirements

When thinking about the target market for a project like this, the team put together a list of requirements important to the consumers. The most important feature for the consumer would be how easy the product is to use and navigate. This feature ranks the highest and would be the main determining factor on whether a consumer purchases this product over a competitor. Two other highly important features to a consumer would be the accuracy of the product and the cost. The accuracy is important because it directly correlates to the success of the customer's garden. If the product is not accurate in watering the plants, the customer will be unsatisfied and the plants will not survive. Cost because every consumer has to consider their budget - how much they are willing to spend for the features they deem most important. Other customer requirements the team brought up include the durability, configurability, environmental friendliness, portability and update frequency; listed from more important to less for consumer preferences. Each of these requirements have a direct correlation to at least one of the engineering requirements set for the product.

The engineering requirements set by the team are: weight of the product, dimensions of the product, battery life, how much power it consumes, up-time of the website, and quality of sensor resolution.

Many of the engineering requirements are directly correlated with each other. As weight improves, or lessens, so will product dimensions and vice versa. As dimensions decrease for the product, the amount of power needed decreases which means power consumption lessens which is also a positive correlation for the requirements. The only negative correlation in the engineering requirements the team found was that as sensor resolution increases, the life of the battery decreases as the lighting for the sensor must remain on longer.

Figure 2: House of Quality



When considering the customer requirement of ease of use, the team considered the impact of all the engineering requirements. Weight, battery life, and up-time all have a positive impact on customer experience. The lighter the product is, the easier it is for the customer to use because they can move it and adjust things within the product easier. The longer the batter life, the less time the consumer needs to worry when there are days with lower sunlight because the batteries can withstand a longer duration without being recharged. When the product has more up-time, the less time the consumer has to spend waiting on product updates. One of the engineering requirements that has a moderate impact on the customer experience is the sensor resolution. Although it can make a large impact on the overall product performance, its direct correlation to ease of use is not as strong as other requirements. The second most important marketing requirement listed by the team was accuracy. The only

engineering requirement that had any correlation, had a strong correlation which is sensor resolution. This requirement is important to this marketing requirement because it could be a tipping point on whether a consumer purchases this product versus another. If the sensor resolution is not high enough to detect watering needs for the plant, the product will not be successful. This is why it is such an important requirement to focus on for both marketing and development of the product.

Something every team and every consumer is going to consider when designing or purchasing a product is the cost. The team rated this marketing requirement as an 8/10 for this product. It was not the highest priority because for a product like this, consumers will pay more to have a more accurate and easy-to-use product. The team also considered that customers using this product will do more research about the technical features because they are most likely growing fruits or vegetables that they will consume. Unfortunately, most things that increase the quality and appeal of the product, have a negative relationship that is strong in correlation. When the team looks at lighter materials, those materials increase the price. When adding a longer battery life and lowering power consumption, price will increase proportionally to how much those requirements change. To increase the resolution of the sensor, the team and consumers will have to pay more for the increase in quality and performace. The only requirement that correlates to cost that was not strong, but still has an impact is the up-time. The more up-time of the product services, the more cost that is involved, but it is a smaller margin compared to the other requirements that directly impact the cost of the product.

Durability is most affected by the engineering requirements of weight and dimension. If the product materials are different, the durability will change and if the dimensions change shape or length, the construction will change affecting product strength. Weight will have a stronger impact on durability because it is something cosumers will consider more heavily and the materials will impact more. This is why weight has a strong correlation and dimensions are moderately correlated. Another marketing requirement that is similarly affected by weight and dimension is the portability of the product. The battery life also affects the portability. As battery life increases, so does the size and configuration of the battery which affects how easy it is to move. Weight has the largest affect on portability, followed by dimensions and then battery life. The configurability of the project is strongly affected by both dimension and power consumption. Battery life has a smaller affect on this, but still changes the set-up of the product.

Although environmental friendliness rates lower on the consideration for a consumer, it is affected by more engineering requirements. Battery life has a small impact on how environmentally friendly this product is because of the waste that will exist once the battery is retired. The factor considered most impactful is the power consumption. The greater the power consumption, the less friendly the product will be

to the environment. If the team is able to focus on lowering the power consumption, marketing will be easiest for environmental friendliness. Up-time and sensor resolution have a moderate affect on this requirement and should have some connsideration in design of the product, but will be considered with more correlation to other requirements than environmental impact.

The last requirement comparison done by the team is update frequency. This marketing requirement is strongly correlated to up-time and moderately correlated to sensor resolution. Both of these engineering requirements affect the time spent on updating the software for the product.

Competition This product already has a competitive edge in the current market because it is unique. It will be the first all-inclusive garden bed that can be used outside. When looking at products that offer something similar, the team found two main options that a consumer may consider next to this product. The first competitor being considered is the Aero Garden. This product is a small indoor herb garden that provides UV light to help the plants thrive. A consumer wanting to start only an herb garden indoors may consider this product because it is more affordable, but would choose our product for its ease-of-use. The full garden bed design is more durable and waters the plants for the consumer. This is a big factor for a consumer to consider because many gardeners over or under water their plants. So although the Aero Garden may be more competitive indoors, the target market for our product will be won over with the requirements our team has focused on. The second competitor we considered is the FarmBot. This product is an automatic watering system that can be installed on an outside garden bed. When consumers glance at both of the products, they will automatically lean towards ours because of the value and price point. The FarmBot may have more accuracy, but not significant enough to justify the price difference. Additionally, this product is harder to assemble and larger which won't accomodate the average home-gardener. FarmBot is for a consumer that has more time, space, and money, but not for the majority of consumers.

3 Research

Our research covers three core components: previous works, related technology, and part selection.

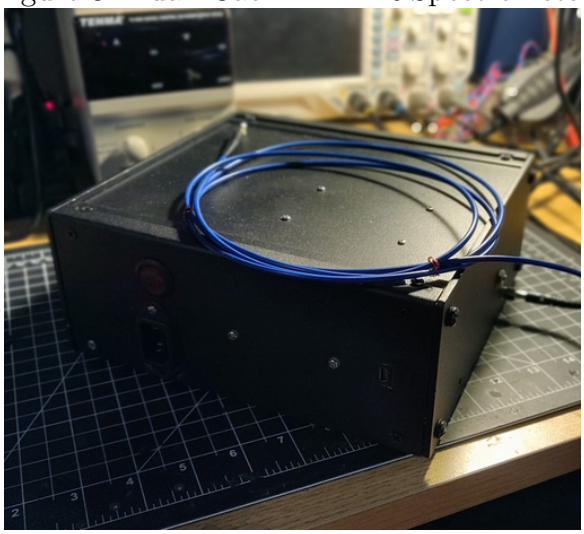
3.1 Previous and Related Works

These are projects that accomplished similar objectives to what we hope to accomplish or highlight a technology we plan to work with.

3.1.1 DIY near-IR spectrometer

Yuan Cao is a Ph.D. student at MIT with a blog where he posts his unofficial projects. He published a project titled "A $500 DIY near-IR spectrometer that would sell for $10,000." In it, he describes his idea, designs and results for a low cost infrared spectrometer. His design features surprisingly high performance for its price. The system uses an InGaAs photodiode, a reflective diffraction grating, a fiber collimator, some cheap optics, and a microcontroller to create a spectrograph of any light source. It boasts very high signal to noise ratios, 5-6nm spectral resolution, and a price point under $500. There are definitely some design features worth imitating, specifically the diffraction grating "scanner" that is rotated to pass wavelengths across the surface of the photodiode, as well as the filter circuitry that cleans up the electrical signal.

Figure 3: Yuan Cao DIY NIR Spectrometer



That said, there are some reasons why this project differs in application from the Auto Garden Bed Near Infrared Spectrometer. First, the detector has a spectral range from 800 – 1600nm. This is the Near Infrared Regime, but it is not very far into the NIR, which in some contexts refers to wavelengths as far out as 2500nm. If Soil Spectroscopy requires this depth, or frequencies in the visible spectrum, the design will have to be modified. Second, this system was built for lab use, specifically to characterize light sources. The reported data was very promising, but in every case the spectrometer targeted an object that was emitting strong optical power in every direction. Soil does not fluoresce, so the Auto Garden Bed will require additional components to probe the soil with an electromagnetic wave. In the end, what's most important is that this project demonstrates that low cost spectroscopy can be achieved.

3.1.2 Chlorophyll Flourescence Spectrometer

In 2021 UCF's ECE Senior Design Group 1 designed a Chlorophyll Fluorescence Spectrometer. The purpose of this project was to design a system that would detect chlorophyll in plants through stimulation by UV rays. It featured a diffraction grating and a monochromatic CMOS sensor. It is interesting to note that unlike the previously explored project, spatial separation of light frequencies is achieved with a monochromatic camera. This means that each intensity can be mapped to the position on the cmos sensor, and the system can "stare" rather than "scan." Eliminating moving parts is a major benefit in projects requiring sensitive optical alignment, making this a feature worth seriously considering.

Figure 4: Chlorophyll Flourescence Spectrometer

This project produces a design with similar goals to the Auto Garden Bed, because it features a light source, a target object, focusing optics, wavelength selection, and generating and interpreting a spectrograph. However, the optical regime of UV rays may be limited to fluorescence spectroscopy and unfit for proximity soil sensing. The research will have to determine what sensors are viable and whether this imposes further constraints on the system.

3.1.3 Smart Garden Controller

The Smart Garden Controller was a UCF senior design project with very similar motivations to the Auto Garden Bed. The goal was to create a system that reduced user labor by automating the irrigation schedule of garden areas and reducing waste

by sensing moisture levels[7]. The system was also designed to anticipate the informational needs of the gardener and come prepared with a set of popular vegetables and plants, corresponding to variables the team had already investigated to maximize flourishing. It used sensors to detect the health of the plant environment, a microcontroller to facilitate irrigation, and a web API to interface to the user for control of the system. The group also set goals to offer ease of setup, competitive price, and Smart Device integration by integrating with a device like Amazon's "Alexa." The last similar design features are wall plug power consumption, lithium ion battery backup power, and http secure information protocol over wifi.

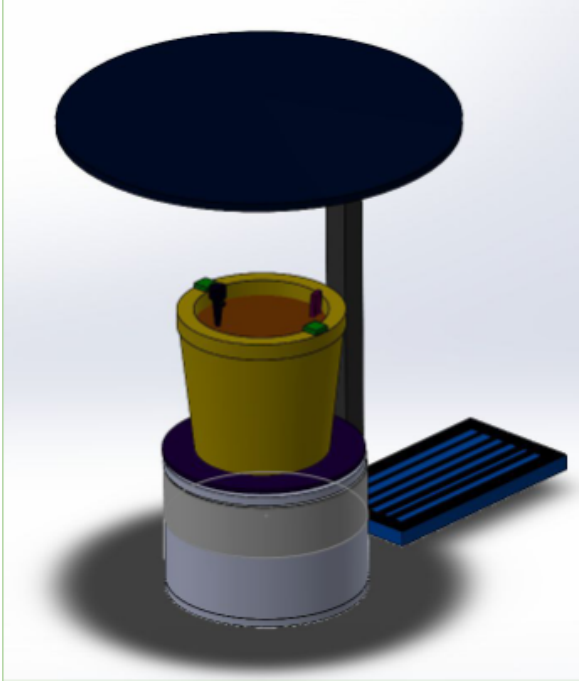
Figure 5: Smart Garden Controller

This project features much of the same core functionality as the Auto Garden bed, however, closer review of the scope and feature set reveal that the designs are incompatible. The Smart Garden Controller was designed to facilitate more precise control of an entire garden, allowing the user to isolate different areas for growing different plants. The area of interest was 100 squared feet, and the system was intended to relieve the gardener of the complication of watering each section for different amounts of time. This meant a wide reaching irrigation network as well as a network of sensors throughout the yard. The scope of the Auto Garden Bed allows for single source irrigation and sensing, which allows for greater flexibility of mechanical design, including power generation and environmental controls.

3.1.4 Automated Rotating Solar Plant Rack with Self-care Capabilities

This is another home garden system with a different target solution. The goal is to facilitate the growth of indoor plants and ensure maximum plant health by rotating the plant base so that the forces driving it to grow sideways cancel out in every direction. Like previous projects, the system is integrated with wireless communication and a schedule selector for different plant types. Unlike previous projects, one of the means of achieving plant health is by protecting sensitive plants from excessive exposure to sunlight, which is a feature that the Auto Garden Bed seeks to implement.

Figure 6: Automated Rotating Solar Plant Rack



Another point of interest in the Automated Rotating Solar Plant Rack is its rotating foundation. Maximizing sun exposure is a goal of this project, and although the application is energy collection, there may be parallel design opportunities with the Solar Plant Rack.

3.1.5 Stem 'n' Leaf

Stem 'n' Leaf is a UCF Senior Design project. The focus of this project was to be a "modular hydroponics system" wherein each plant unit can be fed my a singular control unit. Thus, the "stem" is the control unit and the "leaves" are the plant units[8]. Their design was focused on modularity and scalability. Hence their design is stackable and tileable design of the bed itself. The key features of this plant bed is that it self-regulates pH and integrates with a mobile application.

Figure 7: Stem 'n' Leaf

The project uses a liquid pH-sensor to obtain the relevant data. Our team wants a similar result but will be trying to achieve this via optical means. Their team mentioned the durability of the sensor which is our team's chief concern seeing as acidic and basic solutions tend to be corrosive and promote oxidation of metals thus the optical approach should improve longevity of the project.

3.1.6 Green Steel Garden

Green Steel Garden is another UCF Senior Project that our team is pulling some inspiration from. Another hydroponics system that measures soil nutrients and pH for controlling the parameters of the garden bed[9]. The key feature of this project is the nutrients system and the pH sensor.

Figure 8: Green Steel Garden

This project utilizes a water reservoir and reservoirs of chemicals to balance the pH of the water entering the garden bed. They use peristaltic pumps to accomplish the mixture and a combination of pH and electrical conductivity sensors to get data from the water. The current design consensus for our team is that we will be hooked up to hose which is limited by a solenoid valve, but being able to "treat" the water that is entering the system may be a design decision that has to be considered.

3.1.7 Summary

The previous works discussed here highlight the key components of our garden bed: sensing in conjunction with optics. Something of note in these deliberations is the use of hydroponics in all of these projects; something that is rendered unnecessary and seemingly inefficient in an outdoor garden bed.

The previous works related to optics focus on wavelengths outside of our team's consideration (Infrared), instead focusing on UV and visible light through near-Infrared. Their research highlighted potential flaws in some assumptions we had made. Firstly, that it is possible to reduce the moving components of our optical sensing, obtaining a wider field. Secondly, that using IR spectroscopy is practical for getting data from soil as it is used in commercial projects today and shows tremendous value in commercial sectors.

Most of the other garden bed projects focus on hydroponics but something of interesting note is the rotating solar rack project. The control scheme regarding the degrees of rotational freedom and the means of achieving success will be paramount in implementing our own moving solar array. Of interesting note is that all the other garden bed projects focused primarily on hydroponics as it gives greater measures of control to the soil nutrients which had not been a consideration of our team before looking into the previous works. Instituting a chemical pump into our design may be necessary to achieve our goal of being "set it and forget it" while still providing updates for when to refill the chemical tanks of integration with the water.

All of the garden projects tried to integrate some means of a mobile app or web user interface. They all mentioned that they needed to have started this component earlier. Each of the mobile application projects were not able to achieve this by their

final meeting which is of special concern. Also of note is their choice of stack; many of the projects chose a NoSQL stack which seems counterintuitive given the highly relational nature of the data that is being collected. Our team can learn from this by starting this integration as early as possible and choosing to implement the smallest possible feature set that accomplishes the goals.

3.2 Related Technologies

It is almost always beneficial for developers of a product to research and study related or similar products, as well as products related to their product's subsystems, in detail to gain a better understanding of how a subsystem can operate, and how subsystems can integrate with each other to form a cohesive system. In this section, members of our team explore technologies and solutions related to our product and its various subsystems. Some of the technologies studied below can be, and most likely will be integrated into our final product.

3.2.1 Ocean insight: Ocean ST NIR Microspectrometer

Ocean Insight is a local manufacturer of high-end, low size, weight, and power spectrometers. The ST NIR Microspectrometer is about 40 cubic centimeters in volume with a scan speed of 10ms, a signal to noise ratio of 190:1, and a spectral resolution of 2.2nm. Its spectral range is from 645nm to 1085nm, and it was specifically designed to be integrated into larger systems for customers who were interested in a flexible, low-cost design. Added to that, the system is rugged, and offers a variable slit input size, increasing its flexibility even further. While the designs are proprietary, this system serves as a benchmark for what can be achieved by the industry, and no doubt there are major design changes that can be made to achieve a similar result for the application intended for the Auto Garden Bed. That being said, the selling price for one is $1,750.

Figure 9: Ocean ST NIR Microspectrometer

3.2.2 AgroCares Nutrient Soil Scanner

AgroCares offers a Near Infrared Spectrometer specifically designed for Proximity Soil Sensing. Its spectral range is from 1300 to 2500nm and it uses Micro Electrical Mechanical Systems or MEMS to capture EM Waves reflecting off the soil. The real value of the product is in its wireless communications system. The device uses Bluetooth 4.0 to send data to a cloud data center. There, spectrographs of large data sets of soil with known nutrient contents are compared with the reading, cutting out the need for on-sight calibration. The system is handheld and uses eight tungsten halogen bulbs to blast the soil with energy. This light is collected in an extremely small area, sampling 65 squared millimeters. It would be worth researching to see if the Tungsten bulbs were linked to the 1300 to 2500nm spectral range or if another probe and sensor would suffice.

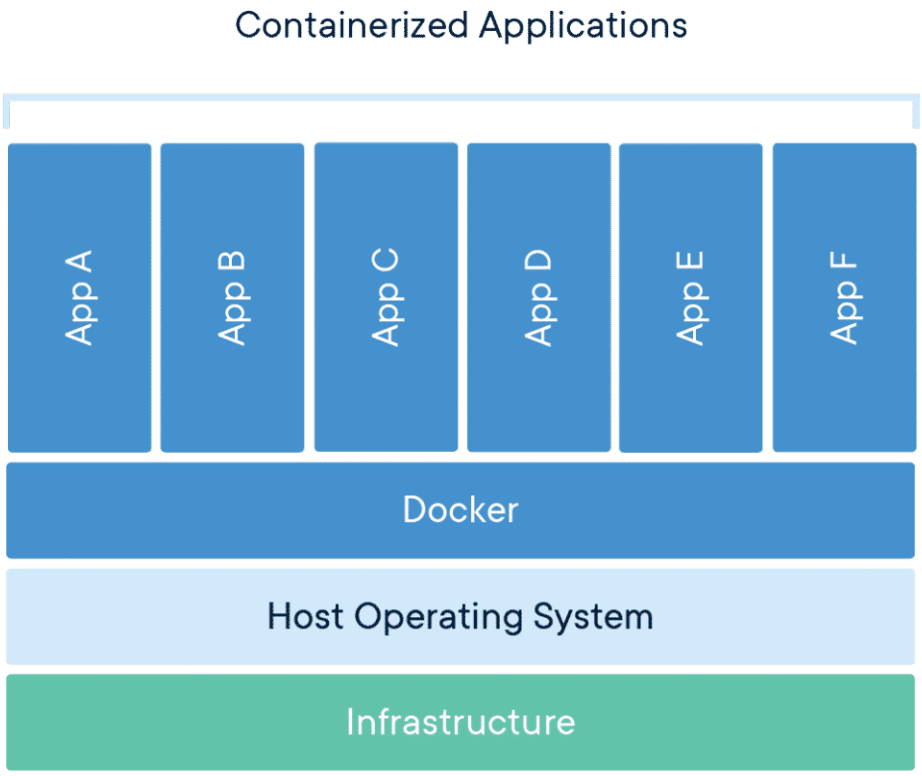
Figure 10: AgroCares Nutrient Soil Scanner

AgroCares offers a Near Infrared Spectrometer specifically designed for Proximity Soil Sensing. Its spectral range is from 1300 to 2500nm and it uses Micro Electrical Mechanical Systems or MEMS to capture EM Waves reflecting off the soil. The real value of the product is in its wireless communications system. The device uses Bluetooth 4.0 to send data to a cloud data center. There, spectrographs of large data sets of soil with known nutrient contents are compared with the reading, cutting out the need for on-sight calibration. The system is handheld and uses eight tungsten halogen bulbs to blast the soil with energy. This light is collected in an extremely small area, sampling 65 squared millimeters. It would be worth researching to see if the Tungsten bulbs were linked to the 1300 to 2500nm spectral range or if another probe and sensor would suffice.

3.2.3 Web Technologies

Docker "Docker is a platform designed to help developers build, share, and run modern applications. We handle the tedious setup, so you can focus on the code." This quote comes directly from Docker's website on why developers should switch to Docker [10]. This technology makes items more portable by making the executable platform agnostic through the use of a docker kernel and containers. The figure below should provide some clarity:

Figure 11: Docker architecture



A developer can program applications such as those demonstrated in figure 11 [11], package them into images and run them in containers. Docker works with the OS kernel to provide the same environment to the application each and everytime.

One of the largest advantages to using and implementing Docker that the team sees is that one team member can program and package the image and it should

"just run" on any other team member's machine. This fact will prove very useful in integration testing especially for the socket server detailed in Section 5.4.

User Interface Because these are web technologies we will be using Javascript frontend frameworks/libraries for creating the frontend. Based upon cursory research, the most feature-rich and most widely used are React and VueJS.

React React is a component-based library for building user interfaces. Many libraries have extended the functionality by adding components to React for use by other developers. A React component is a stateful element in the Document Object Model (DOM). Essentially, based on the state information, a component is rendered in raw HTML to the browser. One such example of a component library is MaterialUI (MUI). MUI is a popular library for adding components like hamburger menus, tables, graphs, etc. Choosing React decreases the time spent developing due to the ease of tossing boilerplate components at the problem. The biggest issue is that all rendering is done in the browser which means that loading an uncached page may take a long time. Frameworks built ontop of React such as NextJS help speed up this process by moving some of the processing to the server.

VueJS VueJS is a framework for building user interfaces. The difference between a library and a framework is that a framework provides a control flow while libraries are just used. VueJS is not too unlike raw HTML and JavaScript wherein `<script>` tags are used to create dynamic pages in response to user actions. The key difference between Vue and the above is that Vue provides the access to component-based programming. Similar to React, Vue works on the basic of components but it uses the default HTML DOM and adds functionality to pre-existing components. This comes with the limitation that components are not nearly as interleaved with data from a web server. Vue is performant and designed for single-page applications, which may not server this project well in the long-run.

Web and Socket Server This section will host information related to technologies for building out a web and socket server for connecting the user interface to the backend. .NET, Java and JavaScript all have reasonable solutions to these but based on the JavaScript user interface, this discussion will be kept to Java and JavaScript solutions and technologies.

Java Spring Boot Java Spring Boot is a full featured library that has different webservers embedded such as Apache Tomcat. Web servers are the means for which an application can be accessed from the outside world. Java Spring Boot makes this process incredibly easy. Part of this project will be communicating via TCP packets as well as through HTTP requests. Java at first glance seems like the easier way

of accomplishing both tasks through the `java.net` libraries and through the use of Spring Boot `@Controller`.

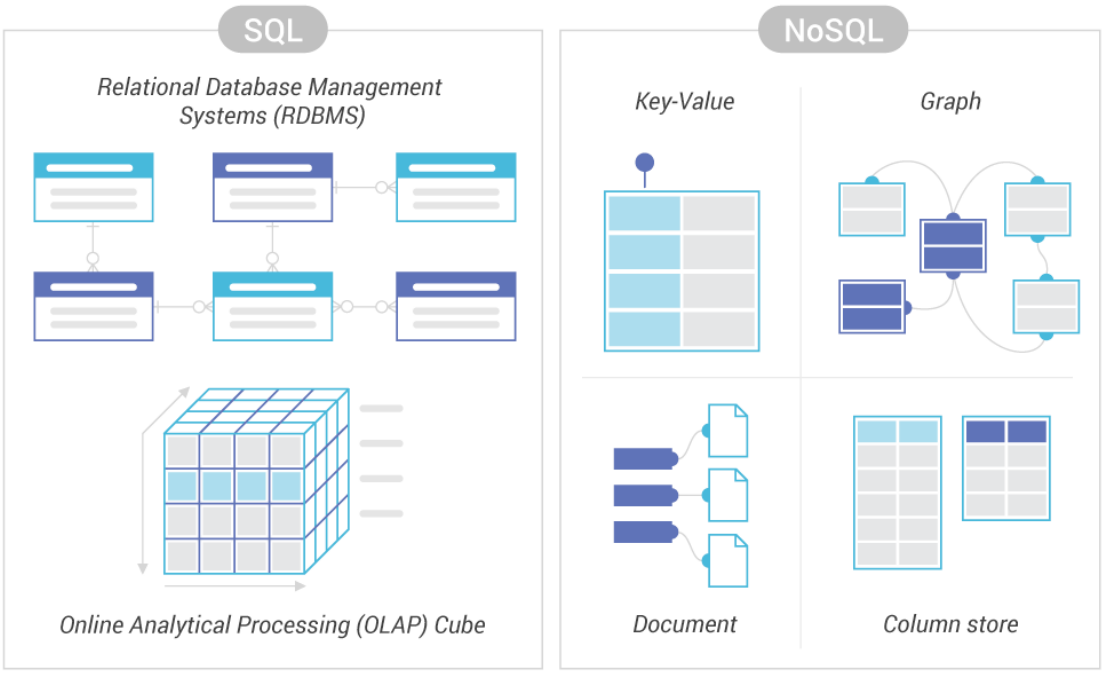
The `java.net` libraries are essentially a port of the network protocols that were made in the C language for creating communications via packets. By deploying using Java Spring Boot, it is possible to create two services packaged into one, the socket server and the web server. Java's memory management and VM environment leave a lot to be desired. The current state of the language and its artifact leave a lot to be desired as well. Because of the nature of running as a server and potentially servicing multiple plant beds, Java's lack of callbacks and lacking multi-threading support means it is downgraded given its overhead.

The `@Controller` is such a great feature in Spring Boot. Java is strictly typed, leading to a lot really helpful features in executing backend requests. Also, because of the overhead, Java ORMs tend to be more fully featured and have support for a variety of other tools that increase velocity when programming. One such tool is Liquibase. Liquibase is a database changelog tool for creating and implementing database migrations based on a code.

ExpressJS Express is a lightweight backend framework for implementing routes and middleware in JavaScript. The biggest disadvantage of using Express is the lack of low-level capabilities. Express was designed to be multiple layers of abstraction away from the kernel which may prove to be tumultuous when trying to create a socket server that communicates with the MCU.

Databases A database is essential for tracking data and persisting it for use later. There are two main types of database, SQL and NoSQL. With these two types of database comes a variety of implementations and on top of that a variety of tools to work with them. Let's compare SQL vs NoSQL first.

Figure 12: Difference between SQL and NoSQL



SQL vs NoSQL In Figure 12 you can see the abstracted way to think about these two different systems. SQL records are exactly that, a record at a point in time. If one entity encapsulates another then another table holds that information. In NoSQL, the information is abstracted to a document with key-value pairs to get specific data. These documents can make references to other documents but for time-ordering this is highly ineffective. SQL records are highly effective for logs and indexing a large amount of data based on a higher level entity such as user that has many garden beds; garden beds that have a lot of data, etc.

MySQL vs PostgreSQL The two SQL databases widely used in industry are MySQL and PostgreSQL. MySQL is touted as "a simple relational database [... that is] very efficient and user-friendly" while PostgreSQL is widely used in data analytic and scientific applications because of the extensibility, scalability and object models. Immediately this seems like the better option but PostgreSQL may be harder to stand up immediately. Special consideration should be given to AWS solutions as well.

MongoDB vs DynamoDB MongoDB is "a general-purpose, document-based" database. DynamoDB is AWS' proprietary solution to NoSQL databases. DynamoDB is more difficult to work with as it is the newer service, it could also potentially be more expensive over time, however, DynamoDB has improvements over MongoDB for indexing and building out reference documents.

3.2.4 Proportional-integral-derivative Control

Proportional-integral-derivative control (PID control) is a common control algorithm (espectially in industrial control systems) to allow a control loop to have reliable performance in a variety of conditions. Simply put, this algorithm allows a controller to receive an input and calculate a proportional output, accounting for error and rapid changes in the process. This algorithm may be useful to our application because it will allow the product to better control its facilities without user input.

The control function is defined in Equation 1, where $u(t)$ is the control variable (e.g. the garden bed's water control solenoid), K_p, K_i, and K_d are gain factors, and $e(t)$ (the error) is the different between a desired setpoint and measured process variable (e.g. the difference between the desired and current ounces of water dispensed).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

Proportional Term The proportional term of Equation 1 is $K_p e(t)$, hence referred to as the P-term. The P-term is proportional to the current error, and the gain K_p determines the magnitude of the P-term. If the gain is too large, then the process variable will oscillate.

Integral Term The integral term of Equation 1 is $K_i \int_0^t e(\tau)\,d\tau$, hence referred to as the I-term. This term accounts for previous values of $e(t)$ by taking the integral of the error, and the gain K_i determines the magnitude of the I-term. The I-term aims to account for residual error in the control loop.

Derivative Term The integral term of Equation 1 is $K_d \frac{de(t)}{dt}$, hence referred to as the D-term. The D-term aims to control future values of $e(t)$ by taking the derivative of the error, and the gain K_d determines the magnitude of the D-term. This term acts to damp rapid changes in the control loop. Higher values of gain may make the control loop more sensitive to noise and lead to instability.

3.2.5 Arduino Impletmentation of Microcontroller Internet Connection

Our team ultimately decided to implement a Texas Instruments microcontroller (detailed later) in the controls subsystem. This TI MCU contains an integrated network stack, and offers simple directions on connecting a compatible 2.4 GHz antenna. However, the one drawback our team did not expect was the relative abstractness, complication, and bureaucracy of creating programming with the tools required by TI. Between their proprietary distrobution of Eclipse, the libraries and SDKs required by the compiler, the configuratio of the compiler and linker, and the relative abstractness and complication of their libraries located in the SDK, the TI MCU is difficult to work with if the developer does not have pervious experience. In this section, Arduino's implementation is investigated and compared to our current TI MCU.

Arduino Arduino is a microcontroller development board (integrating ATmega microcontrollers) distributor with a focus on hobbyists, especially entry-level developers. Compared to Texas Instruments, an industry-focused manufacturer, Arduino development has a very low bar to entry. Because of this, the fine granular control that may be required of an enterprise-level project is not present on Arduino boards—however, there are major advantages that make considering Arduino over TI worthwile:

- Libraries are easy to implement
- The compiler and linker are relatively easy to configure compared to TI
- Large community following, support
- 3rd-party libraries are common and accessible
- Lower cost of components
- Easier to source components (in the year 2022)
- "Shields" (e.g. WiFi, GSM/GPRS, Bluetooth, GPS, motor controller, etc.) easily implementable

GSM/GPRS One shield offered by Arduino is the Arduino MKR GSM 1400, a 3G cellular network shield that enables SMS, voice, and internet connection. Arduino's library gives various "from scratch" examples and ample documentation on how to use different parts of its 1st-party GSM library. For example, if one would like to connect a GSM network, it's as simple as including `GSM.h`, instantiating the `GSM` class (e.g. `GSM gsmAccess;`), and calling `begin()` (e.g. `gsmAcess.bein()`).

WiFi Arduino offers shields like the Arduino MKR WiFi 1010, as well as complete development boards like the Arduino Uno WiFi Rev2 with integrated WiFi modules. Just like the GSM library, Arduino's WiFi library is very easy to implement and use, and its documentation is wholly informative with class and function definitions and plenty of examples. For example, all one need do to connect to a network is detailed in the example in Figure 13.

The user can go on to perform many different functions, with the limitation that they are either TCP or UDP bytestreams. This differs greatly from the TI CC3200's multifunctionality of being able to perform HTTP requests, Websockets, and many more on top of being able to take advantage of TCP and UDP bytestreams.

3.3 Part Selection

In our research for part selection we wanted to create a generalized comparison of the options we have available. We formatted this information in the tables below.

Figure 13: Arduino WiFi code example

```
#include <WiFi.h>

char ssid[] = "exampleNetwork";
void setup() {
    while (status != WL_CONNECTED) {
        // Attempt to connect to SSID
        status = WiFi.begin(ssid);

        // Wait 10 seconds
        delay(10000);
    }

    // Once you reach here, you're connected
```

3.3.1 Controller Subsystem

Minimum Requirements At minimum, any chosen microcontrollers (MCUs) shall support natively, or by addition of a module, these features and traits:

- Ability to connect to external antenna
- Analog-to-digital converter (ADC)
- IEEE 802.11
- In stock and available to order
- JTAG module or equivalent
- Module communication bus (UART, I2C, SPI)
- Network stack
- Onboard CPU sufficient for our purposes
- Onboard memory sufficient for our purposes
- Onboard nonvolatile memory
- Sockets support
- Pins dedicated to analog input
- Pins dedicated to digital I/O

Nice To Have These features would be "nice to have" on any MCU selected, but are not required:

- Digital-to-analog converter (DAC)
- Integrated antenna
- microSD card slot
- Onboard battery
- Pins dedicated to pulse-width modulation (PWM)
- Timer(s) and an RTC
- USB compatibility
- Additional wireless communication protocols (e.g. BT or BLE, Zigbee)

Selections The selections, listed in Table 2 and not in any particular order, match the above criteria and are being considered for selection.

Single-board Computers Use of single-board computers (SBCs) was considered, but will not not need to be used; sockets will be used on an MCU in conjunction with Amazon EC2 services will allow us to offload computing to a cloud solution.

External WiFi Module Use of an external WiFi module is discouraged due to the following reasons:

- Added cost
- Added complexity
- Modules in common use by hobbyists often have poor or no proper documentation, to the extent of:
  - Quick start guide
  - User's guide
  - Datasheets
  - Theory of operation
  - Application uses
  - Troubleshooting guide
  - Schematics and mechanicals
  - Quality and reliability

Table 2: MCU option breakdown

Model	LAUNCH-XL-CC26X2-R1	LAUNCH-CC3220-MODASF	Pico W	Nano 33 BLE	B-L4S5I-IOT01A
Manufacturer	Texas Instruments	Texas Instruments	Raspberry Pi	Arduino	STMicroelectronics
Microcontroller	CC2652R	CC3220-MODASF	RP2040	nRF52840	STM32-L4S5VIT6
Processor	1x ARM Cortex-M4F	1x ARM Cortex-M4	2x ARM Cortex-M0+	1x ARM Cortex-M4	1x ARM Cortex-M4
Maximum Speed (MHz)	48	80	133	64	120
Memory (KB)	256 ROM, 352 flash, 100 SRAM	1024 flash, 256 RAM	16 ROM, 264 SRAM	1024 flash, 256 SRAM	2048 flash, 640 RAM
Wireless capability	BLE5.2, Zigbee, Thread	802.11b/g/n	802.11n	BLE5.3, Zigbee, Thread, Matter	BT4.1, 802.11b/g/n, NFC
Serial capability	UART, I2C, I2S, SPI	UART, I2C, SPI	UART, I2C, SPI, USB1.1	UART, I2C, I2S, SPI, USB2.0	UART, I2C, SPI, USB2.0
Price ($)	40, maybe free	60, maybe free	6	28	53
ADC	8-channel, 12-bit	4-channel, 12-bit	4-channel, 12-bit	8-channel, 12-bit	16-channel, 12-bit
Watchdog timer?	No	Yes	Yes	Yes	Yes
GPIO (pins)	31	29	30	13	16
PWM (channels)	Supported	Supported	16	4	6
Required voltage (V)	1.8 – 3.8	2.3 – 3.6	1.8 – 3.3	4.5 – 21	4.75 – 5.25

Table 3: Differences between CC3200 and CC3220

Model	**CC3200**	**CC3220**
Microcontroller	CC3200	CC3220SF
Secure Flash (MB)	n/a	1
Simultaneous TCP/UDP Sockets	8	16
WiFi Receive Sensitivity (dBm, 1 DSSS)	-95.7	-96
WiFi Receive Sensitivity (dBm, 54 OFDM)	-74.0	-74.5
Hibernate Current Draw (μA)	4	4.5

– Errata

Therefore, all of the MCUs listed above support either the 802.11 or Bluetooth standards.

External ADC Module Use of an external ADC was considered, but ultimately decided against. An external ADC module would only add further cost and complexity to a project where the on-chip ADC is sufficiently effective for the project's requirements. The sample rate and resolution of an on-chip ADC is more than enough for our needs, and the low-cost goal of the project directly conflicts with the idea of purchasing an external ADC module.

Selection Ultimately, the LAUNCHCC3220MODASF was chosen as the microcontroller development board for this project. In the event that the aforementioned LaunchPad is not able to be obtained, the CC3220SF-LAUNCHXL has quivalent capability for the project's needs.

These boards, henceforth referred to as the CC3220, are able to be requested from our university at no upfront cost to our team. This was the driving factor behind choosing the CC3220 over other microcontroller development boards. It was also determined that the microcontroller *must* be able to interface via the 802.11 (WiFi) standard, for reasons that are detailed in subsection 5.1—therefore, the LAUNCHXL-CC26X2R1 and Nano 33 BLE were disqualified from selection. The Pico W was considered due to its low cost, and the B-L4S5I-IOT01A considered because of its abundant peripherals, but both ultimately lost out to the Texas Instruments products.

CC3200 Due to difficulties in acquiring the CC3220 (second-generation SimpleLink device), our team made the decision to use the CC3200 (first-generation SimpleLink). These parts are mostly identical, with the differences detailed in Table 3. Development will take place on the CC3200-LAUNCHXL, with the PCB MCU part to be CC3200 (if needed).

Table 4: Controller subsystem bill of materials

Qty	Cost ($/ea)	Manufacturer	Item	Model Number
1	0.00	Texas Instruments	SimpleLink Wi-Fi CC3200 LaunchPad	CC3220-LAUNCHXL
2	10.88	Texas Instruments	CC3200	CC3200R1M2RGCR
2	3.80	Taiyo Yuden	Surface Mount 2.4 GHz Antenna	AH316M245001-T
2	6.95	Adafruit	Plastic Water Solenoid Valve	997

Bill of Materials The bill of materials for the controller subsystem can be found in Table 4.

3.3.2 Power Subsystem

Power Supply The power system was a key factor in this model, as we attempted to make it an independent system. This was crucial because the power system needed to be able to power all the different components while also charging itself when not operating.

As part of our goal to make the system independent, solar energy played a significant role. We needed the solar panels to operate first before powering other components. The key parts of the system included solar panels that converted light energy into electrical energy, a solar charge controller to regulate output voltage from the solar panel into the battery, and a battery to directly power the other components in the model.

We had many different sensors, electrical, and mechanical components in this model, all of which required power. To design the power system flow and operation, we first determined the total power needed for the entire system to run. This was determined by finding the individual power ratings of each component and calculating them together. Once that was done, we chose the type and quantity of battery needed for the model, taking into consideration how long we wanted the system to run, optional secondary power sources, and optional battery banks. We researched different types of solar panels, such as their efficiency and power rating. We then selected a solar charge controller, a device that sat between the solar panel and the battery to regulate how much power went into the battery. After that, we determined the voltage regulator to regulate the battery's output voltage to power the smaller components. Once we had completed all of that, we were able to find other components that would make the power system more reliable and efficient in any way.

Power Requirement Sensors, electrical, and mechanical components all required power but all consumed different amounts of power. As mentioned before, analyzing the total power requirement was important and crucial because this helped us determine the right parts that were best for this system and for the components.

With the total power determined, the Watts per hour needed for all of the components to run had to be found. The watts per hour were important too because they helped figure out how long each component would operate for. After that was found, we used the altE calculator to help pick out what kind of battery could be used, then solar panel and solar charge controller, respectively. The altE calculator was a great resource because with the proper measurements, it could help us choose what kind of battery we could use, by determining the capacity needed in watt-hours or amp-hours. This calculator could also help us choose how big of a solar panel we needed and how big of a solar charge controller we needed as well.

Rechargeable Battery Selection Solely relying on solar energy wasn't always ideal. This was because the weather may not always guarantee sunlight, which hindered its power retention. For this reason, solar panels were paired with a battery so that the power could be stored and then used at a later time. As part of the goal to have this run as an independent system, an additional battery could be used so that one battery could power the system while the other one could charge.

There were many different types of batteries including Nickel-Cadmium, Nickel-Metal Hydride, Lithium ion, etc. Of the three batteries, they were compared to see which would best fit their model and which would fulfill the requirements on the basis of power output, efficiency, etc.

Nickel Cadmium (NiCd) batteries in today's time were used for RC vehicles, power tools, photography equipment, and more. They were also considered as old technology. Even though they were old, they still had their advantages such as being less expensive, they were super powerful and charged fast, they required little maintenance, and more. These batteries though also had their disadvantages, one being they suffered from " memory" problems and as a result of that, it may reduce that capacity of charges and future battery life. These batteries were also environmentally concerning because cadmium was toxic.

Nickel-Metal Hydride (NiMH) was similar to nickel cadmium, the only difference was that hydrogen was used instead of cadmium as the active element. Hybrid cars, toothbrushes, and phones were just a few of many products that used nickel-metal hydride batteries and had been used in these appliances because of the trouble-free service they granted. It was also because even being partially discharged, it could be charged as many times and would always be at full capacity. Even with advantages like that nickel-metal hydride batteries produced a lot of heat when in use, had a high self-discharge rate, and had memory issues as well, just not as bad as NiCd.

Lithium ion batteries were one of the most popular types of rechargeable batteries for portable products. They were considered the best because lithium ion batteries had high open circuit voltage, low self-discharge rates, and little to no memory effect.

On top of that, they were growing within the military, electric vehicle companies, and aerospace industry with little to no maintenance. Even though they had a lot of advantages, some of the disadvantages included sensitivity towards high temperatures, it could not be fully discharged, and the cost.

Through much consideration and investigation, the four batteries listed in the following table were picked. All of which were 12V lithium iron phosphate (LiFePO4) batteries. Then the selected battery that we decided to use for the model was the Eco Worthy 12V 8Ah LiFePO4 battery. We decided to go with this battery because although it was a little expensive, the Watts per hour and the Ah rating that this battery provided, was great for what we planned on having.

Table 5: Battery Selection

Manufacturer	**Ampere Time**	**Eco Worthy**	**Expert-Power**	**Eco Worthy**
Voltage	12	12	12	12
mAh	6000	10000	5000	8000
Watt per hour	76.8	120	64	96
Cost	$29.99	$59.99	$35.99	$43.99

Solar Panel Selection Solar has been a growing source of energy in the past years, with new developments and breakthroughs in solar cell technology. As the whole purpose of solar energy was to collect sunlight and convert it into electrical energy, that was the minimum for this model to run as an independent system. Then, as a stretch goal, the concept of solar tracking panels was applied to create blinds with the solar panels to open and close according to the position of the sun.

On a basic level, solar panels are made of solar cells and these cells do the collecting and converting. These solar cells are made from crystalline silicon that was melted down into ingots and then cut into sheets. In the solar industry, there are 3 main types of solar panels. These types of panels were monocrystalline, polycrystalline, and thin-film panels all of which had different compositions. Each type has different efficiencies, generates different amounts of power, etc.

Monocrystalline solar panels are solar panels that are made with monocrystalline solar cells. These solar cells are composed of a single silicon crystal which provides electrons more space to move because of the electricity flow that is generated. This makes them more efficient, yet at the same time more costly. Polycrystalline solar panels are solar panels that are made with polycrystalline solar cells. Similar to monocrystalline solar cells, they are made of a silicon crystal, the only difference is that instead of a single crystal, they use several fragments of silicon to form an ingot that is cut into sheets. As a result of melting several fragments into one, this creates a mosaic look as well as giving it a blue hue, whereas the monocrystalline

solar panel has a uniform color and look. Aesthetically, they may look nicer but they are less efficient than monocrystalline solar panels. This also meant that they weren't going to be as pricey compared to the monocrystalline panels because of the efficiency and manufacturing process. Thin-film solar panels differ from crystalline solar panels greatly, all because they are made with different materials. The three main types of thin-film solar panels are amorphous silicon (a-Si), cadmium telluride (CdTe), and copper indium gallium selenide (CIGS). Currently, thin-film solar panels are the least efficient, costly, and have the shortest lifespan. It was predicted that they would have a major growth in the solar industry because although they were the least efficient, they had a higher theoretical efficiency than both monocrystalline and polycrystalline.

The choice of solar panels was based on multiple aspects of each type of solar panel. As it was known, the efficiency rating and cost from most to least went from monocrystalline, polycrystalline, and thin-film, respectively, but it was also necessary to look at its temperature coefficient, power rating, and more to be able to determine which solar panel was exactly needed. Temperature coefficient for solar panels was the power lost as the temperature rose. This played a very important role because in Florida temperatures could get hot. So, when it came to which solar panel was still more efficient in higher temperatures, monocrystalline solar panels were still the best. As follows, it then went from polycrystalline and then thin-film. It could also have been more cost-efficient as well because the other two types of solar panels were less expensive.

Table 6: Solar panel types

Solar Panel Type	**Monocrystalline**	**Polycrystalline**	**Thin - Film**
Efficiency	>20%	15 - 17%	6 - 15%
Power Rating	≤300W	240 - 300W	Indefinite
Performance	Most efficient	Efficient	Least efficient
Temperature	High Tolerance	Low Tolerance	High Tolerance
Cost per Watt	$1 - $1.50	$.70 - $1	$.43 - $.70

After conducting research and investigation, we compared different types of solar panels to determine the best fit for our model. We did not specifically choose a particular type of solar panel as all the options we considered had their benefits. However, the monocrystalline solar panel was a popular choice among the four types we evaluated. All the solar panel choices we assessed had a power rating of 10 Watts and a voltage rating of 12V, except for the Voltaic solar panel which had a voltage rating of 18V.

We ultimately selected the Eco Worthy 10W 12V monocrystalline solar panel after considering various factors. The cost of the solar panel was reasonable, and it provided excellent value for money. Its size and weight made it easy to move around,

and it had received numerous positive reviews. Additionally, it was readily available for purchase on various websites.

Table 7: Solar panel part breakdown

Sku	P108	L02M10-1	NPA10S-12H	SLP010-12U
Manufacturer	**Voltaic Systems**	**Eco Worthy**	**Newpowa**	**SolarLand**
Solar Panel Type	Monocrystalline	Monocrystalline	Monocrystalline	Polycrystalline
Dimensions	10.9 x 8.8 x .16	13.3 x 8.1 x .7	14.37 x 7.68 x .91	14.06 x 11.89 x 1.18
Peak Current	570mA	580mA	630mA	580mA
Open Circuit Voltage	20.45V	20.6V	19.83V	21.6V
Peak Voltage	17.34V	17.3V	16.77V	17V
Wattage	9 watt	10 watt	10 watt	10 watt
Power Tolerance	±10%	±3%	±3%	±5%
Cost	$49	$25.99	$25.99	$35.53

Solar Charge Controller Selection Solar charge controllers played an important role in this system because the system ran on solar energy. A solar charge controller was a regulator that went in between the solar panel and the battery, regulating the total output power coming out of the solar panel. This was important because it prevented the battery from overcharging and possibly reducing its effectiveness. If the wrong charge controller had been picked, it could have resulted in a loss of power that was generated and could have harmed any device. As stated in the power requirement section, finding the total power needed for the whole system was important because it helped determine the proper charge controller to get.

There were two main types of solar charge controllers: Maximum Power Point Tracking (MPPT) and Pulse Width Modulated (PWM). Choosing the right charge controller was based on current and voltage characteristics. This was because they regulated input voltage coming in from the solar panel and output voltage of what was coming out to the battery.

Maximum power point tracking (MPPT) was a technique that observed and regulated energy coming from the solar panel and into the battery. What made this special was that it could match the solar panel voltage to the battery voltage, which allowed it to maximize the charge efficiency. They operated as a DC to DC converter,

taking in high DC input from the solar panel, changing it to high AC voltage, then back down to DC voltage.

Pulse width modulated (PWM) solar charge controllers were considered the original charge controller compared to the MPPT charge controller. They were less expensive and the technique behind this controller was also simpler. On a basic level, the PWM controller acted as an on-off regulator. When the battery voltage reached a certain level, the PWM controller slowly reduced the charging current, up until the battery reached the maximum amount of energy. This made it great for smaller installations because the solar panel and the controller could match better.

While looking at different solar charge controllers, we came across 4 different controllers, all of which were PWM charge controllers. We did not specifically choose the PWM controller but through our search for what charge controller we wanted to use all of the choices we found were PWM. This was great for our model regardless because our system was not large and it was less expensive compared to MPPT charge controllers.

The solar charge controller we chose for our model was the Eco Worthy 10A PWM solar charge controller. The reason we chose this controller was because it came as a kit along with the solar panel. As a result of it coming as a kit with the solar panel, they were already very compatible together, and the price for both of them together was not expensive at all.

Table 9: Charge Controller

Manufacturer	**Eco Worthy**	**Renogy**	**Expert-Power**	**Expert-Power**
Charge-Controller Type	PWM	PWM	PWM	PWM
Output Voltage	12/24V	12/24V	12/24V	12/24V
Rated Charge Current	10A	10A	10A	20A
Max PV Voltage	30V	50V	55V	55V
Self Consumption	$\leq$10mA	$\leq$10mA	$<$10mA	$\leq$13mA
Price ($)	23.99	69.99	34.99	69.99

Voltage Regulator Voltage regulators played an important role in almost every electronic device. All electronic devices operated at different voltage ranges, with some requiring a constant voltage. Common operating voltages were 3V, 5V, and

12V. To provide this voltage range or constant voltage, a voltage regulator was added into the circuit design. Voltage regulators helped regulate voltages during power fluctuations and different variations in loads, preventing damage to any component. They could also regulate DC and AC voltages. For this power system, we focused on DC voltage. This was because solar panels produced DC voltage.

As previously mentioned, voltage regulators played an important role in almost every electronic device. They were used for smaller devices to power components such as sensors, op-amps, and other modules, but could also be used in bigger applications such as TVs, automotive vehicles, industrial applications, and many more. There were two main voltage regulators: linear and switching. Both of these types had the same goal, regulating a system's voltage but operated differently depending on the application that they were used for.

Linear Voltage Regulators Linear voltage regulators, just as the name suggests, were a type of regulator where the linear and electrical components were placed in series with the input and output. The base of the linear voltage regulator was the use of an active pass device (such as a BJT or a MOSFET) which was controlled by a high gain amplifier. To maintain a constant output voltage, the regulator used a closed feedback loop to bias the active pass device. Linear voltage regulators were also known as step-down converters because the output voltage was always less than the input voltage. As the power was consumed and dissipated in the transistor and then was converted into heat while generating a constant output voltage. In the figures below, you could see the general schematic of the linear voltage regulator but also the pin-out diagram for a LM 7805 voltage regulator. As you could see, this type of voltage regulator had three pins for input, output, and ground.

Figure 14: General Linear Voltage Regulator Circuit Schematic

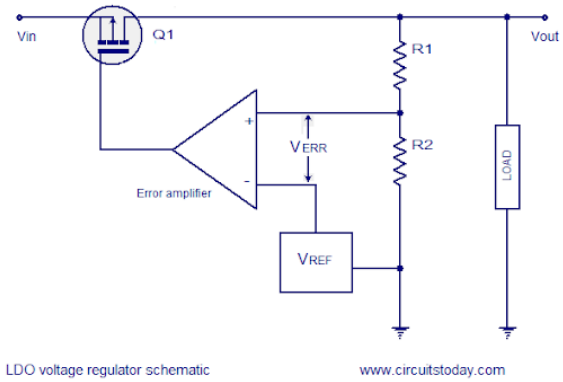
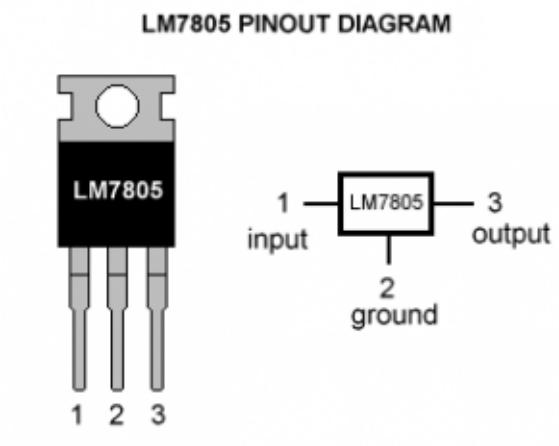


Figure 15: LM7805 Pin Diagram

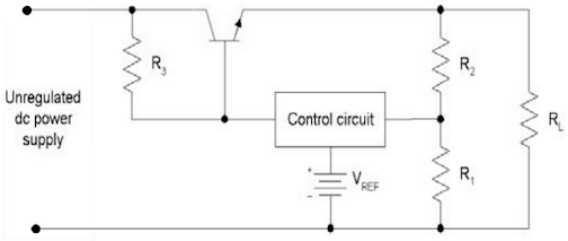


Just like any electrical device, linear voltage regulators had their advantages and disadvantages. Linear voltage regulators were highly integrated devices that were simple, cheap, and responsive to changes in input voltage and load voltage. They did not produce any switching noise, which was a problem with other voltage conversion circuits. However, their biggest disadvantage was their inefficiency, which was caused by the voltage drop across the active pass device and resulted in the regulator producing a lot of heat.

Just like voltage regulators could be broken down into linear and switching voltage regulators, linear voltage regulators could be broken down into different types as well. There were two main types of linear voltage regulators: series voltage regulator and shunt voltage regulator. The main difference between the two was that with a series regulator, the active pass device was connected in series, whereas in a shunt regulator, it was connected in parallel.

As the name suggested for a series voltage regulator, the pass element in the circuit was connected in series with the load, as shown in the figure below. Looking at the circuit, the output voltage was sensed through the voltage divider network in R1 and R2, and was compared to the VREF. As a result, the voltage drop across the transistor could be varied to ensure that the voltage across the load was constant.

Figure 16: Series Voltage Regulator

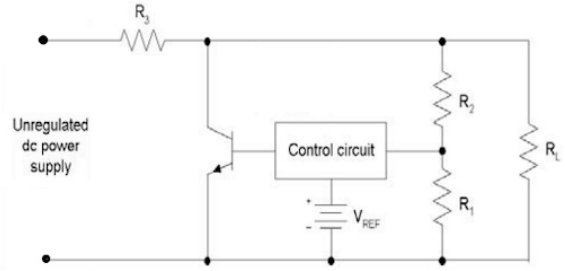


The main advantages of the series voltage regulator were that the current was

effectively used by the load, making it more efficient than shunt voltage regulators. However, the efficiency was still low compared to a switching voltage regulator, but it was simple and the output did not have any switching spikes.

The figure below showed the circuit schematic of a shunt voltage regulator. The pass element in the circuit was connected in parallel while the resistors were connected the same as in the series voltage regulator. For this voltage regulator, voltage was maintained through the current drawn through the resistors, and as the current was varied, the output voltage across the load remained constant.

Figure 17: Shunt Voltage Regulator



When compared to the series voltage regulator, it was just slightly less efficient, but was simpler to implement into the circuit. This type of linear voltage regulator was less common and was used commonly in low-powered circuits and in voltage reference circuits.

Switching Voltage Regulator Switching voltage regulators are a type of regulator that acted as a switch, where input power was turned on until the desired voltage was reached. After the desired voltage was reached, the switch element was turned off and stopped any input power from coming in. With this type of voltage regulator, switching noise occurred because of the high-frequency from the reference voltage and the amplifier. As a result of the noise, capacitors, inductors, and other electrical components were used to smoothen out and reduce the noise. Regardless of the noise that occurred with switching voltage regulators, this type still remained very efficient because of the process of switching on and off at such high speeds. As said before, this type of regulator repeated its operation at high speeds, which allowed it to supply voltage more efficiently and reduce heat generation. The figures below show what the general schematic of a switching voltage regulator looks like but also the pin-out diagram for an lm2596 voltage regulator. You can see the difference in how many pins the lm2596 voltage regulator and the lm7805 voltage regulator have. That is because switching voltage regulators require two more pins for the switch element as well as the feedback.

Figure 18: General Switching Voltage Regulator

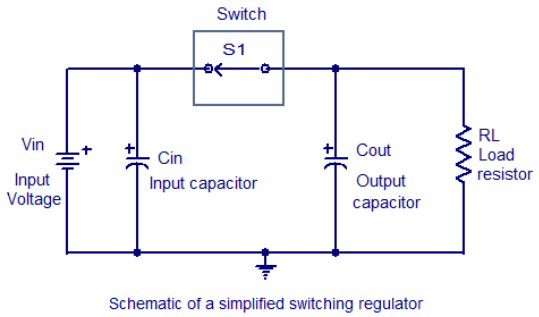
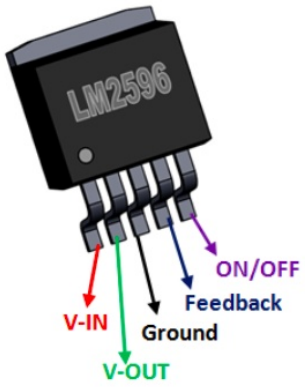


Figure 19: LM2596 Pin Diagram

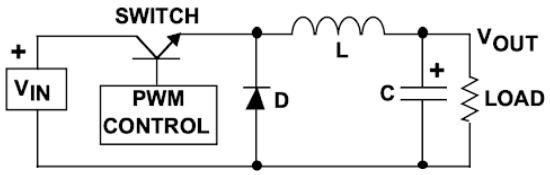


Some advantages of the switching voltage regulator were that it had a much higher efficiency, did not produce a lot of heat, and was capable of higher power efficiencies. The downside was that it had a more complex design, produced more noise, which required it to have more external components added to the circuit design. The choice between linear and switching voltage regulators always depended on what it was being used for because each design required different things, but switching voltage regulators were also a popular choice because of how efficient they were with power input and their general efficiency.

There were 3 main types of switching voltage regulators: buck converter, boost converter, and buck/boost converter. These three types of voltage regulators performed similar actions but all produced different output voltages.

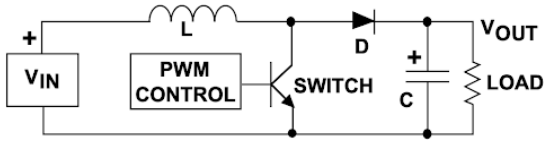
Buck converters, also known as step-down converters, lowered the output voltage than the input voltage. This was similar to linear voltage regulators because linear voltage regulators worked to make sure the output voltage was always lower than the input. The difference was that there was less waste. The figure below showed the circuit schematic of a buck converter. Normally, a transistor was used as the switching element, connecting and disconnecting the input voltage to the inductor.

Figure 20: Buck Converter (Step Down) Circuit Schematic



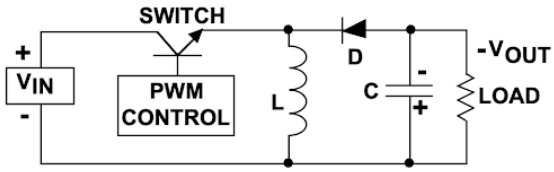
Boost converters, also known as step-up converters, provided a higher output voltage than the input voltage. Even though the output voltage could be higher than the input voltage, the power being provided still had to be regulated within the output power specification of the circuit. Figure 8 showed the circuit schematic of the boost converter. As shown, the components were switched around compared to the buck converter. This allowed the output voltage to increase, because when the switch was on, the voltage had to go across the inductor, increasing the current. When the switch was off, the diode was forward biased and charged the load higher than the input.

Figure 21: Boost Converter (Step Up) Circuit Schematic



The last type was the buck/boost converter and as the name suggests, this type of converter was able to supply either a larger or smaller output voltage, but could also invert the polarity. This type of voltage regulator was common with battery operated products because at first use the battery would supply full power but throughout time the battery as well as input power would depreciate. Figure 9 showed the circuit schematic of the buck/boost converter. This converter operated similarly to both the buck converter and the boost converter but differed because it was capable of inverting the polarity. This was done by forward-biasing the reverse-biased diode when the switch was off.

Figure 22: Buck/Boost Converter Circuit Schematic



This type of converter offered many benefits. One of the reasons it offered so much was because it brought both the buck converter and the boost converter together. It

also offered lower operation cycling and was more efficient between the input and output voltages. A few downsides to this were that there was no isolation between the input and output and the output was always inverted, which resulted in complex sensing and feedback in the circuit.

Voltage Regulator Selection The main function of a voltage regulator was to regulate output voltage from the power source, which was the 12V battery powered by solar panels, to the microcontroller and other components, ensuring that each part received the correct amount of voltage. The microcontroller we used, the Texas Instruments CC3200, operated at a 3.3V input voltage and as a result required a voltage regulator that could lower the output voltage.

As mentioned before, not all components had the same operating voltages. With the various components and voltage requirements, multiple types of voltage regulators were tested to see which one was the most efficient and best fit for our model.

For our system, we looked at the LD1117 series (Linear), LM317 (Linear), LM2576 (Switching), and LM2596 (Switching). Of the various types of regulators, these were picked because the required input voltage for the CC3200 was 3.3V. These different voltage regulators were compared based on efficiency, output voltage, cost, and any other feature that they might offer.

From the datasheets of each voltage regulator, we were able to make the comparisons in the table below.

Table 10: Voltage Regulators

Feature	**LD1117-series**	**LM317**	**LM2576**	**LM2596**
Voltage-Regulator Type	Linear	Linear	Switching	Switching
Operating Voltage	15V	3V - 40V	3V - 40V	4.5V - 40V
Output Voltage	3.3V	1.25V - 37V	3.3V, 5V, 12V,15V	3.3V, 5V, 12V
Output Option	Fixed	Adjustable	Adjustable	Adjustable
Operating Temp	0°C - 125°C	0°C - 125°C	-40°C - 125°C	-40°C - 125°C
Efficiency	≤62%	Varies	75%	73%
Frequency	N/A	N/A	53kHz	150kHz
Cost ($)	$0.90	$0.99	$3.69	$6.73

After comparing the datasheets, we agreed to select the LM317 (linear) and the

LM2576 (switching). We chose these regulators so that we could further observe and compare them to determine which one would perform best and be the most suitable for our model.

3.3.3 Sensing Subsystem

The sensing subsystem consisted of a Visible/Near Infrared Spectrometer.

The spectrometer consisted of several optical, electrical, and mechanical components:

- Silicon Photodiode
- Near Infrared Spectrum InGaAs Photodiode
- Linear Stage Actuator Carriage Rail
- Reflective Diffraction Grating
- Focusing Lens
- Fiber Patch Cable
- Fiber Collimator
- Tungsten Lamp

Silicon Photodiode The silicon photodiode was one of the two detectors in the system, responsible for covering the visible spectrum of the soil emissions. Silicon photodiodes are cheap and overdeveloped for this application, with much of the marketing emphasizing high speed current rise times and low dark current. Our spectrometer did not require either. Instead, it was useful to have a detector with a large surface area. This reduced the precision required of other optical elements in the system while retaining the flexibility to reduce the active area of the detector with a small aperture if needed. As always, low cost was favored.

Table 11: Silicon Photodiodes

Manufacturer	Model	Range (nm)	Area (mm^2)	Dark Current (nA)	Cost ($)
Newark	BPX 61	400-1100	7.02	2	13.06
Digi-key	ODD-1B	400-1100	1	0.2	15.60
Digikey Opto Diode Corp	ODD-5W	300-1100	5	1	14.50
Edmund Optics	PIN-3CD	350-1100	3.2	0.15	34.00
Thorlabs	FD11A	320-1100	1.21	0.002	15.69
Thorlabs	FDS100	350-1100	13	1	16.08

The BPX 61 by Newark had a substantially better size to cost ratio than other relevant devices on the market, however it also had a higher ambient current than all others. One option was to select it and then compensate by investing in more LEDs to illuminate the target area and boost the signal up over the higher dark current, or by investing in more circuitry to clean up and boost the signal. Another option was to forgo the larger surface area and choose the sensor with the least ambient noise, Thorlabs' FD11A. There didn't appear to be any standout choices aside from those two. All the other devices made compromises between their specs, while these two had standout specs, but required compromises in design. The BPX 61 offered the greatest flexibility, so it was chosen for the visible-spectrum photodiode.

Near Infrared Spectrum InGaAs Photodiode The NIR InGaAs photodiode was responsible for detecting the majority of the spectral area of interest in the soil spectrometer. Once again, the relevant specs were active area, cost, and dark current. Most NIR photodetectors were designed for high-speed free space optical communication, but this application was very forgiving in terms of sensor speeds.

Table 13: InGaAs Photodiodes

Manufacturer	Model	Range (nm)	Area (mm^2)	Dark Current (nA)	Cost ($)
Digikey Excelitas	C30617BH	800-1700	0.1	1	43.58
Digikey Advanced Photonix	0800-3111-011	800-1700	1.36	0.2	50.21
Thorlabs	FGA015	800-1700	0.018	0.5	63.00
Thorlabs	FGA01	800-1700	0.01	0.05	67.55
Edmund Optics	N/A	800-1700	0.07	0.03	88.00
Edmund Optics	N/A	800-1700	0.12	0.05	88.00
Edmund Optics	N/A	800-1700	0.3	0.3	94.00
Edmund Optics	N/A	800-1700	0.4	0.4	94.00

The Advanced Photonix 0800-3111-011, distributed by Digikey, had the highest active area, the second lowest cost, and an acceptable midrange dark current. This made it the obvious choice for the system's NIR detector.

Linear Stage Actuator Carriage Rail There were several methods of detecting broad spectral regimes. Scanning was much more viable than staring in this case, since staring detection required an array of detectors. This system was intended to keep costs low for the target user, and detectors were a disproportionately expensive part. Instead, this system had a pair of detectors mounted on a carriage which passed through the spatially separated beams of light. There were several important criteria for this part. It had to be able to take small, precise, increment steps in order for the detectors to capture the soil emission with high resolution. It had to have sufficient range of motion to cover the whole spectral range, from 400nm to 1700nm. And it had to meet the size, weight, and power constraints imposed by the rest of the system.

Table 15: Linear Stage Actuator

Manufacturer	Rattmmotor	TOAUTO	TOAUTO	Zeberoxyz
Model	T0601-50	T0601-100	T0601-101	SFU1605
Stroke Length (mm)	50	100	50	200
Step Angle (°)	1.8	1.8	1.8	1.8
Step Size (mm)	.005	.005	.005	0.025
Voltage (V)	24	24	24	N/A
Current (A)	0.8	0.6	0.6	1.6
Cost ($)	39.00	67.00	89.00	83.89

The ratio between the stroke length and step size needed to be well above 1,500 in order to meet the criterion for spectral resolution. Luckily, the product with the least range was well over that limit, at 10,000:1. That allowed cost to become the driving factor. The T0601-50, distributed by Rattmmotor, was just over half the price of the next cheapest option. This was the group's choice of scanner.

Reflective Diffraction Grating Diffraction was the operating principle behind the spectrometer. The soil emissions were composed of different wavelengths representing the chemical structure of the soil, and this was increased under illumination and then separated by diffraction. A diffraction grating separated incident electromagnetic waves of different frequencies by confining them into a tightly bounded region on reflection or transmission. This confinement caused the waves to disperse, like a prism would, with waves of higher frequencies experiencing greater dispersion. Transmission gratings were not very effective for weak signal diffraction, as in this application, because most of the optical power was concentrated straight through and not separated at all, with only weaker orders fully separating. Reflective diffraction gratings provided the best efficiency for soil spectroscopy. In order to prevent the beam from diffracting so far that it reflected straight back into itself, where it could not be collected, it was necessary to acquire a diffraction grating with grooves which were blazed at an angle, so that the surface normal of the grating had a greater angle than 90 degrees. The blaze angle corresponding to the 1000 nm diffraction regime sufficed. Next, it was necessary to calculate the angular spread of diffraction from 400nm to 1700nm off a diffraction grating from several incident angles. The first order diffraction must not overlap with the incident beam if it was to be measured. Increased size was also desirable, in order to maximize the number of wave-groove interactions and boost diffraction efficiency.

Table 17: Diffraction Gratings

Manufacturer	Model	Density lines/mm	Dimensions (mm)	Blaze Angle (nm)	Cost ($)
Thorlabs	GR13-0610	600	12.7x12.7x6	1000	76.58
Thorlabs	GR25-1210	1200	25x25x6	1000	125.77
Edmund Optics	Stock 43-745	600	12.7x12.7x6	1000	80.00
Edmund Optics	Stock 43-753	1200	12.7x12.7x6	1000	80.00
MKS Newport	33025FL01-520R	600	12.5x12.5x6	1000	155.00
MKS Newport	33025FL01-530R	1200	12.5x12.5x7	1000	155.00
ScienceTech	631-0024	1200	50x50x9.5mn	1000	500.00

The spectral regime was too wide to be able to reliably use a 600 nm grating without losing information in the infrared. Unfortunately, efficiency was not a variable that could be compared from product to product, since it varied with wavelength and per polarization. This meant the only remaining comparison was the ratio from cost to active area. ScienceTech was ruled out immediately. Its cost to size ratio was not far from the other 1200 nm gratings on this list, but it would have been difficult to justify doubling the project budget in order to achieve a marginal increase in grating power efficiency. This spec could also be compensated by cranking up the optical power probing the soil, presumably for less than the $45 difference between the Edmund Optics and Thorlabs models.

Focusing Optic The spectrometer needed a lens to focus each separated beam of unique wavelength onto a spot the size of the sensor. NBK-7 glass was a cheap, high-quality material popular for these applications, making it the obvious choice for the focusing optic. Ray tracing analysis showed that the optimal lens had a focal length of approximately 50mm and a diameter or height of 30mm or about 1 inch. The effective focal length was a soft constraint, since the position of the lens contributed to total focus. However, the diameter of the lens was a hard constraint. This was because if the lens was too wide, it would block the input beam before it hit the diffraction plate. The lens could either be biconvex spherical or cylindrical, with the advantage of plano-cylindrical lenses being that they required fewer parts to mount. In this design, while the width of the lens was a limiting factor, the center thickness and weight were unbounded.

Table 19: Focusing Optic

Model	Manufacturer	Shape	Focal Length (mm)	Diameter /Height (mm)	Cost ($)
Stock #45-163	Edmund Optics	Spherical	50	30	39.00
SLB-30B-50P	Opto Sigma	Spherical	51.1	30	42.84
LB1471	Thorlabs	Spherical	50	25.4	26.95
Stock #35-024	Edmund Optics	Cylindrical	50	25.4	67.00
LJ1695L1	Thorlabs	Cylindrical	50	32	126.96

The priority was optical focus, however, the wide selection of lens designs available ensured that this target could be reached exactly. Since lens diameter was the next limiting factor, the LB1471 and the Edmund Optics Stock #35-024 were preferred. While the other models might have technically been small enough to avoid obstructing the input beam headed towards the diffraction grating, by going several millimeters under the maximum value, the lens position could be adjusted closer to the grating, allowing for greater flexibility of focal length. The cost of the cylindrical #35-024 was two and a half times that of the spherical LB1471. While it might have been simpler to mount, it could not have been simple enough to justify the high cost. Our choice of lens was the LB1471.

Fiber Optic Patch Cable To transmit the Electromagnetic waves from the soil to the spectrometer, we needed a waveguide. Fiber optics were chosen as a cheap, flexible waveguide coated in a durable jacket material, and fiber patch cables came with precision optical surfaces at either end, as well as attachments for connecting to lens mounts and other optical fibers. The beam coupled into the fiber was diffuse because it was being scattered off the surface of the soil. The fiber needed to have the largest possible core diameter and numerical aperture, so it was a multimode fiber. The standard connector in spectroscopy devices was the SMA (subminiature assembly). The fiber was contained within a space of less than a cubic meter, so 2 meters were sufficient for all flexibility and attenuation per kilometer was negligible with one exception. Fiber glass with trace amounts of hydroxyl ions tended to absorb optical power at around 700nm and 950nm, which was problematic because this interfered with determining whether this material was present in the soil matrix. Low OH fibers were preferred because they minimized this absorption. Thorlabs sold a variety of fiber patch cables with Low OH and a spectral range from 400nm to 2000nm and higher.

There were a wide range of fiber patch cable producers, however the above constraints ruled out the majority of suppliers, either because they did not have docu-

mentation indicating whether the material had low Hydroxyl ion concentrations or because the fiber patch cables they offered came with unnecessary features like armor cabling or antireflective coating. Thorlabs offered the only real options for comparison.

Table 20: Fiber Optic Patch Cable

Model	Manufacturer	Numerical Aperture	Core Diameter (um)	Cost ($)
M45L02	Thorlabs	0.5	365	93.63
M44L02	Thorlabs	0.5	200	88.46
M28L02	Thorlabs	0.39	400	110.95
M38L02	Thorlabs	0.39	200	88.46
M14L02	Thorlabs	0.22	50	82.99

After all other design decisions were made, the three remaining criteria were Numerical Aperture, Core Diameter, and Cost. M14L02 was the cheapest model, but not by much, and the low Numerical aperture would have reduced the total light that could have been focused into the fiber. This could not have been fixed with additional lenses. Models M38L02 and M44L02 were the next cheapest. They boasted a much greater core diameter, and one of them had the maximum typical numerical aperture for glass fiber. The M45L02 and M28L02 had the largest core diameters, but in the M28L02 model, this came with a reduced numerical aperture, which decreased input optical power, making it inferior to the M45L02. The M45L02 and M44L02 stood out as having the maximum possible Numerical aperture. Between the two, the question was whether the extra core size was worth the 5$for added performance. This was difficult to determine theoretically, but since the greatest difficulty of the system was achieving sufficient signal to noise ratio, we erred on the side of caution and chose model M45L02, which best transmitted a clear signal.

Fiber Collimator As stated above, the beam that needed to be coupled into the fiber was not collimated by the light source, the soil, or the fiber. In order for it to be coupled into the beam, a fiber collimator was needed. Fiber optic cables came with attachments that housed lens arrays focused at the surface of the fiber core. This allowed light coming into the lens to be coupled into the fiber, and light coming out of the fiber to be collimated into a beam. Fiber collimators could be mechanical, optical, or optomechanical, depending on the application. Fixed collimating packages were the cheapest solution with compatible mounting for SMA fiber cables. Thorlabs offered three relevant models.

Table 21: Fiber Collimator

Model	Manufacturer	Numerical Aperture	Output Beam Diameter (mm)	Cost ($)
F240SMA-980	Thorlabs	0.5	1.7	177.45
F110SMA-980	Thorlabs	0.37	1.35	177.45
F220SMA-980	Thorlabs	0.25	2.4	177.45

Since cost was equivalent for all models, the only factors to consider were numerical aperture and beam size. All three beam sizes were acceptable and would not have had a significant impact on the system quality. In order to maximize the amount of light that could be coupled into the fiber, the numerical aperture was selected to match that of the fiber. Model F240SMA-980 was selected by our group for these reasons.

4 Design Constraints

In this section we will cover the standards we will be adhering to throughout the development of our project as well as some of the realistic constraints that are imposed on us.

4.1 Related Standards

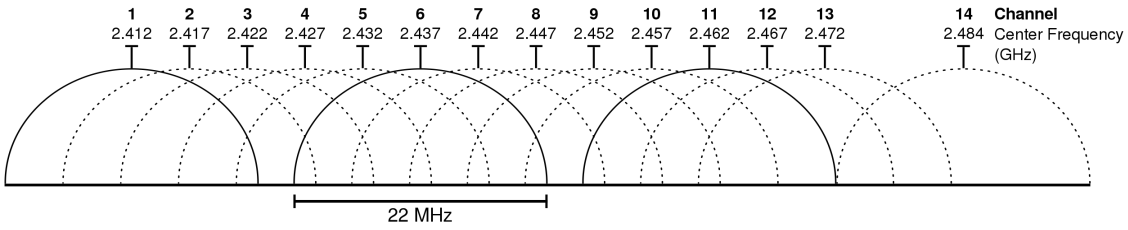
4.1.1 C++14

C++ was programmed according to the C++14 standard provided by Texas Instruments' ARM compiler. This standard is formally known as ISO/IEC 14882:2014. C++ is a superset of C, and builds upon it by introducing object-oriented programming concepts while maintaining the functional language aspect of C.

4.1.2 802.11

The microcontroller (MCU) supported transmission through the Institute of Electrical and Electronics Engineers (IEEE) 802.11b/g/n standard of wireless communication. This standard used the S band of radio frequencies and operated at 2.4 GHz. There were 14 accessible channels, each spanning a bandwidth of 22 MHz (pictured in Figure 23).

Figure 23: 802.11b/g/n channels [1]

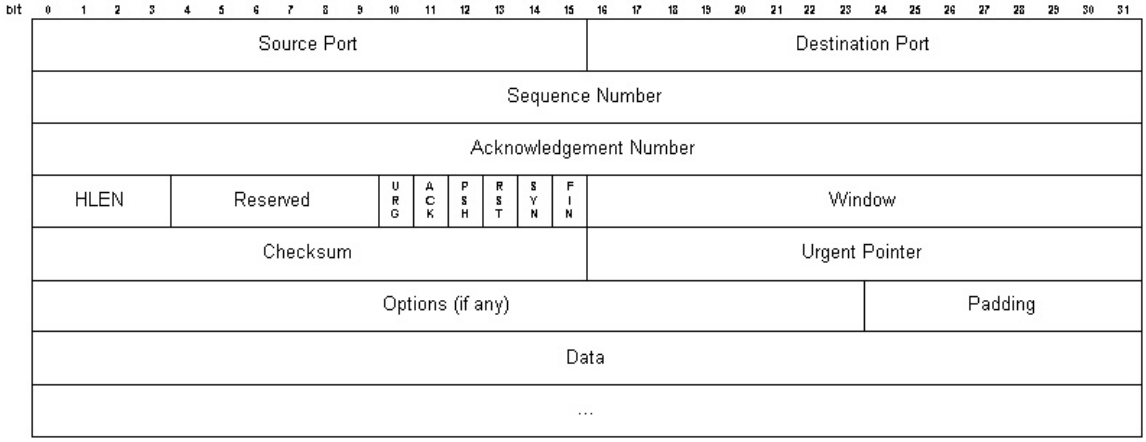


These channels specifically resided in an industrial, scientific, and medical (ISM) band. This standard also provided datagram frames for the transport layer.

4.1.3 TCP

Transmission Control Protocol (TCP) was used to satisfy the transport layer requirements of the product, and was used to transmit symbols (i.e., from any commands, data, settings, telemetry, etc.) between Amazon Web Services (AWS) and the microcontroller (MCU). TCP was chosen over other protocols, such as User Datagram Protocol (UDP), mainly due to its reliability. The extent of TCP's reliability included features such as checksums, duplicate data detection, retrying of transmissions, sequencing, and timers. Such reliability was favored over higher bandwidth or lower latency, as neither of the latter were required for the kilobytes of information being relayed between AWS and the MCU. A standard TCP frame was shown in Figure 24.

Figure 24: TCP frame [2]

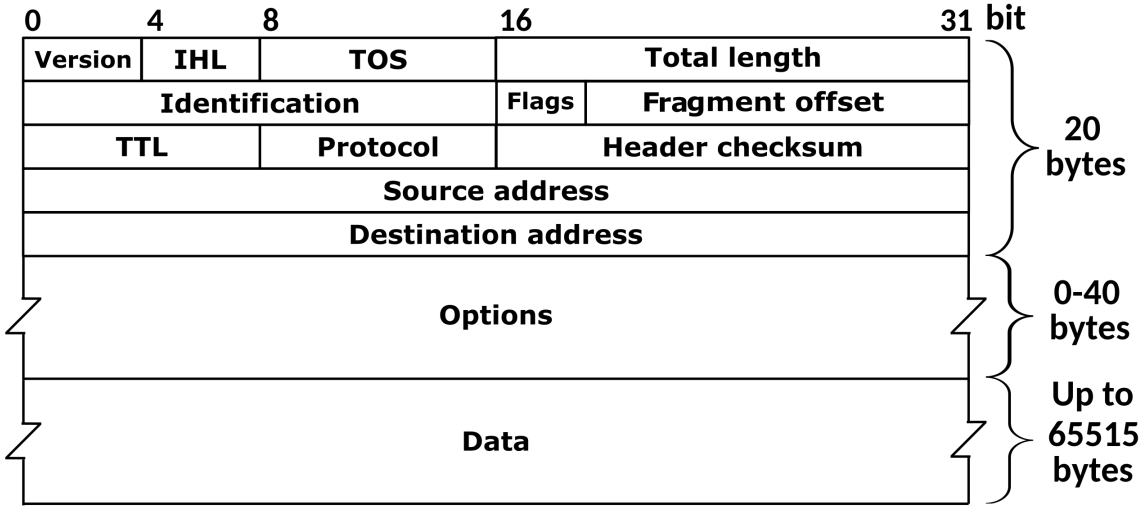


4.1.4 IPv4

IPv4 was the fourth version of the Internet Protocol, a network layer protocol in use to relay data between devices and across networks. The data relayed, datagrams, were sent between sources and hosts that were identified by 32-bit addresses. This

protocol strictly functioned to transport the datagram from one device to another, with no end-to-end reliability, flow control, sequencing, or other measures found in other protocols such as TCP. IPv4 provided two distinct features: fragmentation of whole datagrams, and addressing of devices. A standard IPv4 frame was shown in Figure 25.

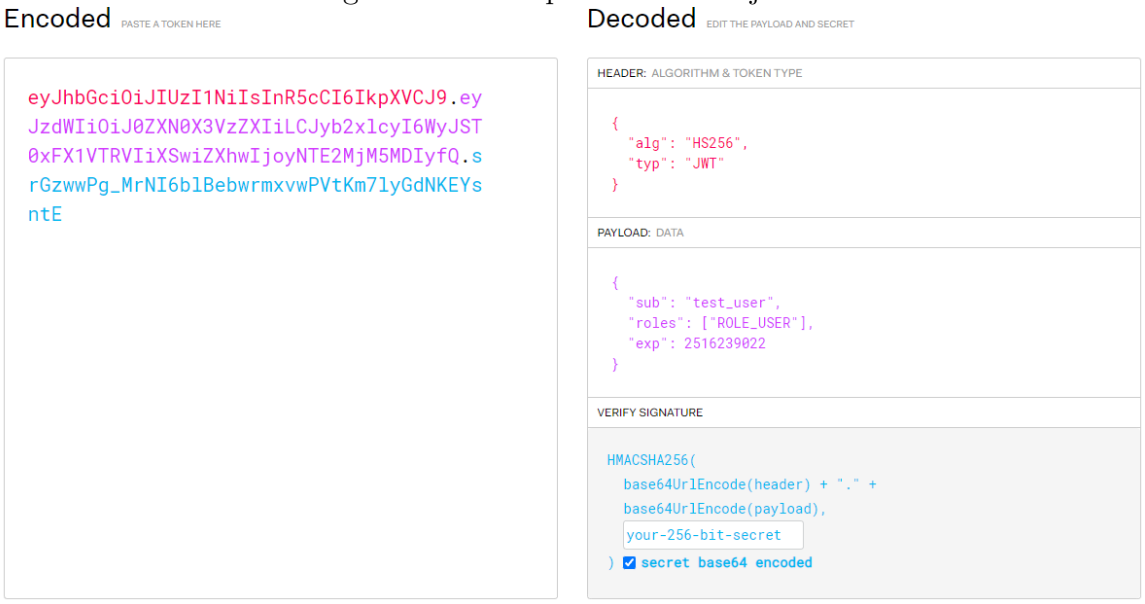
Figure 25: IPv4 frame [3]



4.1.5 JSON Web Token (RFC 7519)

This standard specified a "compact, URL-safe means of representing claims to be transferred between two parties." The means was via the JSON Web Token (JWT). JWTs were split into 3 parts, the header which specified the algorithm the key(s) used to encrypt the message and the type of token, JSON Web Encryption (JWE) or JSON Web Signature (JWS). The second part was the payload. The payload was a JSON formatted object which carried claims. And the third part was the verification signature which was the Base64 encoded header plus a '.' plus the Base64 encoded payload, another '.' and finally this was all encrypted by the key. When a JWT was received, the header and payload could be read by just Base64 decoding these portions of the token. The JWT was verified by decrypting the verification signature, if the signature could not be decrypted with the key then the token was invalid. Figure 26 gives an example of a JWT that was signed with a secret key.

Figure 26: Example JWT from jwt.io



Claims Claims were in the payload of a JWT. The basic claims specified by this standard were `iss` "issuer", `sub` "subject", `aud` "audience", `exp` "expiration", `nbf` "not before", `iat` "issued at", and `jti` "JSON Token ID". For our purposes, all of the JWTs were self-signed so the issuer, JSON Token ID, audience, and issuer claims could all be disregarded as they were meant for cross-service authentication. However, following the standard, the "subject" claim was the user principal, and the expiration and issued at claims were used to invalidate a token cookie after a certain period of time. The standard allowed for public and private claims outside of these as well. Public claims were registered with the IANA while private claims were collision prone, neither of these were of true concern to us.

4.1.6 HTTP/1.1 (RFC 2616)

This standard defined an application-level protocol communicating across the internet. RFC 2616 was an update to previous protocols that increased the capabilities of the original in the form of persistent connections and the ability to send files in the form of MIME types. MIME types held metadata and the binary data of a file. This standard also served to define how dates/times were handled on the web as well.

4.1.7 WebSocket Protocol (RFC 6455)

The WebSocket Protocol standard (aka. "WebSockets") was an application layer protocol that allowed full-duplex communication over a single TCP connection. WebSockets could be considered a comparable to HTTP, but were compatible with HTTP,

though it functioned effectively similarly to TCP serial bytestreams, with the advantage in using WebSockets being in that devices connected via a URL. This allowed devices to rely on URLs for addressing instead of statically-assigned IP addresses. The WebSocket Protocol also further specificied how the connection was made, how the data was formatted, and some security considerations. Most of these items were abstracted away from users in the forms of libraries—however, the security concerns were likely the most important part of this standard in terms of this project. This reason being that we were receiving location and IP data from the packets in order to build out commands like where to point the solar panels. Exposing PII such as location

4.2 Constraints

The following section covers specific limitations, or constraints, the team faced during the project timeline. These restraints had affects on our ability to design specific systems and limited our options when designing different parts of the product. The following were our constraints: economic, time, equipment, safety, environmental, manufacturability, ethical, and sustainability. Each section discussed possible and realized problems with regard to the specific category of constraint. After explaining and detailing each constraint, we explored different solutions that we could use to overcome any barriers. Some problems did not occur for the team, but we wanted to be prepared for any constraints that may arise.

4.2.1 Economic

The first major constraint that we expected and faced was economic. We considered and looked for different sponsorship options for this project, but unfortunately our team was unable to secure a sponsorship. Our funding for the project was entirely self-funded and evenly divided between the four team members. As a result of being funded entirely out-of-pocket, cost was a driving factor in the materials and components our team decided to use for the product. When analyzing various parts and materials for the product, ideally, we would choose the best available, but when budgeting for both us and the consumers we planned to serve, we planned to maximize specifications and requirements while minimizing the cost. This meant our product may not have had all the "nice-to-have" features and instead had features to make it perform just enough to be competitive in the market we designed the product for.

Another hurdle that the economy brought to this project was the availability of certain parts. Manufacturers across the world had been drastically impacted by the economy over the last few years making various products unavailable, hard to get, or more expensive than ever before. We expected to spend more time comparing components because of lack of availability compared to the pricing of such parts. Lack of availability of certain components also affected the final outcome of our product

because we were not able to get the exact parts we wanted as part of our design to meet time requirements for the project.

We ran into the problem of availability with products on multiple occasions. An example of this was when we were looking for charge controllers from the manufacturer Analog Devices. We were originally looking at the charge controller "Power Tracking 2A Battery Charge for Solar Power (LT3652), but quickly found that it was unavailable. This specific option for a charge controller was an ideal option because of its price point and specifications. Our product only needed 5V to operate so when looking for a component we were looking to meet just about our needs. Our second choice was through TI which had a minimum of 5V and max of 28V, but again ran into the issue that the component was not available. We then considered two other options that were more expensive, had higher voltage, and offered other features such as telemetry, and a low-loss power path that we did not need for our product.

Not only were we forced to select something more than needed, but this caused future restraints on components remaining in budget and delayed time because of the extra time spent searching for products available that met our requirements.

The side effects of our budget and product availability in the economy were also affected by the inflation rates. Not only could we run into the problem of not having a product readily available to meet our needs, but inflation could also cause a component to fall outside of our budget.

In order to combat the restraint of time, we looked at future products we might need and considered different options ahead of time. This allowed us to plan ahead and order products for future parts of the project so we wouldn't have to wait on them later.

4.2.2 Time

Most projects share several different constraints. One of the constraints that is found with every product is the restraint of time. Customers always want the next product, and teams need various resources made on a deadline. For this project, we were on a timeline to complete the project and various iterations within the semester. Most projects revolve around the deadline that meets a customer's needs. In this case, the project was part of a larger organization that ran through business cycles every year. The deadline in place was imposed by the restructuring of the university as a workforce.

Not only was the project restricted on a semester timeline, but the project also relied on the schedule set forth by the university. We recently experienced a time con-

straint when a hurricane came through Orlando. Not only did the university close, but some of our team members lost power and were unable to work during this time. Additionally, the university closed over the holidays, limiting resources and the ability to make some decisions to progress the project.

As all parts of this project correlated with each other, so did our restraints. Earlier, we discussed that some products had limited or no availability. This further pressurized the time restraint we had because of product delays. We were already delayed by certain products being available and had plans to look further into our project design to try and avoid further delays with shipping and manufacturing.

Another factor that affected the timing of our project was our own availability. Each of our team members had various projects and responsibilities, and we worked together to find time that worked for everyone on the team. Sometimes our schedules did not align and caused us to push our meetings back a day or two. This was something we worked around by communicating regularly through Discord, even if we were not able to meet face to face.

Since we knew we had another semester working on this project, we had more flexibility this semester to move things where needed in our project timeline. The closer we got to the deadline, the less flexibility we had with our time. Because we recognized this restraint early on, we discussed this topic and allowed time to make changes in the future by making as much progress as possible now.

4.2.3 Equipment

There were three main equipment constraints on our ability to produce our project. One was limited access to software, another was heavily shared tools, and the third was having facilities to build.

This project arose out of a tradition of problem-solving that was used in educational facilities and industries all over the country. A chief part of the College of Engineering and Computer Science program was to teach students to use the tools industry used to solve the problems industry faced. So far, our team had used multiple integrated development environments, version control platforms, word processors and file viewers, text, video and audio communication services, and software for designing optical beam paths, CAD models of physical structures, and PCB simulations to express and determine design features for our project. Each of us had tools that we would have liked to have used but could not, either because of licensing, inexperience, or the need to work cooperatively with the larger team.

Another equipment constraint had been in the form of shared workspaces. In

order to leverage these tools, we needed relatively quiet space to sit where we could communicate with each other and run power-hungry devices while connected to the internet and organizational networks. Our University had provided us with design labs, but their tools were managed by students, and inventory was often untraceable and disorganized.

The third equipment constraint was unique to the nature of this project, but it came from needing to build outdoors. Building a garden bed required land, if only a little, and although parts could be fabricated in clean spaces the system was designed to contain wet, heterogeneous dirt. Space outdoors was necessary to build and that meant either permission from the University or leveraging team member access to land.

Each of these equipment constraints had a different effect on the project, some of them were easy to get around by finding open-source alternatives to licensed software, by cooperating with other students to get more out of leftover optical components, and by using student networks to determine where and when our group would have space to meet. Some were more difficult, causing us to trade time and money that we would have rather not given. In the end, equipment constraints did not likely impact the project deadline or affect the features as laid out in the document.

4.2.4 Safety

This project featured several threats to human safety that needed to be addressed. The power subsystem was rated with sufficiently high voltage and current to cause cardiac arrest. The user had to be protected from direct exposure to electrical conductors within the system, especially as this was an outdoor system with water management mechanisms which could add to the risk. The optical subsystem involved the use of infrared probes and focusing lenses. If handled carelessly, these could potentially create an eye hazard during testing or product use that the victim would not be able to detect. Steps had to be taken to indicate the nature of the threat where it existed. Other risks involved in the project included the chance of a minor cut on a sharp surface becoming infected from exposure to the soil. The structure and components of the project had to be sufficiently dull to ensure against the possibility of this.

Recognizing the importance of our team's safety allowed us to think through different precautions we could take when building and testing the product. One precaution our team planned to utilize was personal protection equipment and understanding the tools we would utilize. We also ensured that we used proper tools for different jobs. We discussed as a team with each other to know what tools we had and potential items needed to purchase to ensure we had the right equipment for the job. Taking inventory of this also allowed us to budget any tools in as needed. We also planned

accordingly on when to bring professionals in for various needs. For example, no one on our team was trained to cut and bend the garden materials on the exterior so we planned to utilize campus resources and personal connections to ensure we got the job done correctly and safely.

4.2.5 Environmental

Our product featured consumers utilizing a garden bed which directly engaged with the environment. Because our product was doing this, there were environmental factors that our team had to consider. Although we were not directly pouring soil into each customer's garden bed, we knew that different fertilizers would be purchased and used to fill our products. This had the potential to disturb ecosystems, even if it was through the butterfly effect. Additionally, we needed to consider the risk of pollution our batteries and materials caused on the environment. When we discussed the marketing requirements, we also considered the environmental impact because we knew our consumers would care about this factor. We also had to consider the fertilizer runoff, sound pollution, and light pollution that could occur as a result of our project.

To consider the impact we could have on the economy, we also took careful consideration in the components and materials we used for our product. The component that had one of the larger impacts on this restraint was our battery. Many of the other components and parts could be re-used, but batteries did not have the same type of use after the product was retired. This was also a reason we decided to make our product solar. We wanted it to last as long as possible and use less energy where it could. Not only did we consider performance standards of various components, but also the lasting impact these parts would have on our environment for years following. The longer we could make our product last and the better, the less it would negatively impact the environment.

4.2.6 Manufacturability

Manufacturability was a set of important criteria early in the design process to avoid making costly mistakes. When we originally selected our project, we wanted to make sure that whatever project we chose would be achievable and able to be built. For this project, we had to ensure that building the garden bed would be feasible for our team and the resources we had available. We considered the two reasons that could cause a product to potentially not be manufacturable. The first one was that the design was too complex to be completed, which could be caused by any number of reasons within the design. We considered all the other constraints that had been discussed and whether those constraints would further constrain the manufacturability of the garden bed. The second reason related back to the cost restraint, and we had to budget and determine if we would be able to fund all the requirements to build out

the entire product and account for variability.

Our team protected against cost-prohibitive manufacturability issues by ensuring that our component selection was traceable and that each component was in production and within our price range at cost, not due to clearance discounts. Another step our team took against future constraints was selecting the bulk of our components and design features before taking steps toward production.

4.2.7 Ethical

When considering the ethics behind the project, the team found it easier to meet the ethical constraint. They knew that the positive impact of the project would outweigh any negative impact it might have on the market. The project would not only encourage consumers to grow their own vegetables and plants, but it would also bring mindfulness to many lives. The team was aware of numerous studies that showed the benefits of being outside and tending to a garden, and this project would enable that behavior. The garden bed was designed to minimize its environmental impact, and the team considered user data and security of the product. They limited security risks for the consumer and surrounding neighbors due to the wireless communication system they were designing and the information it needed to work. There was minimal room for any harm to come to the consumers. The product was also open to any consumer to utilize and did not risk damaging any cultural heritage or societies. The garden bed could be used by anyone who decided to purchase it. The team committed to submitting to University policy as regards the goals and activities of this project.

4.2.8 Sustainability

Sustainability meant creating a system that would produce as much of the resources as it used so that the system did not collapse. An example of this was when we were deciding what materials to make the structure out of. We wanted to maximize the life of the product without hindering our budget. We found that aluminum met several consumer requirements, as well as our other project restraints. Aluminum was also more sustainable than some other materials, such as wood, that were involved in the discussion. Another point for sustainability that we had considered was both battery life and the life of the solar panels. We wanted to find a balance between the life and cost of these components. Additionally, we hoped to have them integrated in a way that when they needed to be replaced, the consumer was not forced to replace the entire product, but only the ones that were beyond their useful life.

Beyond product life sustainability, we had also considered other factors. The power generation subsystem was intended to ensure that our system did not need to draw on the power grid in order to function correctly. This would lower the impact

on both the environment and how well the product could sustain itself. We also considered the water detection to ensure accuracy so that water was not wasted. The sensors and system would minimize water waste by only watering when needed and at appropriate times before or after sunshine.

In a broader sense, our project would increase the wellbeing of people's lives, producing new plants and possibly food. One obstacle to sustainability was power storage. Our system used a solar panel to collect energy, but that energy needed to be stored in order to distribute it in the right amounts and at the right times. We would be selecting our battery carefully and minimizing the waste of this as much as possible.

5 System Hardware and Software Design

5.1 Controller Subsystem

In order for our system to be as self and power efficient as possible from an end-user perspective, it was determined that our product would require an internet connection to offload remote command-and-control to an Amazon Web Services EC2 instance (hence referred to as "AWS" and detailed in subsection 5.4). To make the process of operating our product as hands-off as possible to end-users, the microcontroller connected to the user's home WiFi network for access to AWS. Bluetooth, Zigbee, Thread, and other short-range 2.4 GHz communication protocols were disfavored over WiFi, as it was predicted that most users would not have a device dedicated to connecting our product via such protocols. Long-range (LoRa) protocols were deemed unnecessary, as the intended placement of our product was outside, near or next to the user's home. We did not expect our product to produce or receive large amounts of data, so the decreased bandwidth of a WiFi-enabled product being beyond the outdoor walls of a building was not a significant drawback to our application. A wired connection (802.3/Ethernet) was deemed too invasive to the end-user. It was expected that most, if not all, end-users had a wireless access point and internet access. Thus, connection via the 802.11/WiFi standard was a natural choice for our use case.

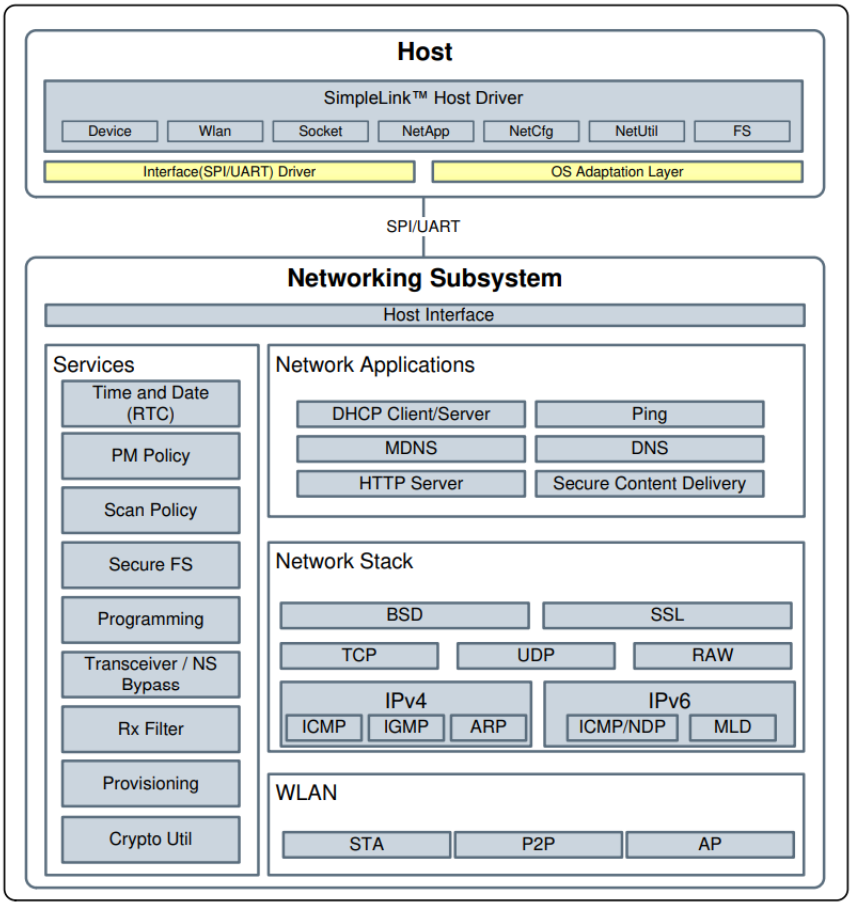
CC3200 Overview The Texas Instruments CC3200-series (hence referred to as the "CC3200", the "MCU", or the "microcontroller") of microcontrollers were WiFi-enabled chips with an ARM Cortex-M4 central processor and a WiFi network processor, along with many useful peripherals and power management modules. This series of processors was delivered alongside a software development kit (SDK) provided by Texas Instruments to ease the development of Internet-of-Things (IoT) applications.

The CC3200's WiFi network processor supported the following standards/features useful to our development (see Figure 27):

- WiFi standards: 802.11b/g/n
- WiFi security: WEP, WPA/WPA2 PSK, WPA2 enterprise, WPA3 personal, WPA3 enterprise
- WiFi provisioning: SmartConfig, WPS2
- IP protocols: IPv4, IPv6
- IP addressing: static IP, DHCPv4, DHCPv6
- Transport: UDP, TCP, RAW
- Host interface: UART, SPI

The CC3200's networking subsystem was identical to the CC3220's, and is further detailed in Figure 27.

Figure 27: CC3220 networking subsystem [4]



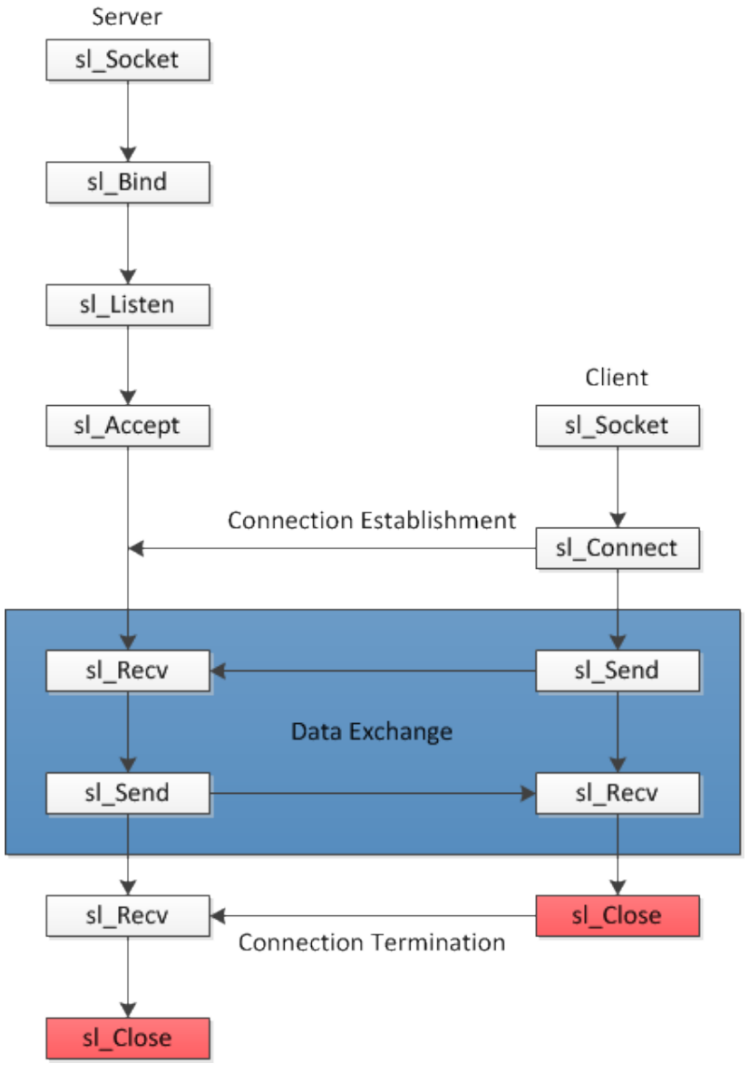
Connection During the development of our product, the microcontroller connected to a WLAN via a hardcoded password (and username, if needed), or via SmartConfig. The system received an IPv4 address via DHCP, or it was able to be statically assigned.

Web Interface Option The first option allowed unanimous adaptation of our product for home users. This option consisted of performing first-time setup through a WiFi Direct connection to the MCU and allowing the user to log in through a web portal to finish setup of their device. The portal allowed a user to connect to their WLAN of choice, and also allowed the user to log in to the MCU locally to view telemetry and settings. The user was also able to configure the MCU for static IP addressing or DHCP IP addressing.

WPS Option The second option was to implement a momentary switch which was programmed to connect to the MCU's WiFi Protected Setup (WPS) feature. This would still allow the user to connect to their network of choice—however, the user's wireless access point (WAP) would need to support WPS. Furthermore, DHCP IP addressing would be forced, and there would be no web portal for the user to access for viewing telemetry and settings. This option would have greatly eased development, though, and would have let the MCU and web teams focus on developing and refining their respective subsystems.

Communication Method The MCU communicated with AWS through TCP sockets. After connecting to the user's home network, the MCU checked if there was an internet connection. Once an internet connection had been established, the MCU opened a socket and connected to the AWS instance via its URL (using the default DNS nameserver provided to network clients). The software control flow of TCP sockets as our product implemented them was detailed in Figure 28.

Figure 28: TCP socket control flow [5]



Connection Parameters Once the MCU had established a connection to the internet and the AWS instance, it attempted to send current system settings and telemetry, as well as sensor readings to AWS. The microcontroller did this at least once every 15 minutes. Sending and receiving may have occurred more often if commanded to by AWS. Because TCP was being used to connect AWS and the microcontroller, any manual retries on a failed send or receive were most likely futile. Therefore, any sort of link error handling was performed by the link and not the microcontroller program.

Over-the-Air Updates At that time, our team did not intend to provide a method for over-the-air updates (OTA); however, this was a provision that could have been developed in the future.

Startup Upon startup, the system shall initialize these peripherals in the following order within 30 seconds of startup: watchdog timer, GPIO, ADC, network stack/WiFi module. The system shall then check for interrupts, and execute ISRs if necessary. The system shall then verify its connection to the internet, if set, by attempting to ping `8.8.8.8` and `8.8.4.4` 16 at most times over the period of 8 seconds. If 4/16 responses have been received at any point, the system shall attempt to verify network DNS status by pinging `google.com` 16 at most times over the period of 8 seconds. If 4/16 responses have been received, the system shall continue with startup and clear the no connection flag (if set). If no connection has been established (i.e. the system fails to ping the aforementioned addresses at either step), the system shall set a flag to indicate no connection has been received, and reset. If no connection has been established, and the flag is set, the system shall go into critical power mode and flash the yellow LED for 1 second, 1 time every 2 seconds. The user shall be able to correct the issue via user interface (e.g. SmartConfig or other developed interface). The system shall measure the battery voltage, and set itself to the appropriate state described earlier.

Shutdown The system shall generate an interrupt to shutdown the system. The shutdown ISR shall run and perform the following actions. The system shall finish any RF transmissions within 5 seconds. The system shall terminate all remaining RF transmissions after the 5 second window has elapsed. The system shall save any network parameters to nonvolatile memory to be used upon startup within 1 second. The system shall safely shutdown any initialized peripherals within 5 seconds. The system shall safely shutdown any sensing components within 5 seconds. The system shall "hand-off" any control it had over the power subsystem to the battery's charge controller within 5 seconds. The system shall go into the shutdown power mode, stopping the processor within 1 second.

Reset In the event of a reset state, the system shall perform the shutdown ISR, with each section of the ISR performing its necessary tasks within the allotted time mentioned above. The system shall perform its startup ISR, with each section of the ISR performing its necessary tasks within the allotted time mentioned above. There shall be a time period of no longer than 5 seconds between the end of the shutdown ISR and the beginning of the startup ISR.

Networking The system shall be able to connect to the user's A 2.4 GHz-based wireless network. It may either have no security, or be secured with WPA2 (Personal or Enterprise). The system shall be able to connect if the signal strength of the network is nominal (around -70 dBm). The network shall adhere to the standards of 802.11b, g, or n. The system shall received an IPv4 address assigned by network DHCP, or the user shall be able to statically assign a local IPv4 address. The network shall use NAT, and not expose the product to the WAN. The system shall use DNS provided by the WLAN gateway. The system shall be able to connect to the internet

within the timeframe described by the startup sequence. The system shall be ready to establish a connection to AWS within 5 seconds of verifying connection to the internet.

AWS The system shall first establish a connection to AWS by sending a burst of 5 data packets of the size and containing the variables described in Figure 31 over a period of 25 seconds. If no connection is established over a period of 15 minutes, the system shall send the same burst of data packets and repeat every 15 minutes until a connection is established.

Receive (Sockets) The system shall listen for commands from AWS via TCP sockets. If no traffic is received from AWS within a period of 40 minutes, the system shall terminate and attempt to reopen the socket. The system must be able to open and configure a socket within 30 seconds.

Send (Sockets) The system shall send data to AWS via TCP sockets. If no traffic is successfully sent to AWS within a period of 40 minutes, the system shall terminate and attempt to reopen the socket. The system must be able to open and configure a socket within 30 seconds.

Receive (Command) While a connection is established with AWS, the system shall continually listen for commands from AWS. The system shall be able to fully receive and decode commands from AWS within 10 seconds of the start of the reception. The system shall be allotted 1 second per command to schedule the received commands.

Send (Data) While a connection is currently established with AWS, the system shall send a burst of 2 data packets over a period of 10 seconds every 15 minutes. The data packets shall be of the size and contain the variables described in Figure 31.

Solar and Battery The system shall monitor and update the solar array current and voltage output values at least every 10 seconds. The system shall run in active mode between sunrise and sunset, while the battery has 40-100% calculated charge remaining. The system shall run in low power mode between sunrise and sunset while the battery has 10-40% calculated charge remaining, and between sunset and sunrise while the battery has 10-100% calculated charge remaining. The system shall run in critical power mode while the battery has 5-10% calculated charge remaining. The system shall be in shutdown mode while the battery has 0-5% calculated charge remaining. The system shall be able to switch power modes within 5 seconds of the interrupt being triggered.

Table 22: MCU power mode behavior

Time of Day	**Calculated Battery Charge (%)**	**Power Mode**
Day time	100-40	Active
Night time	100-40	Low
Any time	40-10	Low
Any time	10-5	Critical
Any time	5-0	Shutdown

Sensing The system shall monitor and perform analog-to-digital conversion on the spectroscopic sensor values at least every 10 seconds while not in shutdown or critical power mode.

Commands from AWS The MCU will receive commands and data from AWS in the following form:

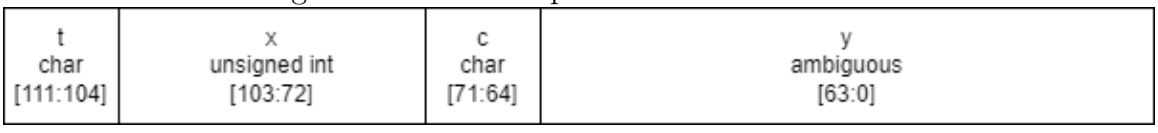
`t x c y`

where `t` (character) marks the beginning of the command string, `x` (unsigned integer) indicates how many seconds to wait before executing command `c` (character) with parameter `y` (ambiguous). The following characters occupy the spot of `c`, and translate to the following commands:

`w y`: water, `y` is boolean
`a y`: solar θ, `y` is double
`b y`: solar ϕ, `y` is double
`l y`: low power, `y` is unsigned int for flags
`s y`: send data, `y` is unsigned int for flags

For example, if AWS instructs the MCU to turn on the water source in 3 minutes, it would send the command `t 180 w 1`. If AWS would like the MCU to change the θ of the solar panels to 45° immediately, it would send `t 0 a 45.0`. It is important to note that the values of `x` and `y` are not strings (e.g. 45° will not be sent as `"45.0"`), but the actual encoding of 45° as a double-precision float. This is to maintain a consistent command string size amongst all transmissions.

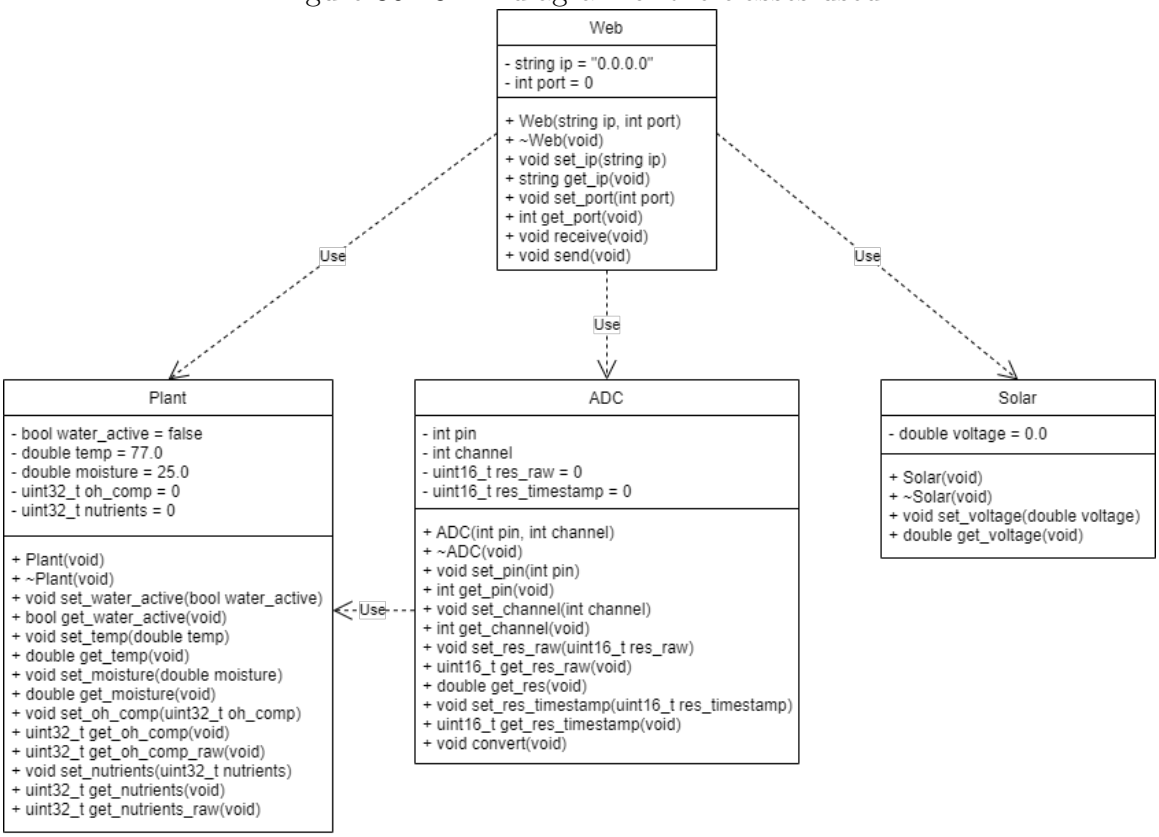
Figure 29: Bitwise representation of command



C++ Structure It should be expected that all headers from the C++ standard library (as defined in C++20) will be used, along with the following nonstandard libraries: Texas Instruments SimpleLink™ CC32xx SDK. Additionally, it is expected that the MCU program will schedule tasks using a Singleton Pattern thread to ensure thread safety when accessing variables.

Classes There will be a few classes defined in the MCU's programming, detailed in Figure 30. Classes will be laid out in the header file according to the definitions given in the UML diagram.

Figure 30: UML diagram of the classes used



Web Class The Web class will have 2 private variables: `ip` (string) and `port` (int). `ip` signifies the IPv4 address of the AWS instance to connect to, and `port` is the port to access AWS through. The default state of this class is `ip = "0.0.0.0"` and `port = 0`. It will have a public deconstructor and a constructor, requiring the variables string `ip` and int `port` to be passed to the constructor. The class will have 4 public setters and getters: `void set_ip(string ip)`, `string get_ip(void)`, `void set_port(int port)`, `int get_port(void)`. The class will have 2 public functions: `void receive(void)` and `void send(void)`. `receive()` will be used to receive the next command from AWS, and will then decode the command and use functions from

other classes to execute the command received from AWS. `send()` will use functions from other classes to combine and send the current sensor readings and settings to AWS. Only 1 object of `class Web` will be instantiated.

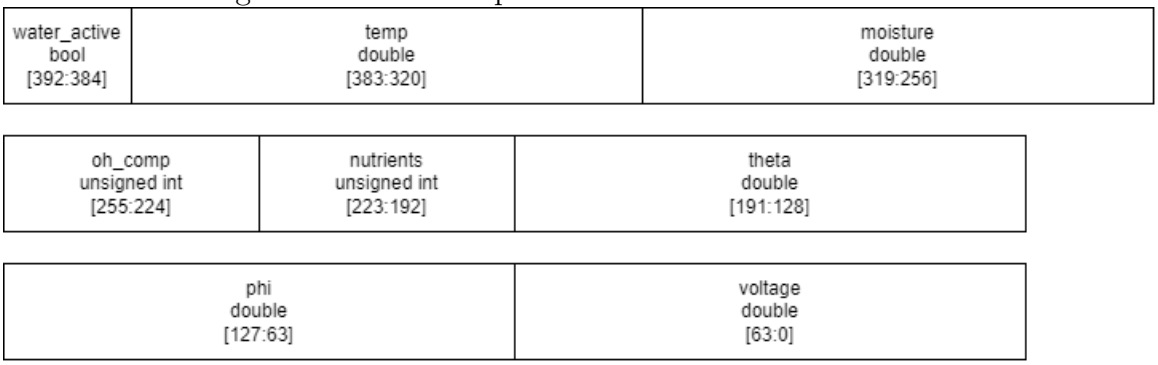
Plant Class The Plant class will have 5 private variables: `water_active` (bool), `temp` (double), `moisture` (double), `oh_comp` (unsigned 32-bit int), and `nutrients` (unsigned 32-bit int). `water_active` is the state of the water source feeding the plant, `temp` is the current air temperature near/around the plant, `moisture` is the moisture percentage of the soil near/around the plant, `oh_comp` is an encoded word of parameters relating to the spectroscopic analysis of the soil's OH composition, and `nutrients` is an encoded word of parameters relating to the spectroscopic analysis of the soil's nutritional composition. The default state of this class is `water_active = false`, `temp = 77.0` (room temperature in °F), `moisture = 25.0` (approximate average soil moisture), `oh_comp = 0`, and `nutrients = 0`. It will have a public deconstructor and a constructor, requiring no variables to be passed to the constructor. The class will have 12 public setters and getters: `void set_water_active(bool water_active)`, `bool get_water_active(void)`, `void set_temp(double temp)`, `double get_temp(void)`, `void set_moisture(double moisture)`, `double get_moisture(void)`, `void set_oh_comp(uint32_t oh_comp)`, `uint32_t get_oh_comp(void)`, `uint32_t get_oh_comp_raw(void)`, `void set_nutrients(uint32_t nutrients)`, `uint32_t get_nutrients(void)`, and `uint32_t get_nutrients_raw(void)`. The purpose of the normal getters for `oh_comp` and `nutrients` are to get human-readable formats for the OH composition of, and nutrients in the soil. The raw getters are to get the raw values of `oh_comp` and `nutrients`. The setters accept the raw values of `oh_comp` and `nutrients`. Only 1 object of `class Plant` will be instantiated.

ADC Class The ADC class will have 4 private variables: `pin` (int), `channel` (int), `res_raw` (unsigned 16-bit int), and `res_timestamp` (unsigned 16-bit int). `pin` is the physical pin of the voltage input to the ADC on the CC3200, `channel` is the ADC channel corresponding to the selected pin. `res_raw` is the raw result of the ADC for the corresponding channel. `res_timestamp` is the timestamp of the ADC result for the corresponding channel. The default state of this class is `res_raw = 0` and `res_timestamp = 0`. It will have a public deconstructor and a constructor, requiring the variables int `pin` and int `channel` to be passed to the constructor. The class will have 9 public setters and getters: `void set_pin(int pin)`, `int get_pin(void)`, `void set_channel(int channel)`, `int get_channel(void)`, `void set_res_raw(uint16_t res_raw)`, `uint16_t get_res_raw(void)`, `double get_res(void)`, `void set_res_timestamp(uint_16t res_timestamp)`, `uint16_t get_res_timestamp(void)`. The `get_res()` getter converts the raw result of the ADC into a voltage. The class will have 1 public function: `void convert(void)`. This function manually triggers conversion in the ADC module of the CC3200. Multiple objects of `class ADC` will be instantiated.

Solar Class The Solar class will have 1 private variable: `voltage` (double). `voltage` is the voltage of the battery updated at regular intervals. The default state of this class is `voltage = 0.0`. It will have a public deconstructor and a constructor, requiring no variables to be passed to the constructor. The class will have 2 public setters and getters: `void set_voltage(double voltage)`, and `double get_voltage(void)`. Only 1 object of `class Solar` will be instantiated.

Data to AWS Unlike commands received from AWS, the MCU will simply send data directly from its classes to AWS. This is done to minimize the overhead of sending extraneous symbols and preserve power stored in the battery, as receiving uses far less power than transmitting in the MCU subsystem. Further optimization can be performed in the future to further reduce overhead (e.g. reducing `water_active` to occupy 1 bit and using the rest of the symbol to encode other data).

Figure 31: Bitwise representation of data sent to AWS



Global Variables No global variables plan to be implemented at this time. Macros may be defined for configuration of the ADC and any other dependent modules.

Interrupts and ISRs If the charge controller indicates that the battery has fallen below a certain voltage (named "low voltage"), the MCU will raise an interrupt and execute `isr_low_power()`. This ISR performs housekeeping before putting the MCU into a low power state, limiting use of its functions and consuming less current.

If the charge controller indicates that the battery has fallen below a certain voltage (named "critical voltage"), the MCU will raise an interrupt and execute `isr_critical_power()`. This ISR performs further housekeeping before putting the MCU into an extreme low power state, limiting all but the features necessary to maintain a connection with AWS and consuming a minimal amount of current.

If the MCU, AWS, or any other devices transmit indication of a dangerous state or if the MCU, AWS, or any other devices transmit indication of a shut down, the MCU will raise an interrupt and execute `isr_shut_down()`. This ISR immediately shuts down all able subsystems, and puts all other subsystems in a fail safe state. This ISR fails safe the entire system.

If the MCU receives indication of a start up (via a momentary switch), the MCU will raise an interrupt and execute `isr_start_up()`. This ISR starts up all relevant subsystems and begins the MCU's programming. All classes will be initialized to default values. This ISR starts up the entire system.

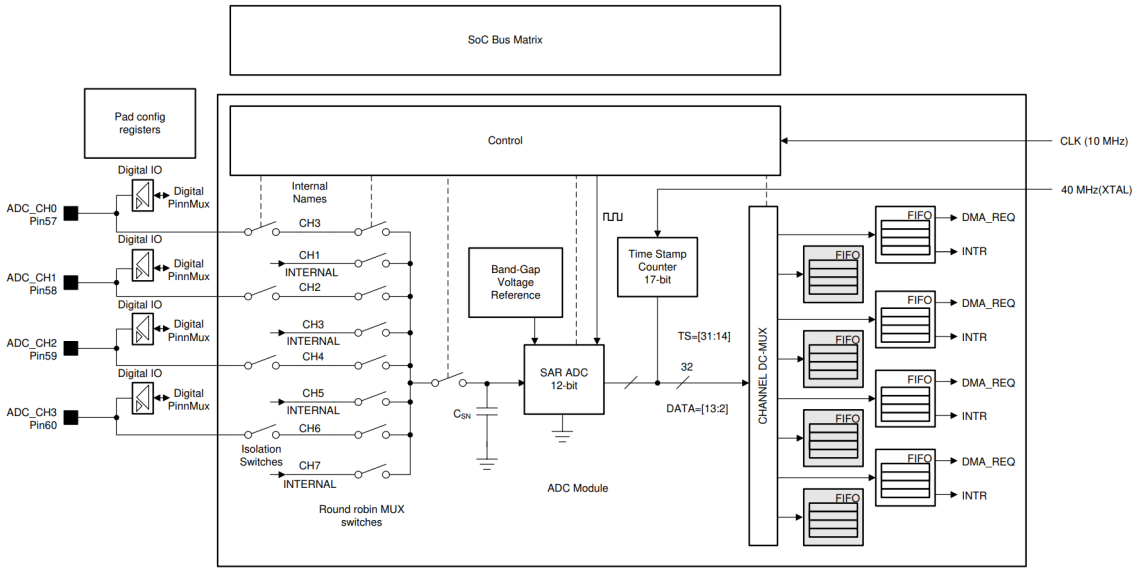
If the MCU determines it must reset the system to a default state, the MCU will raise an interrupt and execute `isr_default_state()`. This ISR returns all relevant subsystems to their default state, as if the MCU had just executed `isr_start_up()`. All classes will be initialized to default values. This ISR resets the entire system.

If the watchdog timer needs to be reset to 0 in the course of normal operation, the triggered interrupt shall execute `isr_wdt()` to reset the WDT.

File Organization Classes, macros, functions, and variables will be defined/prototyped to the extent required in a header file to be named `garden.h`. These classes, functions, and variables will be further defined in a seperate source file named `garden.cpp` as needed. The `main()` function and the remaining classes, macros, functions, and variables, required will be located in `main.cpp`.

Analog-to-digital Conversion The CC3200 contains a 12-bit general purpose analog-to-digital converter (ADC) with 4 externally-accessible channels and a sampling periodicity of 16 µs per channel (62.5 ksps per channel). The architecture of the CC3200's ADC is shown in Figure 32.

Figure 32: CC3200 ADC module architecture [6]



It is expected that the spectrometer circuit will provide a current between 20 µV and 80 mV. The 12-bit ADC provides for an input range of between 0 and 1.8 V.

Therefore, the resolution of the ADC is calculated below:

$$\frac{(1.8-0)\,\mathrm{V}}{2^{12}\,\mathrm{steps}} = 0.4395\,\mathrm{mV/step} \tag{2}$$

Thus, given our ADC resolution and the expected range of our spectrometer, the number of discrete steps of range is given below:

$$\frac{(80-0.02)\,\mathrm{mV}}{0.4395\,\mathrm{mV/step}} = \lfloor 181.98 \rfloor = 181 \tag{3}$$

These steps will be used to measure OH composition and nutrients in the soil, and functions from the class `ADC` will be used to perform analog-to-digital conversion to update values in the class `Plant`.

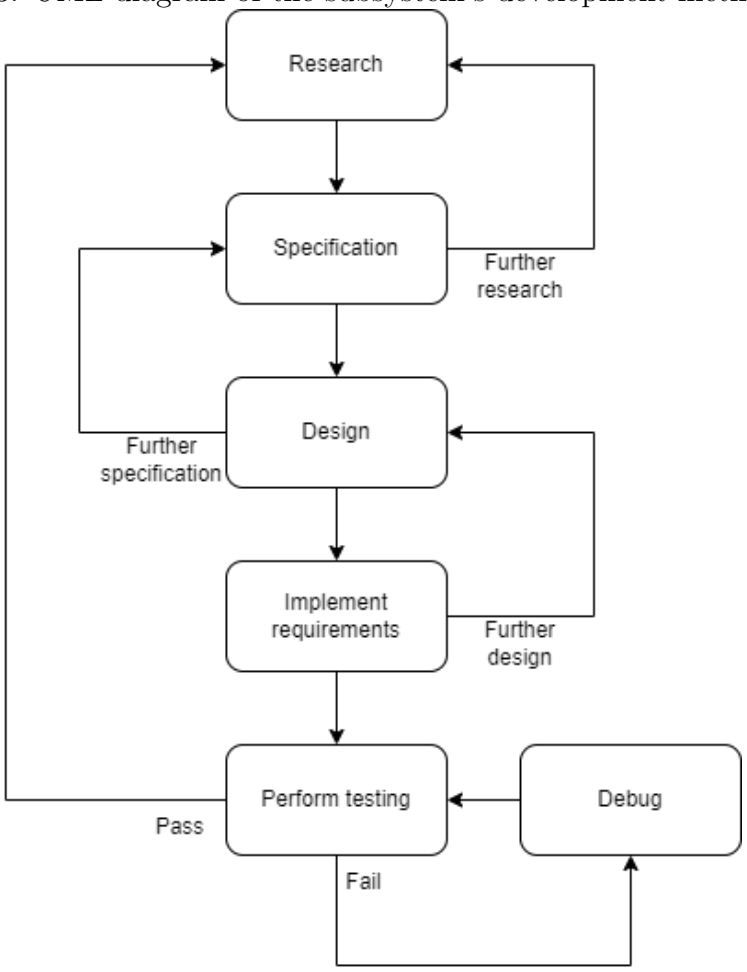
In the future, a more precise ADC module could be explored to give the CC3200 more resolution when performing spectroscopy.

Communication with Servos The MCU will directly control GPIO and bit-bang values to the servo motors controlling the orientation of the solar panels when using functions from the class `Solar`. This method of control may be refined further in the future.

Telemetry No data that would be exclusively considered telemetry will be transmitted between AWS and the MCU (e.g. processor temperature). Instead, all "telemetry" values will be handled with interrupts and ISRs/functions on-chip, while current settings and sensor readings will be transmitted back to AWS.

Development Model An Agile development model will be used. Code reviews will be performed on an as-needed basis by a convening of members of the MCU subsystem and the web subsystem teams.

Figure 33: UML diagram of the subsystem's development methodology



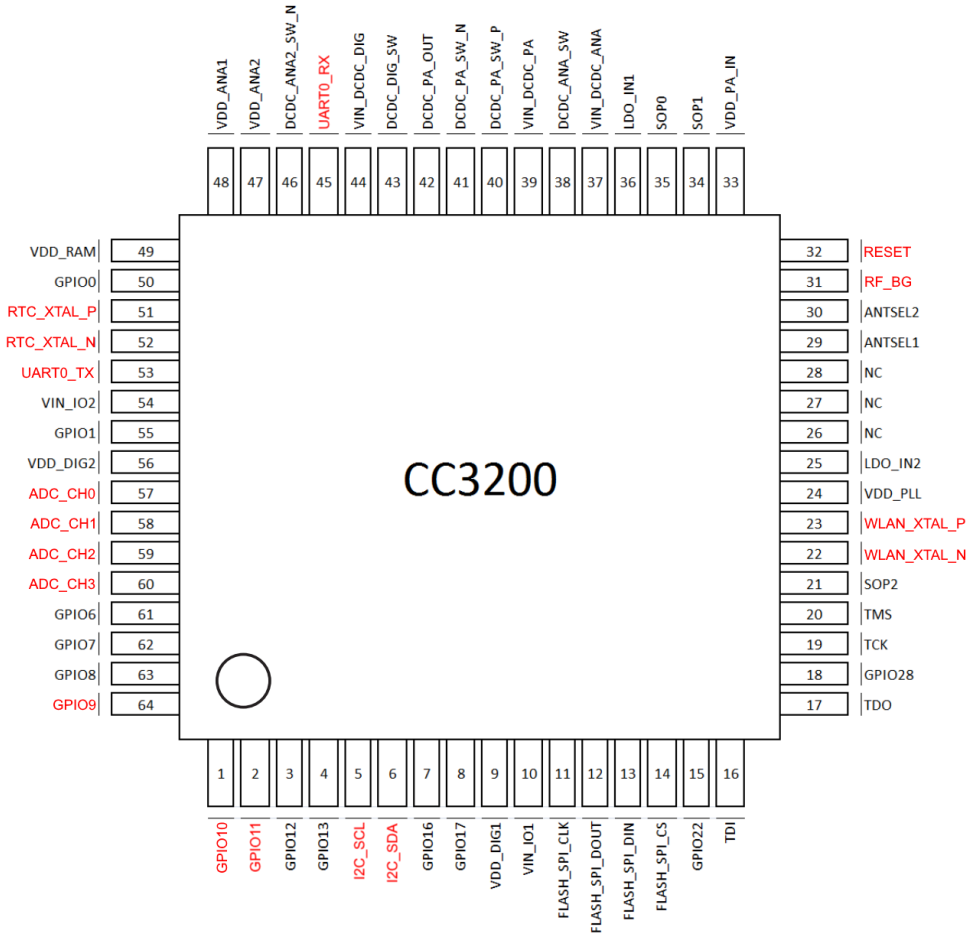
IDE and Git Texas Instruments Code Composer Studio v12 will be used to program, compile (via TI ARM compiler v20), and debug the C++-based project. GitHub will be used as a repository for the project, using Git for version control.

Power The MCU will be connected to power from the power subsystem detailed in subsection 5.2. It is expected to receive 3.3 V, and this will be fed to the integrated DC/DC converter on the CC3200. The converter is able to convert voltages from 2.3 to 3.6 V.

Pinout Diagram The CC3200 will connected to the following components as described by the pinout diagram shown in Figure 34[2].

[2]Red labels indicate pins will be connected to other components in a way that differs from the reference design.

Figure 34: CC3200 pinout diagram



LEDs The system shall be able to make use of its onboard LEDs for notifying the user or developers of system states. When the system is in active mode, the green LED shall be solidly illuminated. When the system is in low power mode, the green LED shall flash 1 time for a period of 0.5 seconds, every 2 seconds. When the system is in critical power mode, the red LED shall flash 2 times for a period of 0.5 seconds per flash, seperated by 1 second between each flash, every 30 seconds. When the system is in shutdown mode, no LEDs shall be active. When the system is starting up, the green and red LED shall be solidly illuminated until the startup sequence is completed and the system transitions into a different power mode. The yellow LED shall solidly illuminate while receiving data through the RF antenna for the duration of the reception. The yellow LED shall blink 1 time every 0.2 seconds for a period of 0.1 second while transmitting data through the RF antenna for the duration of the transmission.

Table 23: MCU LED operation

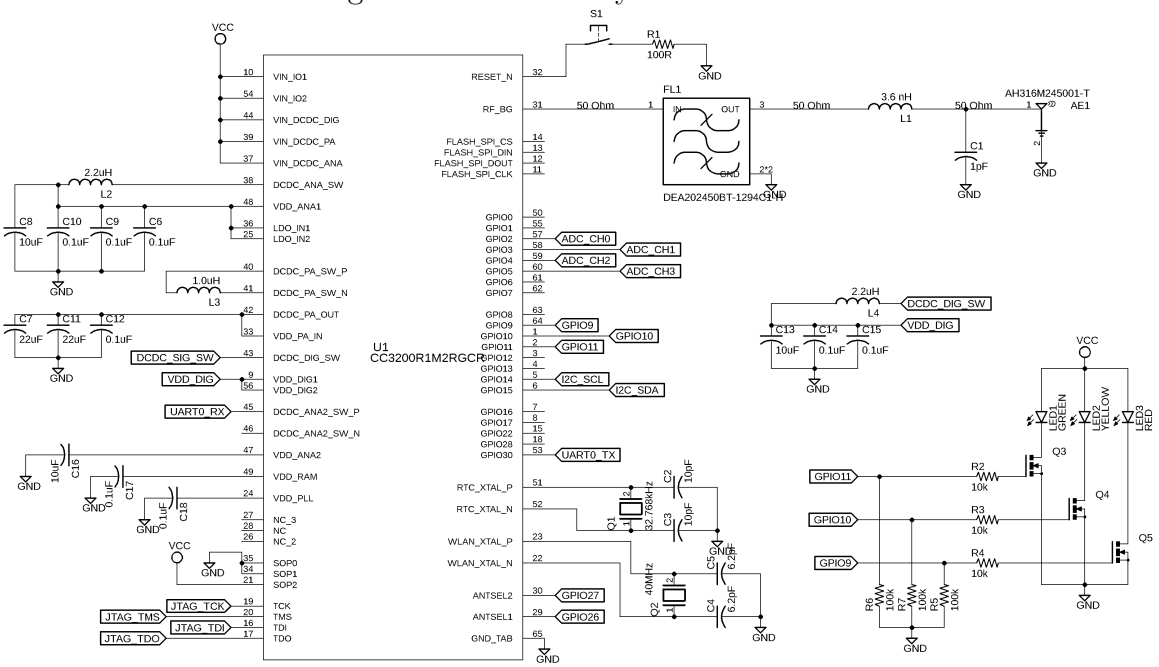
Power Mode	Red LED	Yellow LED	Green LED
Active	Not active	RX: Solidly illuminated, TX: 1 flash of 0.1 s, every 0.2 s	Solidly illuminated
Low	Not active	RX: Solidly illuminated, TX: 1 flash of 0.1 s, every 0.2 s	1 flash of 0.5 s, every 2 s
Critical	2 flash of 0.5 s, 1 s seperation, every 30 s	No connection: 1 flash of 1 s, every 2 s	Not active
Shutdown	Not active	Not active	Not active
Startup	Solidly illuminated	Not active	Solidly illuminated

Watchdog Timer The CC3200 has an onboard watchdog timer used to detect and recover from freezes, crashes, or lockups. When the watchdog timer reaches zero, the watchdog module shall trigger an interrupt. The ISR triggered by the interrupt shall reset the watchdog timer. In the case the ISR is not able to be triggered by the interrupt (e.g. system freeze), the watchdog module shall reset the system. After the system is reset by the watchdog module, the system shall start up normally.

Printed Circuit Board Our controller subsystem will be developed and implemented on a Texas Instruments CC3200SF-LAUNCHXL LaunchPad development board with a plug-in daughterboard hat containing sensing and other components. In an ideal scenario, our team would like to develop a custom PCB for our controller subsystem—however, decreased supply and increased lead times, as well as the extra time needed to develop, test, and implement a PCB forced our hand to use the development board in the final product.

If Texas Instruments' global integrated circuit supply increases and MCUs become more available, and our team is on-time with project goals and objectives, we would like to explore design of a PCB for the controller subsystem. Because the product is using the reference design currently, our team does not yet have a custom schematic for the controller subsystem. Thus, our product's schematic may be found in Figure 35.

Figure 35: MCU subsystem schematic

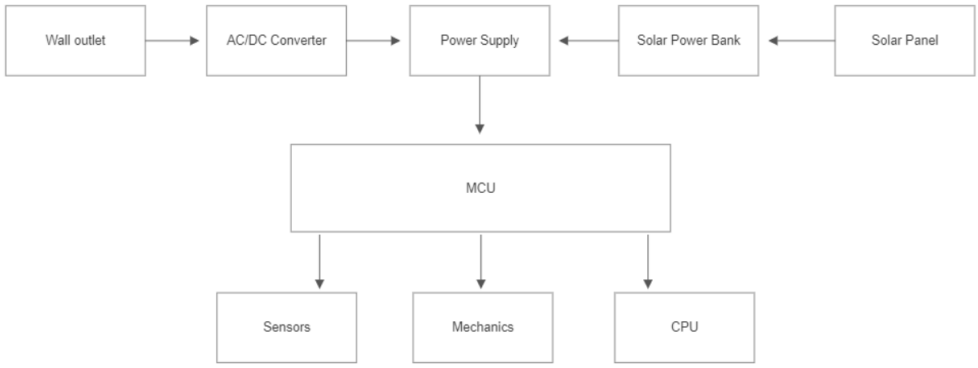


5.2 Power Subsystem

The power system is an important part to any electrical device or component that requires any amount of power. It simply starts from a power source such as a battery or wall outlet then is converted into energy to operate what is being used. At the minimum this model is designed to be an independent system, having the capability to operate on its own. The way that can be achieved is through solar power.

Solar power has been such a strong growing source of energy and will play a vital role in our system. Power is collected through the solar panels and then regulated through the charge controller. The power is regulated through the charge controller so that the battery is not being over charge, which could potentially damage it. Of course, power collected from the solar panels needs to be stored in a battery, which will also be used to power our system. From there, voltage needs to be regulated once more for the microcontroller and other components are powered properly. The block diagram in Figure 36 shows the flow of power through the system.

Figure 36: Power subsystem block diagram



The block labeled "Mechanics" in Figure 36 refers to the solenoid valve for controlling water flow as well as the control scheme for actuating the solar panels to maximize power efficiency.

5.2.1 Solar Panel Control

To get a greater degree of accuracy the use of stepper motors are used. First, based on the choice of solar panels we get the mass and dimensions of the solar panels, these are .76kg and .336m x .2m respectively. This is important for calculating the torque. We only need to provide two degrees of freedom, one about the short axis (the horizontal axis that bisects the .2m side) and the vertical axis (the axis at the intersection that bisects the two sides of the panel). From the length and mass we can calculate the force per unit length:

$$\frac{.76\text{kg}}{.336\text{m}} = 22.62\frac{\text{N}}{\text{m}} \tag{4}$$

Now, knowing that torque is $F \times d$ we can formulate the torque for any given angle about this axis in the following:

$$\int_0^{.168} 22.62 \times x \times \sin\theta \, dx \tag{5}$$

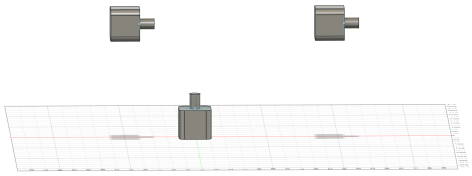
Equation 5 gives the torque for any given angle. Something you might have noticed is that this is for only one side of the panel. The full torque about the central axis is as follows:

$$\int_0^{.168} 22.62 \times x \times \sin\theta \, dx - \int_0^{.168} 22.62 \times x \times \sin 180 - \theta \, dx \tag{6}$$

From Equation 6 we can find the maximum and minimum values of the torque about the axis by finding the crossings of the first derivative with respect to θ and finding the concavity in the second derivate with respect to θ. Doing so proves the intuition that there is no torque around either axis, this equation evaluates to 0 for all θ.

To save on power Figure 37 shows that two motors will work to rotate the panels about the short axis while the center axis is actuated by a single motor and belt system.

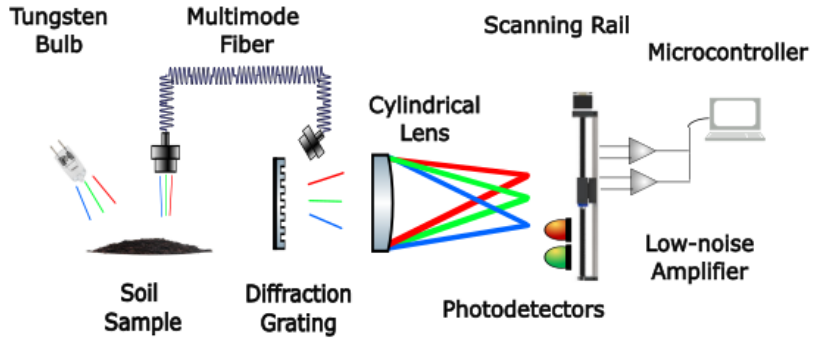
Figure 37: Stepper motor configuration



The reason we have to use two stepper motors for the short axis rotation is that because as the panels rotate about their central axis any rod or pulley or tensioner system would come out of alignment. Another advantage of this due to the lack of torque in the system is that using a motor controller the two motors can be driven with a single H-bridge with a minimum penalty to power, this will ensure that both panels are always pointed at the same angle in the desired direction.

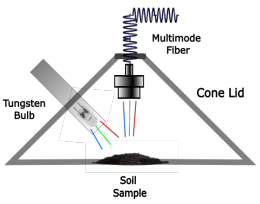
5.3 Sensing Subsystem

Figure 38: Sensing subsystem block diagram



Overview The system determined what automation was needed to optimize the gardening environment using a Near Infrared Optical Spectrometer. The spectrometer probed the soil with a broad band light source, exciting electromagnetic waves in the weak bonds of the material. These waves were collimated into a fiber optic cable and transmitted into a closed chamber. The beam was guided into the housing and reflected off a diffraction grating, which separated wavelengths by angle. A focusing lens positioned the separated spectrum on a surface several inches away. Then Visible and Near Infrared optical sensors were scanned across the surface. At every position, the system microcontroller recorded the sensor positions and amplitudes. The data determined what actions the system would take, whether mechanically or via messaging.

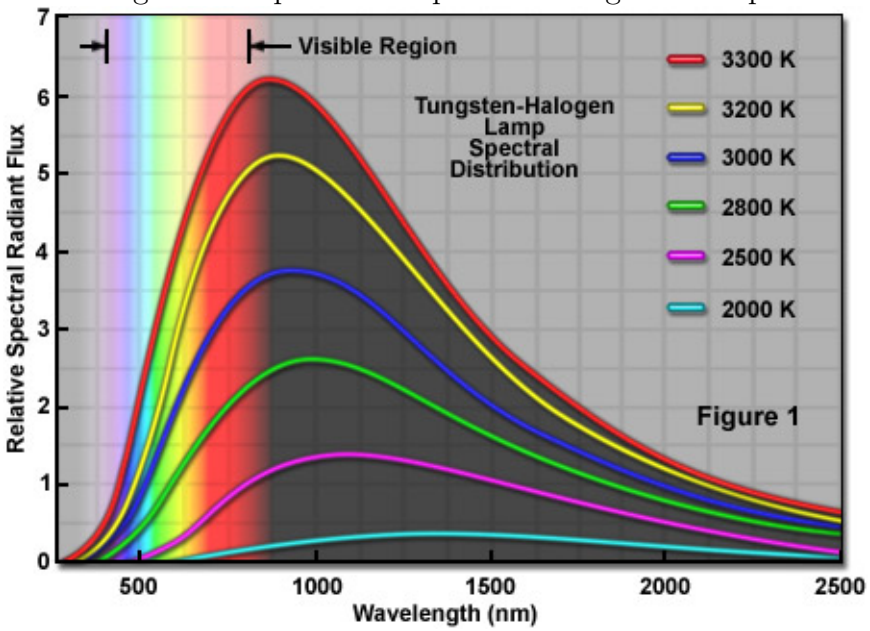
Figure 39: Light Collection System



5.3.1 Light Collection

The light from the bulb shone on a patch of soil and reflected onto the fiber head collimator. In order to maximize the effect of reflectance on the sample, the bulb was seated in a reflective metal tube which shielded reflections from entering the fiber head.

Figure 40: Spectral Output of a Tungsten Lamp



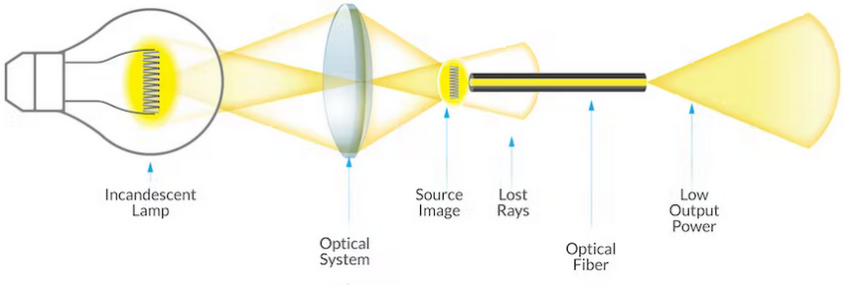
5.3.2 Light Source

The spectrum of interest was from 400nm to 1700nm, however, not all data points were necessary for the system decision matrix. This initially suggested the use of LED lights as a cheap, low power solution, but the number, target wavelengths, and spectral widths of LEDs created several challenges. Broad band Super Luminescent Diodes dramatically reduced the complexity of the problem, but they were an emerging technology with high material cost, placing even the cheapest devices orders of magnitude outside of the project budget. Thankfully, the desired effect could be achieved with a tungsten lamp. Tungsten was a material which emitted a broad range of frequencies, covering the spectrum of interest and then some.

Soil Conditions The optical signal emitted by the soil was weak, so it was essential that the spectrometer was influenced as much as possible by the content of the soil and as little as possible by the topology of the surface. A well-mixed, smoothly flattened, gently compressed soil sample provided the best conditions for constructing a spectrograph. If there were ridges or depressions along the sample, too much or too little optical signal would pass into the spectrometer, creating an apparent signal strength for wavelengths that was not representative of the soil emission. There were several ways to control these conditions. One was to require the user to retrieve a sample for each scan and deposit it into a spectrometer bay. Another was to require the user to flatten the soil with a spatula or other implement and position the scanner input near the soil. These reduced ease of use, which was a target for the project. Another issue was that the position of the signal input and the tungsten light source

relative to the soil sample had to be consistent to avoid signal error, for the same reasons as above. In one study[12,], this issue was resolved by building a transparent block with slots to hold the light source and the input beam. Their design was represented below, showing the block attached at the rear of a plow chisel for clearing soil while scanning. We solved the problem of soil topology and light position the same way, by mounting the light source and input fiber in a block of some transparent material like acrylic.

Figure 41: Coupling a diffuse light into a fiber

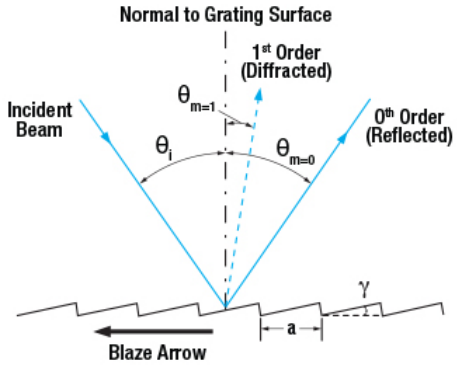


Fiber Optics Fiber optic cables were a standard input for spectrometer devices since they allowed nearly unlimited flexibility for the position and orientation of the target area relative to the device. The basic ray trace for a lens coupling light into a fiber was shown below. Fiber cores faced some basic limitations when it came to coupling. The maximum possible coupling efficiency could be achieved by pressing the optical surface of the fiber core up against a light source with equal random output in every direction (reference). If the fiber was moved some distance away from the source, less light would impede on the core-air interface. If the core diameter was increased, this would increase the light incident on the core. Light moving in random directions would also strike the fiber core at various angles, however, not all angles of incidence would couple into the fiber. Only rays within the numerical aperture of the fiber were accepted. This was why installing large lenses at the end of the fiber did not increase the maximum possible amount of light that could be coupled. The maximum angle was determined by the fiber, rather than the collimator.

This project involved a large diffuse source, the illumined soil. The Fiber collimator was attached to the fiber cable via their SMA connectors, and then the collimator was set in an acrylic block so that it rested 8.06mm above the soil. The block also had a slot for the Light source to illuminate the soil at an angle, similar to the arrangement of the source and sensor in a computer mouse above a mousepad.

The signal traveled through the fiber and up into the spectrometer housing, where the other connector of the cable was mounted in place. Another collimator took the output beam and collimated it so that it propagated through free space into the housing. The collimator had an output beam diameter of 1.7mm. This planar wavefront struck the diffraction grating.

Figure 42: Grating Angular Calculation

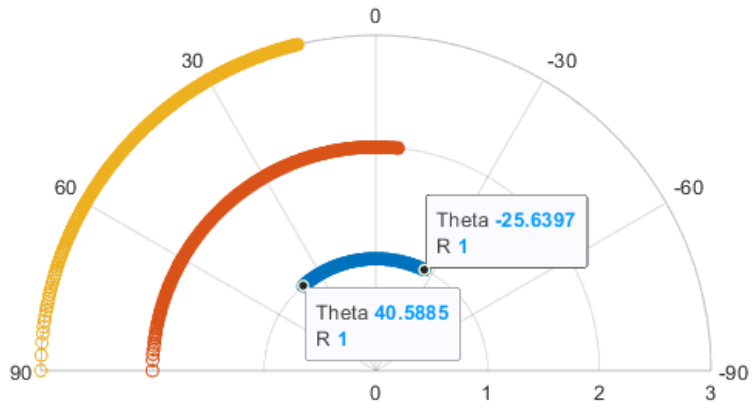


5.3.3 Diffraction Grating

The reflective diffraction grating was a glass mirror with a thin layer of metal deposited on the surface. 1200 lines per millimeter were scored out of the metal horizontally. When a planar wavefront hit the surface of the grating, it reflected off. The confinement of the wave on the surface of the material induced a change of direction proportional to the frequency of the beam, separate from the direction of reflection, equal to the angle of incidence from normal. In order to direct the diffracted beam away from the incoming beam, the grating was placed at an angle of 45 degrees. This changed the angle of incidence and determined the angular range of the system. To increase the working range of the grating, the lines scored into it were blazed at an angle so that light approaching the surface from the angle struck the scored metal at zero degrees from normal. The angular spread of the system ranged from the lower angular wavelengths around 400nm up to 1700nm and beyond.

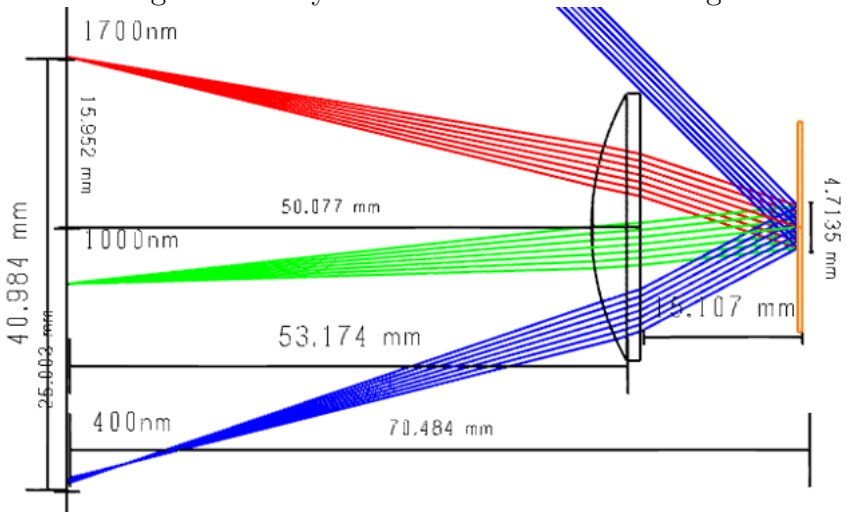
$$a[sin(\theta m) + sin(\theta i)] = m\lambda \tag{7}$$

Figure 43: Diffraction angle calculation



This meant that the spread of the diffracted light was 38 degrees in and 21 degrees out from both the near edge and the far edge of the collimated beam on the surface of the grating. In order to capture this light and sort it so that the scanner could proceed linearly through each band, a focusing optic was needed that covered the full angular range.

Figure 44: Ray Trace of Diffraction Grating



The beam coming out of the fiber had a nonzero width. When the 1.7mm diameter beam hit the surface of the grating, since the grating was at 45 degrees from the beam, light was propagating from an area with a diameter larger than 1.7mm.

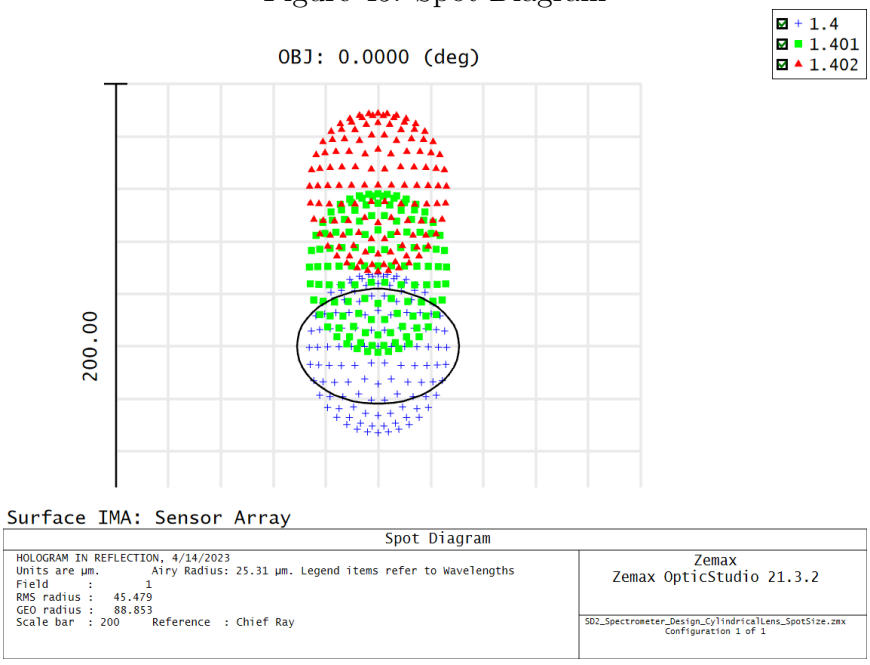
5.3.4 Focusing Lens

The diffraction grating diverged the beam away from the sensor surface. This divergence could be corrected with a lens, focusing the divergent light to a spot the size of the sensor or smaller. There were two lens designs that could work, spherical and cylindrical. Spherical lenses were polished with a radius of curvature along both the horizontal and vertical axes. The advantage of a spherical lens was that they were generally cheaper to manufacture and available in a wider variety of sizes and shapes. The spot size of a circular beam passing through a spherical lens was determined by the distance from the focal length, and the spot was circular.

Cylindrical lenses were cut with a radius of curvature along the horizontal axis, but flat along the vertical axis. A circular beam that passed through a cylindrical lens was focused to the shape of an ellipse, allowing for more vertical flexibility of alignment. Plano-cylindrical lenses offered another advantage; they had a flat bottom, perpendicular to their planar back. This meant they could be stood upright and pressed against a flat surface, dramatically reducing the complexity of the optical mount required to hold them in place.

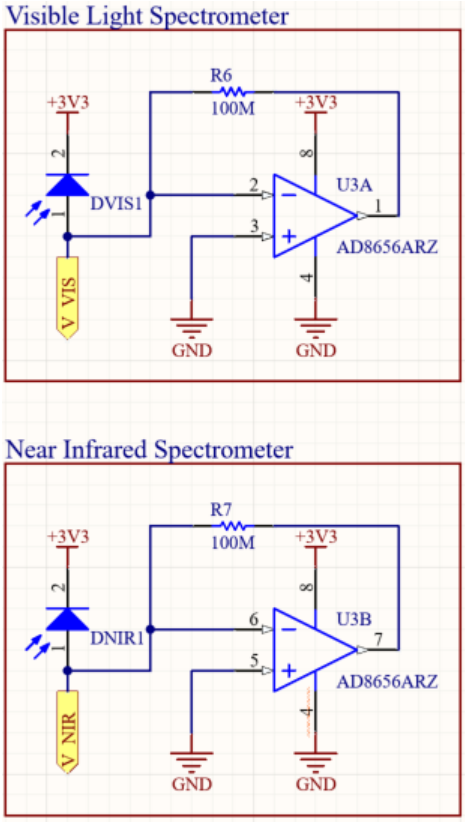
The dimensions and position of the lens were determined by two things, the angular range of the spatially separated beams coming off the reflective diffraction plate, and the width of the scanning region. The lens was also constrained by the angle and width of the input beam, as shown in the ray trace below. If the width of the lens came within a few wavelengths of the beam approaching the diffraction plate, the beam would diffract around its edge.

Figure 45: Spot Diagram



The spot size of an idealized raytrace yielded 27μm, with 30μm between spot centers. This was excellent, since the airy disk suggested by the ray trace was within an order of magnitude. This meant the arrangement of the components did not raise the lower limit of the spectral resolution above 2nm, although it was doubtful the nonideal system would achieve such precision.

Figure 46: Sensing Schematic

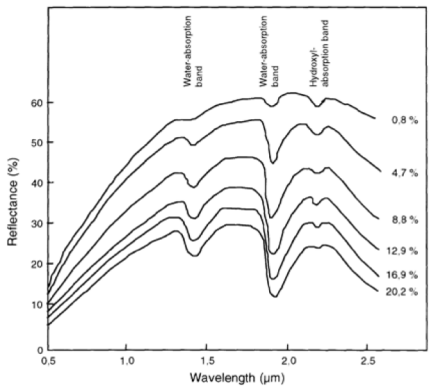


5.3.5 Circuitry

The Photodiode worked by converting a small portion of the incident light into electrical current across the face of the semiconductor. There were three components to the sensing circuitry. First, there was a voltage divider. This voltage divider provided a clean and stable 1.65V to the anode of the photodiode. The photodiode was reversed bias to create a current when detecting light. From there the second part of the circuit began which was a current-to-voltage converter. The photodiode emitted a current based on the power of light that hit the surface per the area that it hit. Using a 100 MΩ resistor converted current to voltage at a rate of 1 pA to 100 μV. There were two issues with this however, first the ADC only had 12-bits and as discussed in the controller section, this resulted in about half a milli-Volt per step. This meant that the system lost about 4x the resolution of the sensor. This is where the last part, the instrumentation amplifier took effect. The instrumentation amplifier took

the difference between the voltage output and a reference voltage and scaled all the parts. For example, if the output voltage of the current-to-voltage converter was .3V (which happened to be the dark current of the visible light sensor) subtracting the voltage of the dark current and scaling the remainder helped.

Figure 47: Soil Spectrograph



Spectrum of Interest Soil is a heterogeneous combination of organic and inorganic substances, and the balance of soil nutrients contributes directly to the quality of the garden environment. Each substance contributes to the total radiative emission of the soil, so by measuring the traces in a sample with unknown quantities and comparing it to traces from a sample with known quantities, the nutrients can be estimated. Soil Carbon content, Moisture Content, Phosphorous, and pH can all be detected through reflectance within the 400nm to 1700nm range [12]. The spectrograph looked something like the one above.

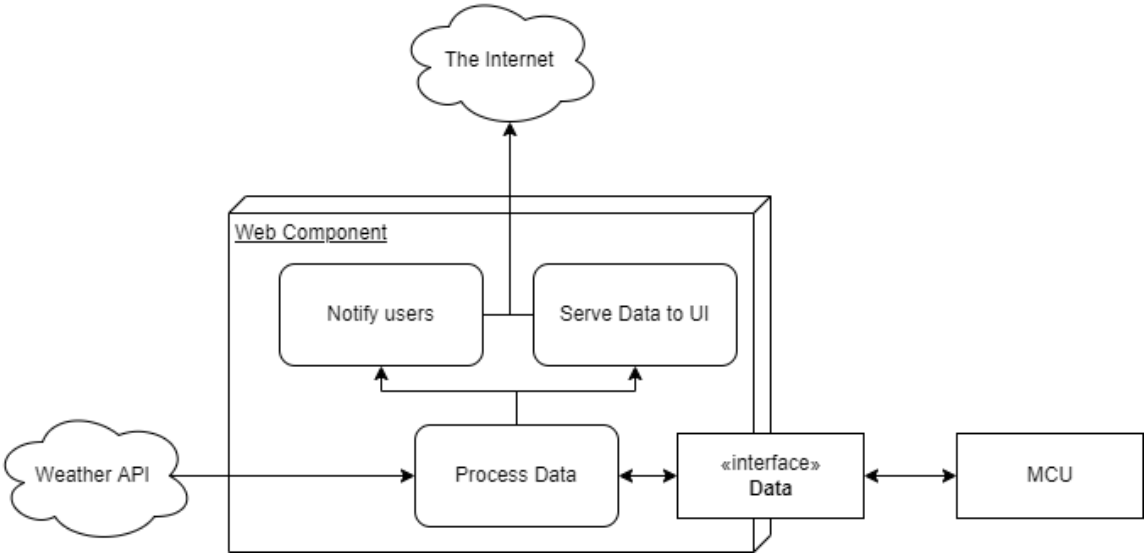
5.4 Web Subsystem

The web subsystem will be split into several distinct parts to achieve several goals. First and foremost, the web will communicate over TCP with the MCU to get data, process the data and send commands back to the MCU. Secondly, there will need to be a database in order to log data and be able to support multiple of these garden beds in a scaled solution. Third, there needs to be a user interface that allows the user to set settings and read the data about their garden bed(s). Lastly, the web component of this project will communicate using HTTP requests with a weather service to get upcoming weather such as rain, sunlight, and freeze warnings in order to help the user with maintaining their plant bed.

The block diagram below shows a high level overview of flow of data between different sources. The microcontroller and web component share a two way interface to transmit controller data to the web and commands back to the MCU. Refer to the MCU subsection for more details on the commands. Part of the this component will

be a way to process the data received from the weather service and the MCU to serve it to the UI and thus the user.

Figure 48: Web component block diagram



Section 3.2.3 has information on the technologies in the stack while the following sections will go into more detail about how these choices will be used in order to achieve the goals outlined above. These choices are a React frontend as it is a technology that the team is familiar with and has good support and allows for an event-driven UI as would be useful to serve the polling data collection of the backend. Express will be used as a middleware and routing layer for the user interface to the server backend. This choice was made again to keep the language in JavaScript and the vastness of suppport on the platform. Express also supports Socket.js which will be necessary for communicating over TCP with the microcontroller. MySQL will be used as a relational database as the data is relational. SQL databases also typically serve data-driven applications better over service-driven.

5.4.1 Server Backend

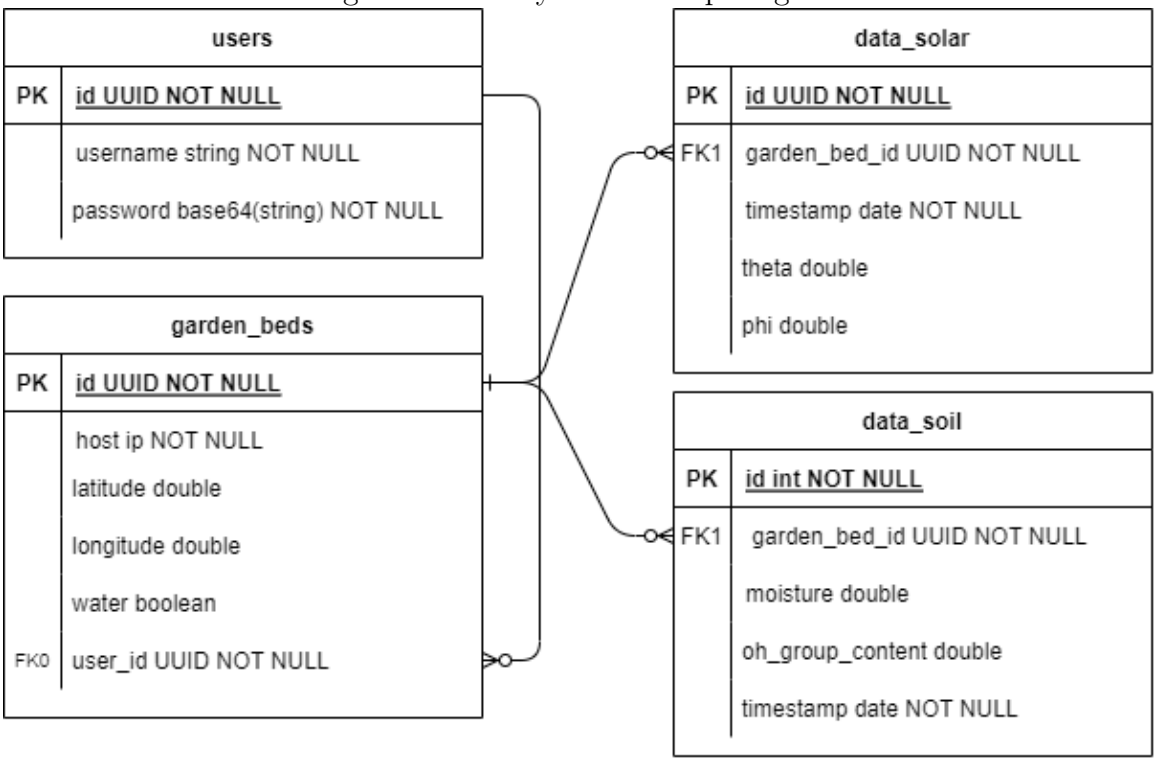
As mentioned above, the backend will consist of two parts, a MySQL database as this is a free option and serves the purposes well especially for logging data, as well as an Express backend which will serve all the API endpoints for the user interface as well as a serve to create TCP connections with the microcontroller.

Communication This section will cover how we plan on implementing communications with the microcontroller. As covered in 5.1 these two systems will communicate over TCP. The means of implementing this will be through sockets. This backend will serve as a server to make connections to. Through the use of the Socket.io library, it

will be nearly superficial to create this connection and read raw data and process it. Once data is received will be stored in the database for computation.

Database The database will be relational. Relating a plant bed and its current state to the logs of data stored within the database as well. The way this will be implemented is there will be a table, `garden_bed` which will store a garden bed and the client socket information. From this point, the ID of the `garden_bed` will be used in a lookup in other tables such as `data_solar` or `data_soil` to be able to present this data to the user in the UI upon request. These data tables will have a timestamp of their creation to be able to lookup the most recent data and retrieve those items and also to create a log and graphs if we so chose. Figure 49 has some details on the type of data and where it can be found within the database given the design above.

Figure 49: Entity relationship diagram



Another option for the organization of the database is to create a table for each plant bed, which for this project would only be one, that holds all the data and logs regarding to that plant bed. This solution is slightly harder to implement. It is harder to implement because ORMs (Object-Relational Mapping) do not like the dynamic nature of tables. ORMs know the schema of a database table per the name, and when the database becomes highly dynamic in this architecture, the ORM typically

requires more work to setup. For more detail on ORMs refer to the technology section of this document.

Data Processing All data will be processed on the EC2 instance within the server. Using some heuristic methods and research into peak conditions for different plant types, the server will be able to determine different actions to be made on the plant bed in order to promote growth.

Accounts and Plant Bed Linkages Each user account will be linked with potentially multiple plant beds through the database designed above. The backend will validate user logins to endpoints using JWTs in cookies in order to validate each user that logs in.

API Endpoint Design Representational State Transfer (REST) is a paradigm that indicates a certain API is platform agnostic thus allowing for any number of clients to call the API so long as they follow the standards that are implemented. REST is typically implemented through HTTP requests, this is the case for our project as well. HTTP comes fully packaged with various methods of calling different URIs (unique resource identifier). The best way to represent this is that a top-level domain refers to the server while appendages to the domain denote the resource and the verb. For example, `http://autogardenbed.com/garden`, is the location of the garden resource. From there we can use HTTP methods to instruct the server on what action to take. The most common operations are POST, GET, PUT, DELETE which corresponds to create, read, update, and delete. Our API will be no different.

The URIs will follow the convention of `.../[noun]/[verb]/{identifier}`. The "noun" refers to the type of object that the request will interact with. In our case this should be something like "garden" or "data". Verbs will be used rarely but a situation that might arise is where the user wants to send a command to their garden bed. This will be accomplished through the verb portion of the URI. Table 24 gives a high level overview of the URIs, the associated methods and what they will accomplish.

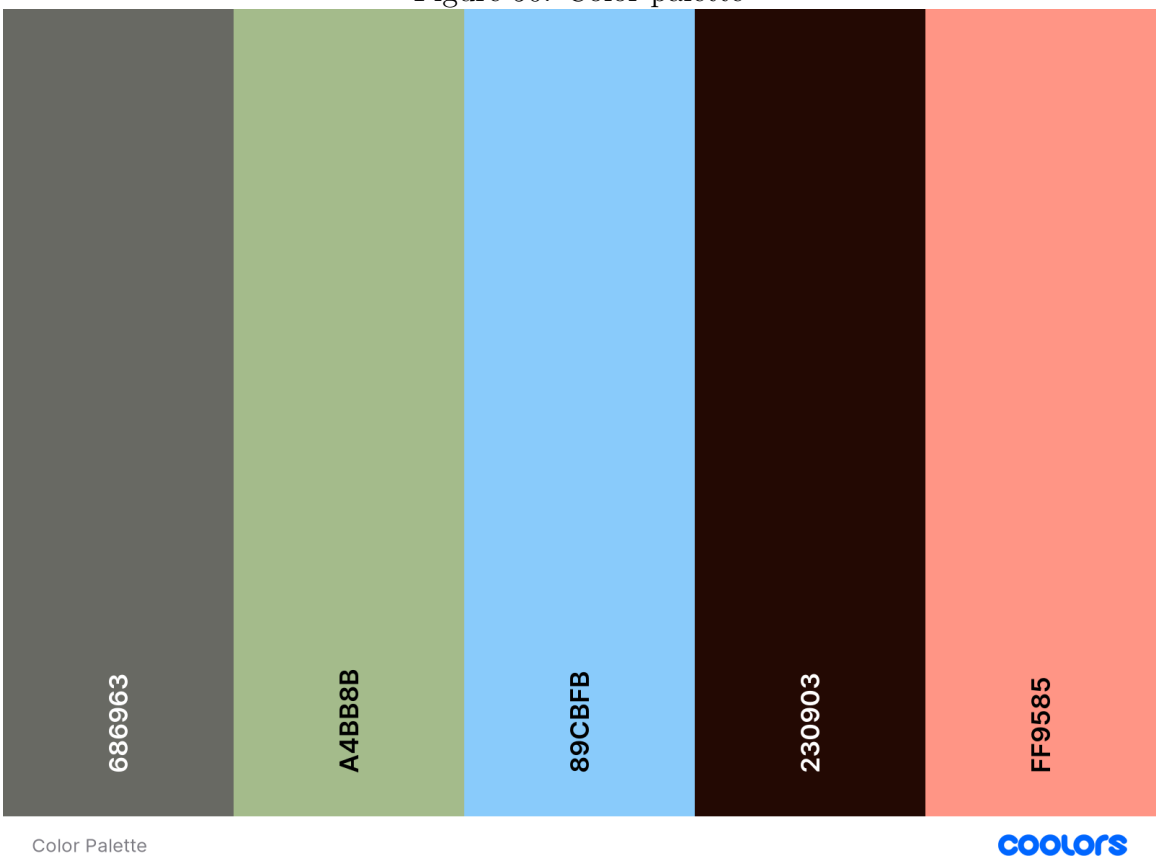
URI	HTTP Method	Function
.../user	POST	Create a new user
	GET	Get a user's information such as a list of associated garden beds
	PUT	Update user information such a password
	DELETE	Delete user account
.../user/auth	POST	Takes user credentials and validates against database; returns a JWT for validation in other endpoints
.../garden	POST	Creates a new garden bed
	GET	Get data on a garden bed, join garden bed info with data
	PUT	Update garden bed information, such as IP, location, etc.
	DELETE	Delete the garden bed from the database and ensure no records are orphaned
.../garden/{garden_id}/command	POST	Sends the command specified by the request object to the garden bed

Table 24: URI table

5.4.2 User Interface

The below hex values were selected using a color picking service that guarantees a palette that is useable for users who might be colorblind.

Figure 50: Color palette



For user experience, catering to colorblindness is important. From left to right the purpose of each color is: component color, emphasis, highlight, background color, error.

To design the user interface we will be using Figma. Figma offers some prototyping and features in order to quickly build a user interface. In conjunctions with React as a framework and CSS, putting everything together should be relatively trivial.

5.5 Subsystem Integration

The integration of all the subsystems is centered around the control subsystem. In the following sections, the team will discuss how these integrations will occur as the design process was centered around creating individual black boxes for each system to work against.

5.5.1 Sensing

The sensing subsystem has three key components for the control subsystem to interact with: the linear rail and carriage the spectrometer is mounted to, the laser generating

photons for the spectrometer to receive, and the spectrometer itself.

Linear Rail and Carriage The spectrometer will be mounted to a carriage that may slide back and forth on the linear rail. The position of the carriage on the linear rail determines the wavelength of light sampled, and therefore the wavelength can be interpolated by the position of the stepper motor controlling the spectrometer carriage. It is a natural conclusion that the microcontroller will be able to direct the spectrometer to measure a specific wavelength by moving the stepper motor a certain number of predetermined steps. The microcontroller will sample the entire spectral range and compile this data to prepare to send to the web subsystem for processing.

(8)

Laser The laser mounted to the Carriage will be commanded to power on by the microcontroller whenever the microcontroller wishes to obtain a sample from the spectrometer. The laser itself draws a minimal amount of current and is compatible with the voltages regulated and output by the microcontroller, and will simply be connected to one of the microcontroller's GPIO pins.

Spectrometer The spectrometer itself will be stationary relative to the spectrometer's carriage. The spectrometer is a passive instrument that generates current depending on the flux density on the sensor. Due to the position of the spectrometer, this flux density is based on the strength of the specific wavelength emission. This current is connected to the sensing subsystem circuitry described in 5.3. This circuitry will proportionally convert the current to a voltage, and the output of the circuit will be connected to an analog-configured pin of the microcontroller, which will be taken as an input to the microcontroller's analog-to-digital converter. The ADC will convert the voltage to a value able to be measured in programming. The current (and by extension, voltage) level measured will be paired with the wavelength and transmitted by the MCU to the Amazon EC2 instance.

5.5.2 Power

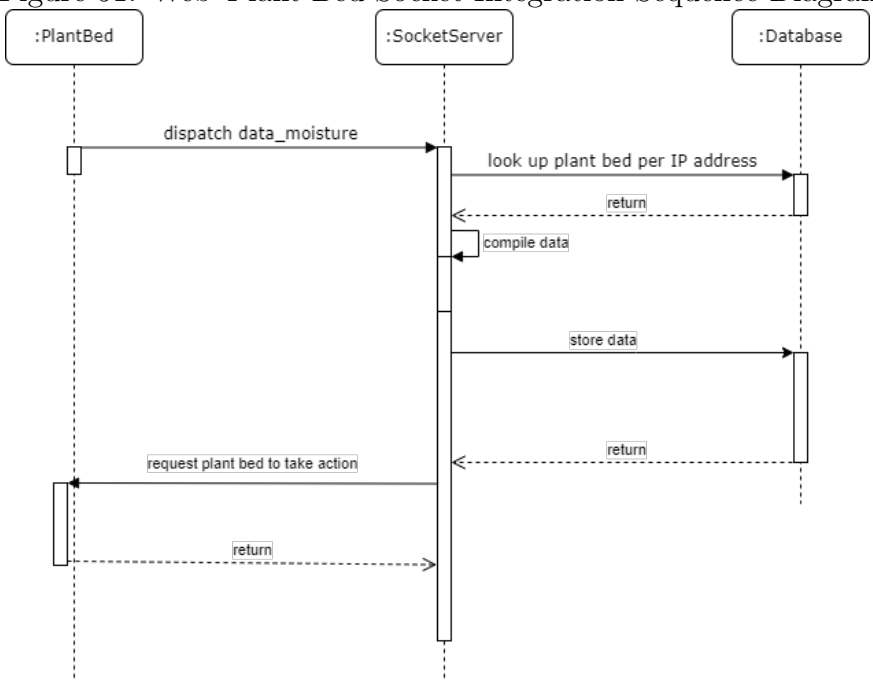
The charge controller will deliver voltage of the battery to the microcontroller via

5.5.3 Web

The web and control subsystems interact via two way WebSocket protocol (4.1.7), which will utilize TCP (4.1.3) as the transport layer protocol. Section 5.1 talks about the format of the raw texts in the packets. The web subsystem will be deployed to a fixed domain so the memory of the control system can use a fixed host for the Socket server. The socket server will always start by processing the address information of the incoming socket, and grabbing previous data from the database accordingly. The

socket server will process the data, storing it in the database for reporting on later. If necessary, the socket server will send a request to the plant bed to do some function. This is laid out in Figure 51.

Figure 51: Web–Plant Bed Socket Integration Sequence Diagram



6 Testing

Incremental unit testing will be used during the development process to ensure each subsystem fulfills design requirements as described in **??**. Controller subsystem testing will be performed in subsection 6.1, power subsystem testing will be performed in subsection 6.2, sensing subsystem testing will be performed in subsection 6.3, and web subsystem testing will be performed in subsection 6.4. After subsystems have fulfilled unit testing requirements, integration testing will be performed with the criteria defined in subsection 6.5.

6.1 Controller Subsystem Testing

The MCU subsystem will undergo a littany of testing. The electrical characteristics, as well as hardware and software behavior will be tested to ensure a working product. Any use of "system" or "the system" in this section refers exclusively to, and the entirety of the MCU subsystem.

6.1.1 Electrical Characteristics

The MCU subsystem shall adhere to the following electrical characteristics related to power draw and voltage input in the following states. Unless noted, all voltages will be in direct current, and all conditions will be tested at room temperature (77 °F).

General Requirements The system shall be able to accept a voltage of -0.5 to 3.8 V into V_{BAT}, -0.5 to 4.3 V into any digital IO-configured pin, -0.5 to 2.1 V into any RF-configured pin, and -0.5 to 2.1 V into any analog-configured pin without damage. The system shall be able to accept a current of 0 to 620 mA into any pin without damage. The system shall be able to regulate voltage going into V_{BAT}, and provide 3.3 ± 0.3 V on its 3V3 rail to any components in the sensing subsystem that may require it.

Results TBD.

Operating Temperature The system shall be able to operate nominally within the temperature range of 30 to 140 °F. Crystal oscillators shall fall within the thermal drift ratings provided in the MCU technical reference material.

Results TBD.

Shutdown Power Draw The system shall not draw more than 10 μA in shut down mode. The system shall not consume more than 100 μW in shut down mode.

Results TBD.

Critical Power Draw The system shall not draw more than 1 mA in critical power mode. The system shall not consume more than 5 mW in critical power mode.

Results TBD.

Low Power Draw The system shall not draw more than 100 mA while receiving or idling in low power mode, and shall not draw more than 250 mA while transmitting in low power mode. The system shall not consume more than 500 mW while receiving or idling in low power mode, and shall not consume more than 1.25 W while transmitting in low power mode. The system shall be able to sufficiently power any components in the sensing subsystem in low power mode.

Results TBD.

Active Power Draw The system shall not draw more than 150 mA while receiving or idling in active mode, and shall not draw more than 300 mA while transmitting in active mode. The system shall not consume more than 750 mW while receiving or idling in active mode, and shall not consume more than 1.50 W while transmitting in active mode. The system shall be able to sufficiently power any components in the sensing subsystem in active mode.

Results TBD.

Startup Power Draw The system shall not draw more than 750 mA within the first 20 seconds of powering on or starting up the system. The system shall not consume more than 4.00 W within the first 20 seconds of powering on or starting up the system.

Results TBD.

Solar Panels and Battery System The system shall be able to operate nominally solely powered off of the solar and battery subsystem detailed in subsection 5.2. The system shall not require any external power to operate in any modes (e.g. startup, active, etc.).

Results TBD.

6.1.2 Hardware Behavior

The MCU subsystem shall pass the following tests related to use of the MCU's on-board hardware, peripherals, and supported operations. The programming developed and test the below criteria will be intended to only perform testing on the below modules, and may or may not be present in the product software build.

Upload Programming The developers shall be able to successfully upload programming (software) that meets the requirements detailed in subsection 5.1. The developers shall be able to successfully upload programming that allows and/or performs unit testing (as detailed in this section, subsection 6.1). Programming uploaded to the MCU shall execute either upon powering the system, upon command, or when executing an ISR to start up the system.

Results TBD.

I2C The developers shall be able to initialize I2C. I2C shall function nominally if used for communication with any other modules in the overall garden bed system.

Results TBD.

Onboard LEDs The system shall be able to make use of its onboard LEDs for notifying the user or developers of system states according to the specification in subsection 5.1.

Results TBD.

Watchdog Timer The developers shall be able to initialize the watchdog timer. The watchdog timer shall behave according to the specification in subsection 5.1.

Results TBD.

WiFi Module The WiFi module shall conform to the 802.11b/g/n standards. The developers shall be able to configure the WiFi module to support 802.11b, 802.11g, or 802.11n in WiFi station mode. The developers shall be able to configure the WiFi module to support 802.11b, or 802.11g in WiFi Direct mode. The WiFi module shall allow the system to connect to either an open WLAN, or a WLAN secured by WPA2 (Personal or Enterprise). While connected, the WiFi module shall be capable of sending and receiving data at speeds of at least 10 kbps at least 80% of the time.

Results TBD.

GPIO The developers shall be able to configure the GPIO pins to the configurations available as described by technical reference documentation. The developers shall be able to set the state of the desired GPIO pins to the configurations available as described by technical reference documentation.

Results The developers are able to configure GPIO pins to the configurations available as described by technical reference documentation via the Texas Instruments SYSCONFIG system configuration tool.

Interrupts/ISRs The developers shall be able to configure the interrupts available on the MCU as described by technical reference documentation to trigger an ISR of their choice. ISRs shall run within 1 second of triggering the interrupt linked to the specific ISR. An empty ISR shall return the MCU to its previous state before the interrupt (i.e. clear any flags and return any registers to their previous state).

Results TBD.

ADC The developers shall be able to configure the ADC on the MCU to adhere to the parameters described in the technical reference documentation. The ADC shall be able to measure signals with a voltage of between 0 and 1.8 V. The ADC shall be able to accurately measure a voltage level down to a resolution of $\leq$1 mV. The ADC shall be able to measure at a sampling rate of $\geq$50 kilosamples per second per channel. The ADC shall support conversion on up to 4 channels. The ADC shall support the ablility to timestamp samples with the clock mentioned by the technical reference documentation.

Results TBD.

Reset Button The reset button shall be able to "power cycle" the system. The reset button shall not erase any programming uploaded to the MCU. The reset button shall not keep the system from starting for more than 1 second after the button has been released.

Results The reset button "power cycles" the system, does not erase any programming uploaded to the MCU, and does not keep the system from starting for more than 1 second after the button has been released.

Startup The system shall not take longer than 10 seconds to begin its programming upon sufficient power delivery to the MCU as described by the by the technical reference documentation. The system shall not perform the programmed startup sequence unless interrupted to do so, or if given sufficient power as described by the by the technical reference documentation after being shut down.

Results The system does not take longer than 10 seconds to begin its programming upon sufficient power delivery to the MCU. The system does not perform the programmed startup sequence unless the parameters described above are met.

Processor The central processor of the MCU shall maintain at least a 20 MHz clock speed while in active mode.

Results The central processor of the MCU is able to maintain an 80 MHz clock speed while in active mode.

Memory The programming uploaded to the MCU shall not take up more than 64 KB of space in memory.

Results TBD.

6.1.3 Software Behavior

The MCU subsystem shall pass the following tests related to use of the MCU to support application of the product. The programming developed and test the below criteria is intended to be the product software build.

Startup The microcontroller shall behave according to the specifications on startup in subsection 5.1 during its startup sequence.

Results TBD.

Shutdown The microcontroller shall behave according to the specifications on shutdown in subsection 5.1 during its shutdown sequence.

Results TBD.

Reset The microcontroller shall behave according to the specifications on reset in subsection 5.1 during its reset sequence.

Results TBD.

Networking The system shall be able to connect to the user's WLAN if it meets the following standards:

- A 2.4 GHz-based wireless network
- Contains no security or is secured with WPA2 (Personal or Enterprise)
- $\geq$-70 dBm signal strength
- Adheres to the standards of 802.11b, g, or n
- DHCP or statically-assigned IPv4 addressing
- Uses NAT and does not expose system to WAN
- Provides DNS to LAN devices
- Able to resolve and successfully connect to `8.8.8.8` and `google.com` within 5 seconds

Results TBD.

AWS The microcontroller shall behave according to the specifications on connecting to and communicating with AWS in subsection 5.1.

Results TBD.

Sockets The microcontroller shall behave according to the specifications on sockets in subsection 5.1.

Results TBD.

Solar and Battery The system shall monitor and update the solar array current and voltage output values at least every 10 seconds. The system shall behave according to the specifications on power modes and battery percentages in subsection 5.1.

Results TBD.

Sensing The system shall monitor and perform analog-to-digital conversion on the spectroscopic sensor values at least every 10 seconds while not in shutdown or critical power mode.

Results TBD.

6.2 Power Subsystem Testing

The Power subsystem is an important part in making sure that the whole system is fully operational. With various components such as the microcontroller, sensors, mechanical components and more, we need to make sure that each component is operating properly. This is a confirmation that the whole system is operating but also a test to see which parts work best with each other and what works best under given situations. One thing that will be tested is the voltage regulators to see which of the selected voltage regulators is most efficient and regulate power the best see fit.

6.2.1 Voltage Regulator

Providing power is a very important part to operating a system, but providing the right amount of power is key to make sure all the components are operating properly. As said in section 3 in the Power Subsystem, the LM317 (linear) and the LM2576 (switching) were selected to be tested and compared to see which would be the most efficient in regulating voltage. In this test, a simple circuit will be built with a battery, capacitors and the voltage regulator. The output voltage will be measured and displayed on an oscilloscope to verify which of the voltage regulators were able to regulate voltage more efficiently and successfully.

6.3 Sensing Subsystem Testing

6.3.1 Component Testing

Three components in the Sensing subsystem required testing on arrival.

- The Optical Sensors
- The Linear Actuator
- The Transimpedance Amplifier Board

A small transimpedance amplifier was constructed on a breadboard in the electrical lab to convert nA of electrical current to mV of electrical voltage. With an oscilloscope, the photodetectors were determined to be measuring optical power, with varying ambient light levels and light sources increasing the voltage within a measurable range.

Similarly, the stepper motor was connected to a Texas Instruments evaluation board with motor controller software. The stepper motor turned the screw, successfully carrying the mount in steps along the stroke length of the rail.

When the circuit board arrived it was installed with insulating spacers onto the metal mount on the linear rail and both components were optically isolated inside an opaque box. A fiber and diffraction grating were aligned inside to allow outside light to be dispersed in front of the rail. The photodetectors were successfully amplified by the circuit board, prompting the team to move on to permanent installation of each of the optical components.

6.3.2 Performance Testing

In order to evaluate the performance of the system, two tests were conducted.

The first test was an integration test to connect all parts of the Spectrometer subsystem and generate a spectrograph. For the purposes of this test, it did not matter what data was produced, only that the system was capable of recording data. The light collection cone was set on top of a large piece of flattened aluminum foil. See the two resulting datasets below.

Figure 52: Aluminum Test VIS Data

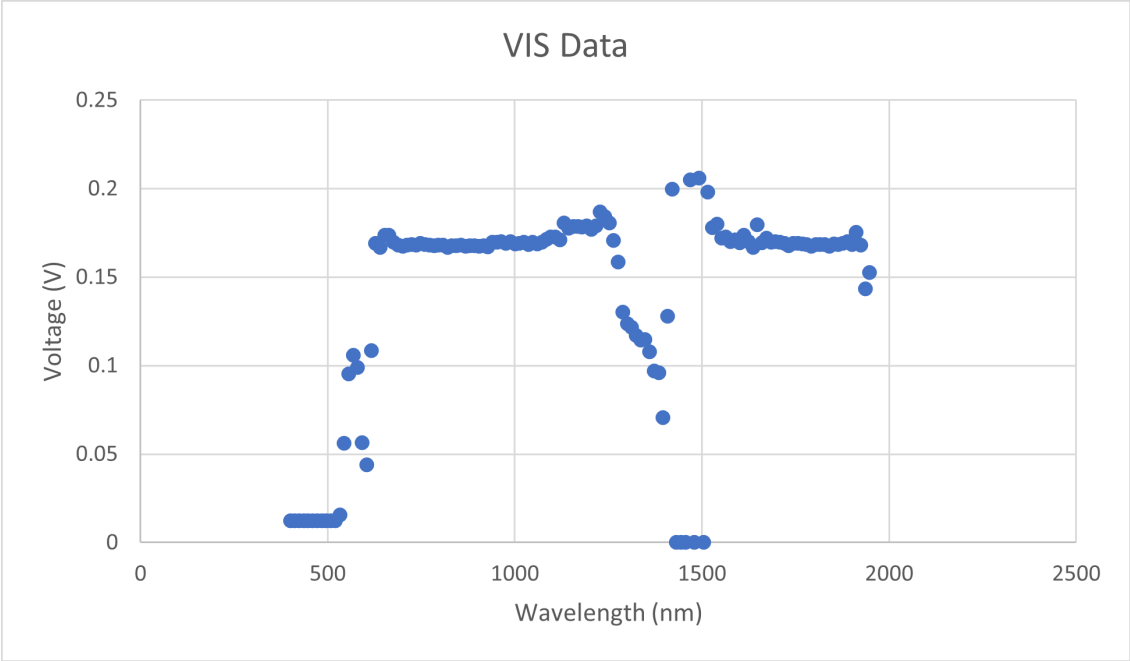
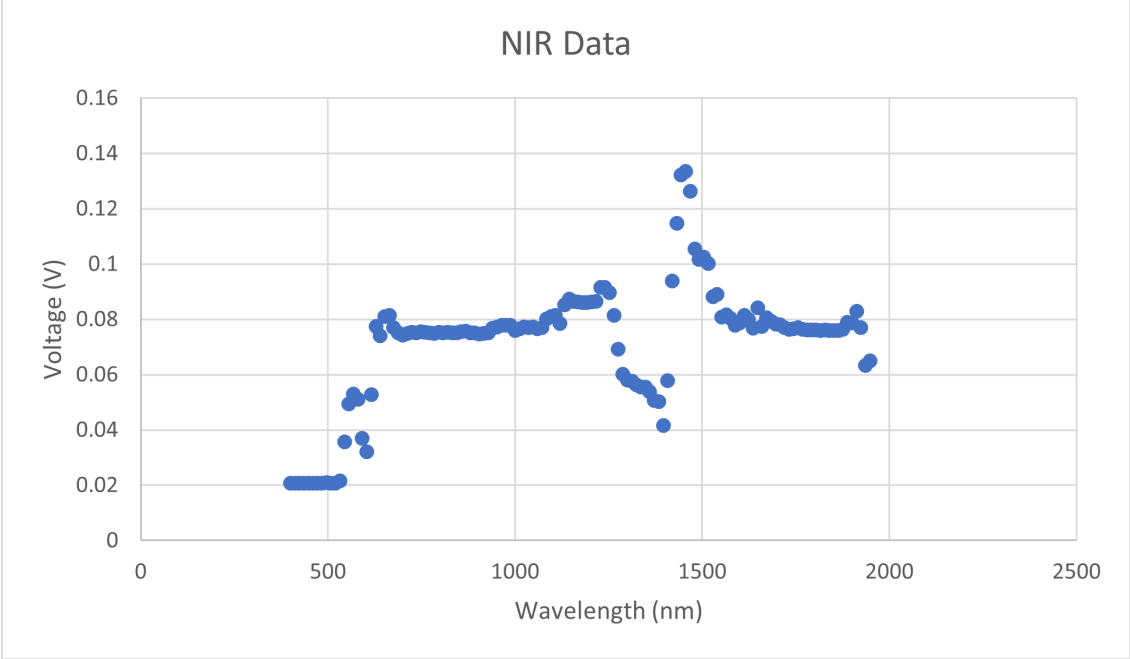


Figure 53: Aluminum Test NIR Data



The Second Test was to take three different soil samples and determine whether the spectrometer data was significant. To conduct this test, two fertilized bags of soil

were purchased from a hardware store and one sample was collected from the ground nearby. The results of these tests are below as well.

Figure 54: Miracle Grow Potting Mix Soil with 0.21 0.11 0.16 fertilizer

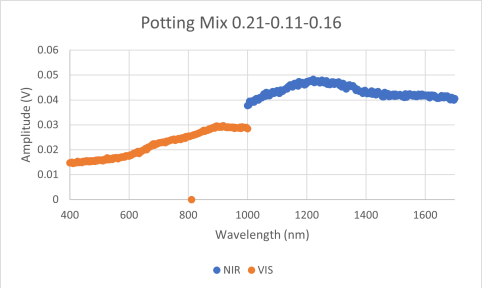


Figure 55: Miracle Grow Performance Composted Manure with 0.19 0.03 0.03 fertilizer

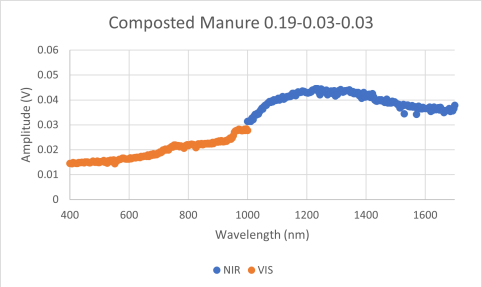
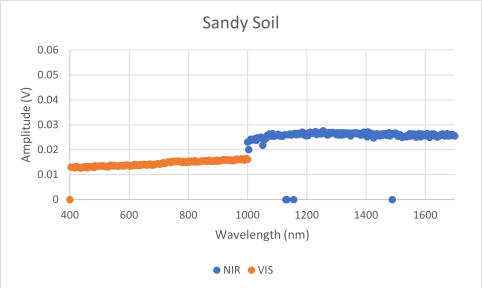


Figure 56: Sandy Soil from University Grounds



These tests proved successful. At this time there is a call to cease activity on the project, and further tests will not be conducted. However, if there were more time to develop the system, the next test would be to engage the full resolution of the device in measurement and begin calibrating it against larger and larger datasets.

6.4 Web Testing

There are three components to test in the web system. The first is the API endpoints and backend functionality. The second is the integration of the user interface and the

API. And lastly, the functionality of the socket server must be tested as well.

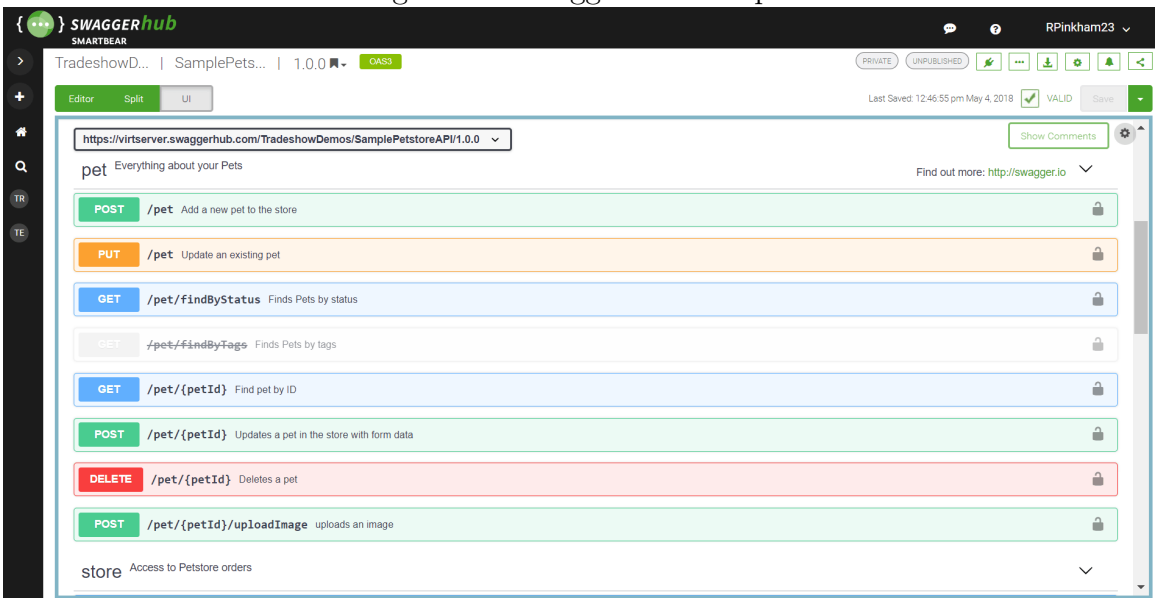
6.4.1 API Testing

There will be two key ways that the API testing will be conducted: unit tests and integration tests. In a web context these are slightly different than in a strict engineering context. Generally speaking, a unit test of a an API or backend service checks that business logic is working as expected. For example, in our project there will be some filter to grab the correct information from the database based on the plant bed. In a unit test, we might mock the database with dummy data and ensure that this is occurring properly.

On the other hand, the integration test of the API will be done in two ways which will also serve to self-document the API. Using a mock client, we can program requests for certain conditions such as certain failure or success. In this instance, the request is actually routed through all the middleware, connects to the database and performs the function. We can validate the result by looking at the database and "asserting" all these values are correct.

The self-documenting test will be done using Swagger and OpenAPI 3.0. Swagger is a tool for building UI elements that can build and run requests on the web server and the response elements can be validated against data types. See Figure 57 for a visual representation as to what this might look like.

Figure 57: Swagger UI example



Test Plan The unit tests and integration tests will be ran on each push of code changes to GitHub and through the use of GitHub actions, passing tests can be validated and added to the branch protection rules. Branch protection rules prevent

broken code from becoming a part of the code base. These rules are integral to our test plan. See Figure 58 to see how passing/failing tests would appear.

Figure 58: GitHub workflow

The key to integrating this quality of life feature is designing good tests. The API in general has four functions: create, read, update, delete. The unit tests will be configured to ensure error handling and data composition are done properly. For example, when making a request about a particular plant bed, if the ID does not exist, we want to the error to be verbose but the request to fail. Similarly, we want to ensure that the logic for building data tables from the database occurs properly. The integration tests will have one well-formed request and one malformed request in order to test the error handling and full integration of correct data.

6.4.2 User Interface Testing

There are two key ways to be able test the user interface: using prototypes and integrating with the backend. Using React the expectation is that the user interface will be reactive. Thus, we use prototypes which are components built on hardcoded data that reacts to user input, thus we can test the viability of components. For example, we want to test that a graph component when hovered shows the data point in a floating box. We would build the UI with static data and ensure this result in the UI. Secondly, we need to test that the integration with the backend is working properly. This occurs by running an "end-to-end" test. Essentially, this is a final test. The plan is to give the testable product to consumers and let them have their way with the UI to discover bugs.

React comes packaged with a testing library that allows for quick unit testing of components in a similar way to the unit tests mentioned in the API section (6.4.1). Implementing this will automate the unit tests as opposed to manually building the entire program and checking manually each component. We will be using this documentation to build out the automatic test suite.

6.4.3 Socket Testing

Unit testing sockets will be done in a similar way to unit testing the API endpoints however the unit test will create a mock socket client and the two features will be tested separately: sending and receiving.

Sending Packets to Client To test that the client is receiving the packets properly in isolation, the mock client will open a connection to the server and the unit test will call the send packet function. The client will then expect certain values in a certain character set and this will all be validated by the assertion.

Receiving Packets from Client The mock client will send a packet to the socket server with static data that is formatted exactly as it would come from the MCU. From there, the unit test will assert that the received packet has all the required metadata, uses the correct character set and is formatted properly.

6.5 Integration Testing

The microntroller is the key linkage between all the other subsystem's. Integration testing between these subsystems will be done in isolation before assembling the entire project. The integrations will be done and tested in the following order before a final full system test is completed.

6.5.1 Web Integration

Due to the use of Docker, the web integration test can be done at all points in development. There are two ways this can be done, downloading and running the image, or building the image from the source code. The processes for each are as follows:

Docker Image

1. Web developer uses docker to build the image
2. Microcontroller developer downloads the built image from docker hub
3. Use Docker Desktop to run image
4. Test

Build Docker Image from Source

1. Web developer pushes changes to Github
2. Microcontroller developer pulls changes from Github

3. Developer builds image from new source code

4. Run the image and test

Testing Areas There are two key areas that need to be test in this integration. First, that the microcontroller and web are communicating appropriately following the standards and design criteria. Second, that the microntroller is able to read and translate commands sent by the server and responds accordingly.

Through the use of UART on the microcontroller and verbose logging on the server application we can validate that the packets that are exchanged between the two systems are received and translated properly. Part of this test will be forcing packets to be sent, so the microcontroller will have commands in UART in order to force send packets to the web server so these can be verified. Similarly, the web component will need a way to force send packets to the microcontroller. These packets can be verified by looking at the metadata associated, the charset, length, and payload.

Once the packets have been verified we need to test that the microcontroller is responding to the packets it is being sent. This can be done in two ways. First, using UART we can print a response to the packet to show that the logic is working. The second part of this testing is ensuring digital logic on the pins to the control mechanisms are also working appropriately.

6.5.2 Sensor Integration

Sensor-controller integration shall be performed in the following manner. The sensing and controlling subsystems shall be connected according to the system schematics. This involves connecting the sensing subsystem power rail to V_{CC}, as well as the sensing subsystem's V_{out} to the MCU's analog-to-digital converter. The MCU shall be connected via backchannel UART to a compatible computer with a serial terminal for monitoring. The sensing subsystem shall be powered nominally and take samples on a timer providing regularly-timed samples for testing. Any raw values obtained shall be printed to the computer's serial terminal. The sensing subsystem shall measure a standardized sample, and the values obtained shall be compared against the known and calculated values of a standardized sample. Any error shall be calculated, and adjustments shall be made.

6.5.3 Power Integration

Power-controller integration shall be performed in the following manner. The sensing and power subsystems shall be connected according to the system schematics. This involves connecting the controller subsystem power rail to V_{CC}, the MCU's SDA and SCL buses to the power subsystem's charge controller IC, and any analog lines in to the MCU's analog-to-digital converter to the power sensing and monitoring circuitry. The MCU shall be connected via backchannel UART to a compatible computer with a

serial terminal. The power subsystem shall be providing power nominally, and testing shall be performed in the following conditions:

- Battery 100% charged, solar panels connected
- Battery 20% charged, solar panels connected
- Battery 100% charged, solar panels disconnected
- Battery 20% charged, solar panels disconnected

Any voltage and current figures obtained shall be printed to the computer's serial terminal for monitoring. Any voltages and current figures measured shall fall within safe expected ranges. Any error shall be calculated, and adjustments shall be made.

6.5.4 Full Integration

After testing each of the individual integrations the final test will be seeing the whole system built and put together. The goal of the previous testing should be that there are issues with the code and the invidual parts. The full integration will consist of the following parts: power, control system tuning, usability.

Power In the previous testing sections the team has not yet covered a "stress" test of the power system. After integrating all the parts of the project, the team will do a 1 week trial run of the power system under an elevated load. This elevated load is more frequent scanning, longer duration of scanning, more transmissions to and from the web interface. This is in the hopes of tuning the power system and proving the viability of our power system. We can differentiate this stress test from lab tests due to the "real world load" that is being applied that could not have been tested until the completion of all the sections.

Control System Tuning The control system will need tuning with the sensor data. Models are only so good. The point of the control system tuning will be to measure soil water content against the sensor's interpretation of the soil moisture content as well as the checking the activation levels of the microcontroller.

Usability The team will give friends and/or family the ability to demo the garden bed in person and from the website. They will have the following rubric to grade the project in order to help fix some of the usability shortcomings:

Table 25: Usability Matrix

Area	Metric	Grade	Notes
Web	Navigable? Intuitive? Appealing?		
In-Person	Durability? Setup? Satisfaction?		

The team felt that the user experience was an essential part of a full integration test. This is a lengthy process but something that will be invaluable for further market research as well as simple improvements to UI. The in-person metrics will help the team evaluate whether they met their design goals or not.

7 Administrative Content

This next section will detail our plans for designing and implementing the project. We will review the methodologies that our team has decided to use and give a brief overview of when we plan to complete different parts of the project. Because this is a living document with our project, you will notice that some items from our Fall section were not completed as originally planned and have been marked as so and also moved to the Spring section. We decided to leave the items in Fall as well as any real project would so that future projects can plan their time more accurately.

7.1 Milestones

Our team has decided to utilize the Agile approach for this project. We chose this methodology because it allows our team to focus on sections of the project and aligns well with the semester schedule. By making iterations for this project internally, we are able to track our progress and make status updates. Another benefit of this will be tracking any delays or problems. If we are behind on a section, we have already planned ahead and allowed ourselves room for some variance. We are also using smaller deliverables for the project which gives us more tracking because we have more internal deadlines. We will be using Jira to track our progress and add reports throughout the process. We will also be utilizing Discord as a form of communication with each other. This will be a place where we can discuss any issues or ask quick questions when we are not in person. Below, we have a high-level breakdown written for our project goals.

7.1.1 Fall

- COMPLETED: Select components for each subsystem

  - Document selection reasoning
  - Order to ensure on-time delivery
- Model physical bed
  - This item was not completed and will be added to Spring
- Build physical bed
  - This item was not completed and will be added to Spring
- PARTIALLY COMPLETED: Understand how subsystems will integrate:
  - Communication protocols (REST, I2C, SPI, DSP, etc)
  - Power requirements
- UI for Web subsystem
  - This item was not completed and will be added to Spring.

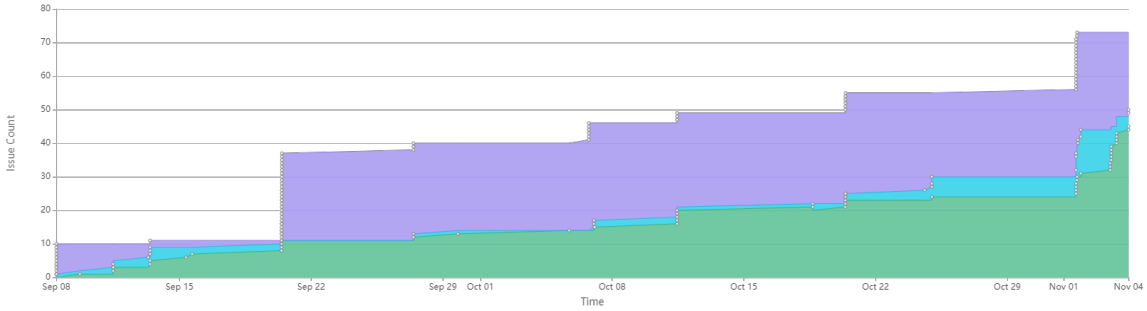
7.1.2 Spring

- Model physical bed
- Build physical bed
- UI for Web subsystems
- Test subsystems in isolation
- Start integrating subsystems
- Control scheme for moving solar panels with sun and to provide shade
- Web API complete
- MCU coding complete
- Stretch goals

7.2 Progress

7.2.1 Senior Design I

Figure 59: Cumulative Flow Diagram from Jira



In Figure 59: purple designates tasks that are marked unfinished in the the backlog and current sprint, blue represents in progress tasks, and green represents finished tasks.

Throughout Senior Design I we have been gathering research and have started laying out the design of our garden bed and have completed the majority of our part selection. The Figure in 59 may be a little misleading at this point because we have not refined our backlog to fully encapsulate meaningful tasks instead breaking it down into larger subsystem requirement-esque tasks.

7.3 Budget

Table 26: Breakdown of budget by subsystem

Subsystem	Estimated Cost	Comment
MCU	$60	The MCU, wiring harness
Power	$200	Solar panels, batteries, control system
Sensing	$200	Components for sensing, optical sensors
Web	$30	Web service pricing
Non-Subsystem	$100	The plant bed, soil, water, fittings, etc
Total	$590	

8 References

[1] R. Flickenger *et al.*, "Wireless networking in the developing world second edition," tech. rep., Hacker Friendly LLC, 2007.

[2] J. Kristoff, "The transmission control protocol.".

[3] J. Postel, "Internet protocol," tech. rep., Sept. 1981.

[4] T. Instruments, *SimpleLink™ Wi-Fi® CC3x20, CC3x3x Network Processor User's Guide.*

[5] T. Instruments, *CC3100/CC3200 SimpleLink™ Wi-Fi® Internet-on-a-Chip User's Guide.*

[6] T. Instruments, *SimpleLink™ Wi-Fi® CC32xx ADC.*

[7] R. Coelho, T. Corson, G. Chirino, and A. Burns, "Smart garden controller," 2018. UCF Self-Sponsored Senior Design Project.

[8] A. Loree, B. Mitchell, D. Ellington, and J. Rodriguez, "Stem 'n' leaf: Modular hydroponics." web, 2021.

[9] C. Hodge, R. Azore, J. Williams, and J. Powell, "Green steel garden." web, 2021.

[10] Docker, "Docker homepage." web.

[11] Docker, *Use Containers to Build, Share, and Run your applications.*

[12] A. Mouazen, M. Maleki, J. D. Baerdemaeker, and H. Ramon, "On-line measurement of some selected soil properties using a VIS–NIR sensor," *Soil and Tillage Research*, vol. 93, pp. 13–27, Mar. 2007.