Team Green

Requirements Specification

April 16th, 2021

Mason Gerace - mag779@nau.edu
Joshua Pollock - jgp77@nau.edu
Alenn Wright - agw73@nau.edu

Table of Contents

1. Introduction

a. Background of your client

The client who proposed the Research Greenhouse project is Adair Patterson. In 2014, she graduated from Northern Arizona University with a Bachelor of Science in Biology. She started working in the Research Greenhouse shortly after graduation and began working with multiple departments and disciplines to assist with their research. Some notable projects that have come from the greenhouses are: working with the NAU Biology department to research plant-fungal ecology, assisting the Restoration Institute in reviving plant populations that have been on the decline, and lastly working with Federal Agencies on researching how to reestablishing native species after wildfires. During the past decade the research greenhouse has expanded its mission to include growing plants for habitat restoration, developing a campus arboretum, partnering with local communities to build public gardens, and plant native landscaping with NAU Facility Services. Besides working with NAU and other Government agencies, the Research Greenhouse also is working on partnering with local communities to build public gardens and educational spaces, developing a campus arboretum, and growing plants for habitat restoration projects. Recently, Ms. Patterson took over as the acting Laboratory and Research Facility Manager and proposed the current project to the Engineering department.

b. The problem being solved

The Research Greenhouses support university-wide research. Over the past 30 years, they have supported projects that have encompassed many disciplines and departments, including Biology, Chemistry, Ecological Monitoring and Assessment, Ecological Restoration Institute, Engineering, Environmental Sciences, and Forestry. Each greenhouse will measure about 1000 square feet, two of them are equipped with CO2 generators, and the facility is well equipped to handle most projects. They also have timer-controlled lighting, high-pressure sodium or metal halide timer control systems, as well as multi-purpose tools and shaded areas inside each greenhouse for plants. The NAU Research Greenhouses come with a level of control that is unparalleled to at-home greenhouses. Overall, the greenhouses are equipped to handle just about everything plant-wise, however, there is not a system in place that allows for convenient monitoring of temperature or humidity inside of the greenhouses.

Each of the eight greenhouses supports several different research projects simultaneously from diverse customers such as students, NAU researchers, and external customers. The greenhouses have control systems installed for regulating temperature and humidity to

prevent them from reaching critical levels. The monitoring system is built into the control system, with a small 1’ x 3’ LED screen that can display the temperature or humidity level. To get the small LED screen, you must open up the circuit box for the control system, which needs a master key. As it stands, the monitoring system is too inconveniently located for real-time monitoring. As seen in Figure 4, the temperature control box is a large system that maintains the balance of temperature and humidity inside of the greenhouses. Customers typically need a record of the temperature and humidity for their projects, but it cannot be provided without being at the greenhouse in person and making a record of past readings. There have been past projects that also need a record of the light level, however, there was not a system in place that could do so. Future projects will also use high humidity enclosures inside the greenhouse and there is not a monitoring system that can be placed inside of an enclosure to measure the humidity.

2. Design Process

a. Overall design process description

To act as the main hub to maintain our data, our team utilized a Raspberry Pi 3B. The Raspberry Pi will act as the server for the ESP8266 microcontrollers, allowing data to be sent from the sensors to the server for collection. The interface between the ESP microcontrollers and the Raspberry Pi will be a Mosquitto server. The Raspberry Pi acts as the broker, handling each request sent to it from the ESP modules. The language Python is used to host this Mosquitto server as well as manipulate the sensor data payload into the correct CSV database file. As well as hosting the Mosquitto server, the Raspberry Pi will post each CSV database file to the website using a simple PHP submission form located on our website. This is done using Python and is executed following the writing of new data to each database CSV file. Another Python script is utilized to perform a data wipe at a specific set time every week to avoid performance and data loss. Lastly, a cron job was created to execute each of these scripts at specific times. Cron jobs are a built-in function of Linux to allow for the execution of scripts at specified frequencies and times. With this set up this way, the Raspberry Pi can be power cycled and will automatically launch all necessary scripts on boot.

The main component of our sensor module is the ESP8266 NodeMcu D1 mini which is a microcontroller that includes a Wi-Fi modem. This board will handle mitigation of the sensor processing, as well as the transmission of the data gathered by sensors. The ESP can operate within a temperature range of -40 ℃ to 125 ℃, with a power input of 4.5 to 9 volts at 200 microamps to 200 milliamps of current. The memory of the device is 4

megabytes compared to the Arduino Uno's 32-kilobyte capacity will assure us plenty of capacity for complex code.

The BME280 sensor is capable of reading humidity, temperature, and pressure. The sensor operates at temperatures from -40 ℃ to 85 ℃ at a voltage of 1.2 to 3.6 volts with a max current draw of 3.6 microamps. The BME sensor has a low response time of 1 second or 1 Hertz and is relatively accurate at about 3 percent error. The sensor also has a considerable lifetime with less than 2 percent hysteresis, being useful for up to 10 years. Since the operating voltage is within the range of the ESP8266 power supply voltage the BME280 became a perfect candidate for sensing humidity and temperature.

For sensing light, the initial sensor that was chosen was the TEMT6000, which is a phototransistor sensor capable of reading and the light intensity in lux. The TEMP6000 is a voltage divider, meaning it needs to have a constant power supply. This complicated the project since the ESP goes into deep sleep mode to help with battery efficiency. A switch was made to include the BH1750, a 16-bit ambient light sensor. The BH1750 can measure from 0-65,000 lux, comes with an integrated voltage regulator, and a level shifting circuit that allows the sensor to be used with a 3.3 volt or 5-volt input.

To regulate current from the solar panel to the battery, a TP4056 charge protection circuit was implemented for lithium-ion and lithium polymer batteries. This circuit protects against overcurrent, overvoltage, undercurrent, under-voltage, and short circuit. The TP4056 circuit allows for a 5-volt solar panel to be hooked up to the IN+ and IN- ports on the board. This is used to charge the battery safely and efficiently. Due to the form factor of this device, the features provided, and the efficiency of the TP4056, it is a fantastic choice for charging and protecting the battery.

A 150 milliamp solar cell is used to charge our lithium-ion. Due to the size of the system as a whole, this solar cell will need to be small while providing enough power to trickle charge the battery. With the TP4056 module, our choice was limited to using a 5-volt solar cell. This is because the TP4056 requires a 5-volt input to charge on the IN+ and IN- terminals. Our team looked at many different solar cells at 5 volts and found a good physical size for the cell. The solar cells that we found provide 150 milliamps of current at 5 volts. With the daily power draw, the battery will be able to power the module for up to 14 days, and the 150 milliamps solar cell can fully recharge the battery within a full day of sunlight.

When it comes to batteries, there are two options, either lithium-polymer or lithium-ion cells. Lithium-ion cells are similar to lithium polymer batteries, but instead of a soft exterior, the lithium-ion cell has a hard outer shell similar to a AA battery. Lithium-ion

cells are slightly more expensive than lithium polymer cells. The tradeoff to being more expensive is that they are much more durable. The most common lithium-ion cell is a 18650 cell. These cells offer a typical capacity of 2850 milliamp hours. The 18650 cells have a nominal voltage of 4.2 volts and a minimum voltage of 3.7 volts. While a battery charging circuit should be used to charge the batteries, the cells have a discharge cutoff at 2.5 volts. The 18650 lithium-ion cell is the best choice for a rechargeable battery with decent durability and a small form factor.

b. Functional decomposition

As shown in Figure 1, our system architecture depicts every component used in our project. We broke the system into 3 major sections: a power system, the sensor system, and a wireless communication system. The power section consists of a 150 milliamp solar cell, a TP4056 charge protection circuit, and a 18650 lithium-ion battery rated for 3000 milliamp hours. The solar panel is soldered directly into the charge circuit, then the charge circuit will charge and discharge the generated power from the solar panel depending on the state of charge of the battery. As described in the design process, the charge circuit is then soldered to the battery mount. Since the battery can be used for up to 14 days without needing to be recharged, the solar panel will only be recharging the amount of power consumed overnight.

The sensor section consists of an ESP8266 microcontroller, a BME280 temperature and humidity sensor, and a BH1750 light sensor. The BH1750 can be used on either 3.3 volts or 5 volts as the input voltage, this allowed us to solder the BME280 to the 3.3-volt output of the ESP8266 and the BH1750 to the 5-volt output. The individual sensors are then wired to a data line input from the ESP8266. After uploading the code to the microcontroller, we connected the reset pin to an input. By connecting these ports, it allows the ESP8266 to go into deep sleep mode, disabling all unnecessary functions except the clock. This plays a major role in the battery life of the system and is why the modules can be powered off the lithium battery for upwards of 14 days.

The wireless communication system is the Raspberry Pi. The interface for the sensor modules and the Raspberry Pi is the Mosquitto server. The Raspberry Pi acts as a server for the sensor modules, which allows data to be sent from the modules to the server for collection and management. The Raspberry Pi will then use python to sort the data into readable measurements. After storing the data as CSV files, we will then send them to our website for real-time use. For a more detailed explanation, refer to section 2.a for the overall design process.

c. Prototype findings: results or effect

For our team's prototyping research, we attempted to divide the project into three parts to give one of each to members of the team. We decided that this split would divide the project into the following categories: power delivery, wireless communication, and collecting sensor data. Since Alenn had access to a 3D printer, he volunteered to handle the casings and printing. Josh already had a solar cell and lithium-ion 18650 battery, so he volunteered to take power delivery. Lastly, Mason was given the task of collecting sensor data. For Josh's prototyping, he would oversee figuring out the low power operation of the ESP8266 microcontroller and collecting some sensor data using the low power mode. Working with the low power mode and reading sensor data, he would still be able to focus on power delivery by reducing the current used by the ESP8266 microcontroller. Mason ended up using the ESP8266 to connect to a DHT11 sensor and attempted to get the sensor readings posted on to a webpage using PHP and C, falling under our main function of collecting sensor data. Alenn worked on solving how we were going to waterproof our circuit, testing different casings, plastics, and housings.

We were also able to collect sensor data, by connecting a DHT11 sensor to the ESP8266. We would have liked to go further and post the sensor data over to the local host webpage, but we were not able to figure out utilizing PHP code to make this work. Since our prototype parts did not make it on time, we had limited supplies, just ESP modules and limited hardware that our group has individually gathered over the years. The DHT11 is not the most accurate sensor, that's why we have chosen the BME280 as our final temperature and humidity sensor. The BME280 is a good prototyping sensor since it was simple to connect and figure out how to change certain parts of the code to accommodate the BME sensor. By utilizing the language of C, it was simple to upload code to the ESP via the Arduino IDE.

When prototyping the circuit protection, the main focus was water protection of the circuit. The high humidity of the greenhouse can be prone to short-circuiting electronic circuits, so we need to take precautions. Naturally, our project will have housing, so we designed it to have a mount for the solar panel and a light sensor. Then the other exposed sensors will be from a square hole through the side of the casing. The functionality of having the tunnel is to reduce the sun damage on the parts as well as not to expose them to unnecessary condensation that may build up. The square hole tunnel will also help to minimize the chance of water ingress into the module.

The first step in prototyping the housing was to select the materials that we were going to use to construct the casings. It did not make sense to go with metal due to the project

wanting to be low cost, it made more sense to go with plastic since we already have access to 3D printers, and we have more leeway for designing the casing. Now choosing our plastic may seem pretty simple but there is some testing that went into it. Considering the expense wood, metal, and carbon fiber filament was too irrationally expensive. When considering that we want the project to last, PLA (Polylactic acid), PVA (Polyvinyl alcohol), and HDPE (High-density polyethylene) are eliminated due to their biodegradable and dissolving aspects being undesirable in the final product. So, then we needed to test ABS (Acrylonitrile butadiene styrene), Nylon, and PETT (Polyethylene terephthalate) filament that we had available to us beforehand.

To test the plastics, the best way we found was to print sample 3D printer calibration cubes and subject them to the sun and water-abundant environments. To test water absorption, we placed the cubes in water and then after a few hours put them on a plate after dabbing them with a towel to see if water seeps out. At the end of this testing, the nylon filament absorbed and took on the most water. This result was disappointing as its durability is one of its main selling points but did not think that the warping of the filament would be so extreme. PETT was the second-worst as it did not warp in the printing process but rather started to expand in water, this is a little concerning as we do not want the casing to expand. Finally, ABS still took on water but it took on less than half of what the other printers soaked up, additionally, ABS can be easily reformed and sealed with acetone which is capable of dissolving the plastic but in small doses, it can melt which results in waterproofing the 3D prints.

The biggest challenges we faced were understanding the PHP, HTML, Arduino IDE, as well as finding the correct melting point for the plastics. Getting the Arduino IDE and PHP to work together was extremely challenging since we didn't have much experience with it or have the correct parts we wanted to use to prototype. The ESP8266 offers its own development IDE, but it has limited features and support. The Arduino IDE was the best option for finding libraries and support for the ESP8266. To allow the ESP8266 to interface with the Arduino IDE, we needed to add the ESP8266 board to the additional board manager in the Arduino IDE. Once this board library was installed, we needed to select the ESP8266 for the 'Node MCU 1.0' board. This is the board that was compatible with the ESP8266s that we had. If we had the chance to do this prototyping step again, we would order our parts 2 more weeks in advance so we would have around 2 weeks in total of testing all of our sensors, power system, and solar panel. This would allow us to use the sensors that we planned on using instead of random sensors that we had lying around. This prototyping was still very successful, and our team will be able to use the research we found with the sleep modes of the ESP8266, the interconnectivity of the sensors, as well as wireless communications.

Overall, the demonstrations of our prototypes were successful. We found success in getting the ESP8266 to operate in its low power mode called deep sleep, collecting sensor data, and testing different types of plastic. While in deep sleep, the ESP8266 only consumes about 20 microamps and disables everything but the real-time clock (RTC). This means it disables the wireless modem, the system clock, and the CPU. With a 18650-lithium ion cell, we will have around 3000 milliamp hours of energy to power the system. In deep sleep mode, this would allow us to keep the system operational for 175,000 hours. This is an impressive amount of time and should allow us to power the system entirely on a 18650-lithium ion cell and a solar cell. We then used a photoresistor to calculate the lux value and a DHT11 to measure the temperature and humidity. The DHT11 uses 0.3 milliamps while measuring and 60 microamps during standby. This is a surprisingly low amount of power and should barely impact the overall system operation time. The sensor mainly operates in a standby mode and the measuring mode only takes a few milliseconds to complete. The DHT11 sensor is not our final choice in the sensor as it is inaccurate, but it is a great prototyping option as it is easily accessible. The photoresistor used a large amount of power however and our group wants to find different options for sensing lux. A photoresistor is a voltage divider and therefore at least uses 3.3 milliamps constantly and can use an even greater amount of current depending on the light level. There is no standby mode for the photoresistor and therefore we eliminated it from our final design.

3. Final Design

a. System architecture with supporting details such as behavior, flowcharts, schematics, diagrams, etc.

As shown in Figure 1, the System Architecture components that are shaded in a blue tint are the parts that make up the sensor modules. The power system is comprised of the solar panel, the TP4056 charge circuit, and an 18650 lithium-ion battery. The power system is then connected to the sensor system and this makes up the sensor module. To get the longest battery life possible out of the 18650 battery, we enable the microcontroller to go into deep sleep mode. Deep sleep mode disables all unnecessary clocks and wireless modems on the ESP, reducing current consumption about 20 microamps. The ESP will automatically pull the D0 output LOW when deep sleep is over and when connected to the RESET pin, the ESP will wake up from sleep. When the ESP wakes from sleep mode, the data collection process follows these steps: connect to Wi-Fi, connect to the MQTT broker, read measurement data from the BME280 and BH1750, and pass data into the MQTT database for processing. The Raspberry Pi will then execute two Python scripts: one to sort the data and store it in CSV files, the other to upload the data to the website using PHP. The website data that is stored on the Raspberry Pi is deleted and overwritten with a blank template at the start of every week to conserve space and improve website performance. The Raspberry Pi will also detect if a USB is inserted and if so, it will copy over the CSV files for the 10 modules creating a new folder with the corresponding month and day it was copied in.

b. Text explanations of the major components of your system

The system architecture shows the charging system consisting of a 150 milliamp solar panel connected to a TP4056 charge controller and protection circuit. This charges the 18650 lithium-ion battery that powers the ESP8266 microcontroller. The battery, charge controller, and solar panel make up the power systems for each of the modules. The BME280 handles measuring temperature, pressure, and humidity. The BH1750 handles measuring light levels in lux. When these are connected to an ESP microcontroller, we can wirelessly send the data from the ESP to the server. Connecting the ESP to the power system will make up the entirety of each of the sensor modules. The modules then connect to the server that is hosted by the Raspberry Pi, sorted, stored, then pushed to the website.

4. Results

a. The requirements spreadsheet with the color indication of requirements testing results

Type of Test	Status	Req. #	Requirement
		1	**Humidity Sensor**
Inspect		1.1	Low hysteresis, ideally accurate for up to 5 years.
UTM		1.2	Humidity sensor accuracy within +-2.5%
Inspect		1.3	Low power usage when operating, 10-100 µA
		2	**Temperature Sensor**
Inspect		2.1	Operable at temperatures between 40 and 90 degrees Fahrenheit.
UTM		2.2	An accuracy of +-1.5 degrees Fahrenheit.
Inspect		2.3	Low power usage, when operating it should be between 10-100 µA.
		3	**Light Sensor**
Inspect		3.1	The light sensor will measure illuminance in lux.
Inspect		3.2	Low power usage, when reading light usage should be in the range of 100-200µA
UTM		3.3	Accuracy within +-2.5%
		4	**System**
Inspect	*	4.1	The system will need to be water-resistant to withstand moisture and mist (IP56 rating, Ingress Protection).
UTS		4.2	The sensors will need to be able to be picked up and moved to different locations.
Integration		4.2.1	The sensors will need to be able to be picked up and moved to different locations.
Integration	*	4.3	The system will have remote access to data.
Integration		4.3.1	Able to access temperature, humidity, and light readings from the website.

Inspect		4.4	Withstand temperature between 40 degrees and 90 degrees Fahrenheit.
Integration		4.5	Notification to the user if the temperature is outside the designated “safe” range, depending on each greenhouse the ranges are 70-85 degrees Fahrenheit.
Integration		4.6	The system should operate with minimal user interaction.
Inspect	*	4.7	Each unit must be self-sustaining and not require outside power.
Inspect		4.7.1	The unit must be able to charge and power itself to reduce downtime and interaction with the unit.
		5	**Data collection will ideally happen every 5-15 minutes.**
Integration	*	5.1	Data will be categorized by each greenhouse.
Integration		5.2	A daily collection of sensor data will be sent and stored on the Raspberry Pi
Integration		5.2.1	If a USB is inserted, the Raspberry Pi will save the readings onto the storage device
Integration		5.2.2	Data readings will be stored in a CSV file on a Raspberry Pi
Integration	*	5.3	Must collect data at least every hour.
Integration		5.3.1	Data collection will ideally happen every 5-15 minutes.
Integration		5.4	Must be capable of sending the data to a website for mobile use.
Integration		5.4.1	The website should be able to view sensor data from all modules within the 8 greenhouses.

b. Important test results

The requirements that are crucial to the success of the project are indicated by the Asterisk (*) in the requirements table. The requirements that were deemed the most crucial to the project were: waterproofing and achieving an ingress rating of 56, having remote access to the sensor data, the unit's ability to sustain itself and power itself, categorization of the data, as well as having multiple data readings every hour. Having at least 1 sensor reading an hour was one of the few hard requirements that the client of the project gave us, as well as the ability to remotely access the data. Ideally, the client wanted to have 2-4 readings an hour to better track the conditions of each greenhouse. To fulfill the remote access requirement, we used a Mosquito database to push the sensor readings to a website for real-time use. An ingress rating is something that is used to define the levels of sealing effectiveness of enclosures against foreign substances, such as water or dirt. The rating of 56 refers to two things, the level it can protect against dirt, and the level it can protect against water. The 5 of the 56 means that the system is dust protected, limited ingress is permitted and it will not affect the electrical system inside the enclosure. The 6 sides of the 56 determine the ingress protection against water, with the 6 levels of protection meaning the enclosure can resist water, but will not be able to survive being submerged. Since each of the modules will be operating inside of a greenhouse, achieving this stage of water and dirt proofing was required to be able to withstand daily use. Organizing data was one of the hardest requirements out of the project. Every day, about 100 readings are being taken, out of 10 modules, it totals around 1,000 data points per day. To present this data and graph it, organization by module was needed. It was a quick fix within our C code for the ESP modules, applying a unique identifier to each. By doing that, we can organize the units by number, with units 1-8 being placed in each of the 8 greenhouses and units 9 and 10 being placed wherever extra measurements are needed. With this organization method, we can keep track of the time, date, and module that makes a sensor reading. The main idea behind this project is to make each unit have as little maintenance as possible. To aid this, we wanted to make each unit able to sustain itself. Each module has a 150 milliamp solar panel that directly charges the battery. In the case that a module runs out of battery, the solar panel can provide charge to the module and fully charge the battery within 12 hours of being in the sun. This allows the modules to be placed in the sun to charge the battery instead of the client taking apart each module and charging the battery manually. In all, the important parts of our requirements are meant to keep the system running while being able to withstand the harsh electrical environment that is the Research Greenhouse.

c. Analysis of Results

When it came to testing all the individual requirements to make sure that each part would fulfill what we needed it to, we started doing this as soon as we were finished prototyping. After prototyping at the end of the Fall semester, we got together and ordered a small batch of parts so we could start testing them. This included the BME280, BH1750, as well as several ESP8266 D1 mini microcontroller modules. This allowed us to start our inspection and unit tests of the sensors. Overall, we needed both sensors to allow us to enter deep sleep mode from the ESP, have low hysteresis so they would be accurate for years to come, as well as having a high reading accuracy and low power consumption. With these requirements in mind, the BME280 and BH1750 passed all of the tests and they were kept throughout the rest of the spring semester and implemented into the final product.

The next set of requirements was regarding the system and casing. Since each module will live its life inside of a greenhouse, we needed to protect the circuit and waterproof the casing to allow the modules to be able to be placed anywhere in the greenhouse without having to worry about them being ruined. The important requirements from this section are mentioned in the previous paragraph, but several inspections, integration, and unit tests were still needed for the system. We needed the sensor modules to be able to be moved freely, while also having the ability to measure enclosures that will be inside of the greenhouse. The system also needed to be able to withstand temperatures within 40-90 degrees Fahrenheit, and if it is not within those rangers the system needs to notify the client that it’s reached critical conditions. By utilizing a Raspberry Pi with the ESP microcontrollers, the ESP’s can connect to a Mosquito broker if it's connected to Wi-Fi. By including a battery that gets charged by a solar panel, the modules can be placed freely throughout the greenhouses. Since the module communicates wirelessly and is self-sustaining, the modules require little to no maintenance and can be placed in sunlight to start providing power to themselves.

The last section of requirements was the integration testing of the data collection from the ESPs to the Website. The daily collection of sensor data will be kept on the Raspberry Pi for a week until its removed, and if a USB connection is detected, the readings will be stored onto that USB device. Both of these processes are done using python scripts. The data readings needed to be stored in a file, so we decided the Comma Separated Value files would work best for the type of data we are measuring. Data collection was touched on in the section before this, and the collected data would then need to be pushed to a website for real-time access. For the data to be pushed to the website, the ESP’s are connected wireless to a Mosquito server via Wi-Fi. If the ESP wakes from deep sleep to

get readings, the readings will be sent to the server to be saved. Our website then utilizes PHP to push the data from the database to the website, which then allows a user to view the data from any browser-connected device.

5. Conclusion of Capstone Report

a. Most important requirements and their results

The most important requirement of this project given to us by our client was to produce waterproof sensor circuits that collect data on Light, Humidity, and Temperature. This collected data was to be made available for our client to view from outside the Research Greenhouse Complex. The result of our project is that we produced a product that is wireless and solar charging that can collect the required data from the greenhouse accurately and precisely. The modules then Utilize Wi-Fi to communicate to a computer to send data to our capstone website to be viewed online visually and can access the data files when necessary. For a more detailed explanation of the results and requirements, refer to 3.a-3.c.

b. Lessons Learned

Throughout the project, the team ran into countless problems along the way. The three problems that delayed our project the most were our NAU Wi-Fi problem, shipping times, the range of Wi-Fi within the greenhouse complex, and having a three person team. Early in the first semester, we realized quickly that the ESP8266's were not able to connect to the NAU Wi-Fi and this was a major problem. Our early design was having the ESP's connect to Wi-Fi, then send the sensor measurements to the database. Since this design wouldn't work, the workaround we thought of including was the Raspberry Pi. Since we could have the modules connect directly to the Raspberry Pi, this eliminated the Wi-Fi problem. Another problem that caused a 3-week delay in the overall project was the shipping of the Raspberry Pi, NAU ITS flagged the shipment and it had to get approval from an ITS manager. Since ITS does not work very fast, it delayed the shipping by 2 weeks, then it shipped to the Dean's Office which caused another week-long delay. The last major problem we ran into was the range of the Wi-Fi inside the greenhouse complex. The range, if we are broadcasting from the center of the greenhouse, is just barely able to reach the furthest two greenhouses. To remedy this, we attached the antenna to the ESP8266's, this allowed us to increase the range by 1.2 to 1.5 times the original distance. In the first semester iteration of this course, we had a team member drop the course. This resulted in our four person team being changed to a three person team. When this occurred, the professor gave us permission to under-deliver, allowing at least 5 modules for full credit. However, after getting to know our client, we knew that 5 modules would not be sufficient. The team then produced 10 modules instead. The

project itself was not very complicated, when it came to the amount of software we used, we concluded that if we were not a full team of computer engineers there would have been a much higher possibility of not finishing. In conclusion, this project taught us that Murphy's Law is true, anything that can go wrong will go wrong. Other than that, we have learned many valuable lessons regarding time management, collaboration with peers, as well as developing refined project designs. When we graduate and go into the industry, we will take these lessons learned and apply them to future projects.

6. User Manual

a. Introduction

From Team Green,
We hope you enjoy the product we developed for you, we are pleased that you have chosen us for your Research Greenhouse capstone project. The project description stated how much of a need there was for reliable sensor measurements that would be accessible from places outside the greenhouse. Our wish is that the old, headache of a solution that you had before this project is something that you will never have to deal with again. If the system we developed ends up being used for the next few years, nothing would make us happier. The 3 of us provided for a powerful self-sustaining system for collecting temperature, humidity, pressure, and light levels in the greenhouses. We wish this custom designed project fulfills everything you could have wished for.

Key highlights of the project include:

- Mobile, self sustaining sensor modules
- Extremely low maintenance/repair needs
- Live data readings every 15 minutes
- Data going from sensor module to website in under a second

The purpose of this user manual is to help guide you through any possible problem that could arise. This manual will be able to help someone who is or is not well versed in electrical components. On the software side, the team has the ability to SSH into the main hub of the project to maintain the remote measurement system. We aim to make sure that you can view sensor data in an easy-to-access format. Additionally, we want you to have access to accurate and precise data for your projects. We hope our product will continue to assist you for many years to come.

The research greenhouse consists of eight different greenhouses that are home to projects from students, NAU researchers, as well as other external clients. Currently, the greenhouse has an outdated control system, with no way to view the temperature, humidity, and other sensor data in real-time or keep track of past readings. This puts clients at a disadvantage because they often need to be able to keep track of temperature and humidity for their projects.

As of now, the greenhouses have a control unit that keeps the temperature in a range of 40-80 degrees Fahrenheit, keeping them from reaching critical conditions. Our client has purchased a Bluetooth temperature and humidity sensor so she could keep an eye on the greenhouse via a mobile device. Due to the limited range of Bluetooth, she is unable to

see the data in real-time or see past measurements. The research greenhouses are home to projects from students, NAU researchers, as well as other external customers. The problem proposed by our client is not a niche issue as many Wi-Fi-based sensor modules already exist for greenhouse applications. Greenhouses require constant and precise monitoring. Without this monitoring, the plants in the greenhouse can be exposed to conditions that can potentially kill them. Slight changes in temperature, humidity, or even light can affect the growth of the plants.

The project had three subsystems and that was the wireless communication, the sensor measurements, and the circuit protection. The reason we went with wireless communication was that to get the sensor data to be accessed online Wi-Fi would be a natural component of our design and then we found an Arduino based Wi-Fi module called the ESP8266 microcontroller. We had figured that the chipsets for multiple modules might be cheaper and less labor-intensive than wiring the modules in the greenhouses. Due to us needing many different modules, to make the project simple for the client we made them into mini solar-powered sensor boxes. The mobility of the sensor boxes also fits a client's desire to move them closer to certain projects in the Greenhouse. For the light-sensing needs of the modules, we chose the BH1750 for its precision, communication protocol, and ability to be turned off when the data doesn't need to be updated. This is a similar case with choosing the BME280 as it would give us a high accuracy of data and supply us with the required humidity and temperature readings. Due to the BH1750 and the BME280 having I2C communication and different sensor addresses, we could wire these to be connected to the same wires reducing the wiring complexity to 2 wires for the sensor data for later possible maintenance.

Later on during testing, we found it impossible to connect to the NAU Wi-Fi with the ESP8266 chips so we came up with a different solution that has not been done to our knowledge before. We had gotten a Raspberry Pi which is an ARM-based microcontroller to act as an access point for the ESP chips to connect to an MQTT server which filters the data and sends it back to the Raspberry Pi and through a series of Python and PHP scripts the data is compiled in CSV files and uploaded to the Capstone Website through a PHP upload form.

For protecting the circuit, we decided to coat the circuit in a layer of nail polish because the prior experience it does not wear away fast if not constantly rubbed which these modules will most likely see very minimal human contact so it's a perfect choice. Additionally, we designed a case to put the circuit in to properly protect it. Making proper inlets to place the solar panel on, as well as holes for the sensors so they could collect accurate measurements. For the temperature and humidity sensor, we grew concerned that it may be afflicted by direct sunlight. We designed a tunnel in the case which the

BME280 sensor only has access to, to measure ambient temperature. As it is common with 3D prints, they tend to have gaps and splits called layers. These layers can however be sealed using acetone by rubbing it onto the ABS casing.

b. Installation

The Raspberry Pi home station should require minimal interaction to install and get it working properly. All scripts on the Raspberry Pi are set up to run on startup or at specified times throughout the day. If there is ever an issue with powering on the Raspberry Pi and having data post to the website, see troubleshooting steps below. Otherwise, all that is required to set up the Raspberry Pi is to provide 5 volts through micro usb to the Raspberry Pi. If for some reason the supplied power supply fails, replace it with a high quality power supply. The Raspberry Pi is quite sensitive to cheaper power supplies and will not function properly if the power supply does not put out a steady 5 volts. A power supply of at least 2 amps should be used to provide the Raspberry Pi with enough current to function correctly.

Setting up the Raspberry Pi was the majority of our issues getting this project completed. This was due to the ESP modules not being able to correct directly to NAU's Wi-Fi. NAU uses a WPA2 Enterprise security with a PEAP and radius server for authentication on their network. This meant that each MAC address of the ESP modules would have to be authenticated through NAU along with acquiring a security certificate for each of the ESP modules from NAU ITS. This is why our team decided to use a Raspberry Pi instead of directly posting the data from the ESP modules.

The Raspberry Pi is set up as a one way router for each of the ESP modules to connect to. This is secured in a similar way to NAU's security, but allows us to have more control over each device. We have the Pi broadcasting its own network that will only allow 10 predetermined MAC addresses to connect to it. The network the Pi is broadcasting is also hidden from view to prevent prying eyes from trying to connect to it. We felt this was the most secure way to have a network setup while not utilizing NAU's own Wi-Fi. The network is completely sandboxed from NAU's network and only allows LAN connections to the Raspberry Pi. The Raspberry Pi itself is connected to NAU's network via ethernet cable but does not pass the connection through to the ESP modules. This would still allow us to pass the sensor data to our website, while being on a device that is approved to connect to NAU's network.

Once we had the Pi's network setup, we were able to start having data sent to the Raspberry Pi from the ESP's through the utilization of a Mosquitto server. The Pi is acting as the Mosquitto broker, processing each payload sent to it and saving it to a CSV

file. This was done using the Paho MQTT package in tandem with Python. We used Python in this case as it provides a lightweight and well documented scripting language that has plenty of libraries for interfacing with the ESPs and data sent to the MQTT broker. To enhance the range of the ESP modules, we added small 3 centimeter wires to the antenna of the ESP8266 chip in accordance with some suggestions from online forums. This allowed us to place modules in each greenhouse without the need for network repeaters.

The modules that we made were designed to be very low maintenance and require minimal contact. To test if a module is in range, open the module and press the black reset button on the side of the ESP. After 20 seconds, check the online graph to see if the data was uploaded to the website. If it did it is in range of the Raspberry Pi access point and you can close the device.

c. Configuration and Use

The modules will be able to run autonomously if placed within range of the Raspberry Pi access point. The modules are designed to have minimal interaction by the users. The main use of the modules is for sensor data to be uploaded to our capstone website and graphed for an easy view of the data. It is optional however for a USB that has been named “DATA” in all capital letters and the Raspberry Pi will dump the module CSV files onto the USB thumb drive. While the USB is plugged in the data will be stored into a new folder named based on the month and day. To prevent excessive data build up the Raspberry Pi deletes the data on Sundays at midnight. Also see our GitHub repository for all software relating to this project which will be linked in the conclusion with our emails.

d. Maintenance

The maintenance for this project was made very simple by reducing the amount of unnecessary fiddling of the modules. However, the waterproofing of the circuit in the environment of the greenhouse will need to be reapplied every 3-5 years to prevent breakdown. A new coat of nail polish should be applied to each device and over the BH1750 light sensor. If the ABS casing is cracked or splitting, acetone can be used to help to melt the ABS plastic over the cracks. ABS glue is also available in the PVC or ABS isles of a hardware store and should be used for larger cracks. The difference in this glue is that there are already dissolved ABS particles in the glue that fill in the gaps. Nail polish can also be used as a temporary fix to cracks in the casement but should be repaired eventually using the correct methods. These repair methods should ensure that there is a waterproof coating on each of the circuit components and that the ABS casing is sealed from condensation and water.

e. Troubleshooting Operation

In the unfortunate event that a module starts to malfunction there are a few small things that you can do to ensure there is nothing very technical that you will have to do to the circuit. One is to pull the module closer to the Raspberry Pi hub in case of a short range of the individual module. Another thing you can do is by using a micro USB cable to plug into the charge circuit to charge it up a bit in a rare case that the battery discharged. A multimeter can be used to read the voltage of the 18650 cell and it should be within the range of 3.7 to 4.2 volts. If it is under this, the battery needs to be charged. If these steps do not work, you may have to install the Arduino IDE and have some ability to use a soldering iron on electrical components.

For troubleshooting with the Arduino IDE, the first step is to unplug the charge circuit from the ESP8266 D1 board and use a micro USB cable into your computer with the IDE and you will first see if it can be recognized by the Arduino IDE. To find if it is recognized you will navigate to Tools > Port, if Port appears grey there is a small chance something has been shorted on the ESP module. If it is visible please skip this next section but recommend reading it so you can have a better understanding of troubleshooting modules in the future.

For this section, you will need to heat a soldering iron enough to melt the solder. Before soldering, please ensure that the ESP is not supplied power from a computer or the 18650 battery. Failure to do so may result in the ESP modules being shorted. Once unplugged you will need to navigate to a pin labeled G, this should also be marked by a black wire, this is ground, and unsoldering it from the ESP will help troubleshoot the sensors or the ESP is the problem. Once the black wire is unsoldered put the iron off to the side and try plugging the ESP into the computer with the IDE and try searching if it is then recognized by the port. If it appears now one of the sensors is shorted/fried and will most likely need to be replaced. Likewise, if the ESP does not appear on the Ports then it will most likely need to be replaced. Replacements and reinstallation will be covered below.

In the case of the ESP being recognized please make sure that the solder that is connected to the pins on the opposite side of the USB port of the board this makes sure the ESP can sleep and save energy and if not jumped then the ESP will never go to sleep and then it will drain the battery and never post more data to the Raspberry Pi. If the solder connect is there you will now navigate to Tools > Serial Monitor. The purpose of this is to see the output of the ESP for if the sensors are not shorted but rather an improper solder joint. For fixing this, you may need to reflow or resolder the solder connections on the ESP board. If the module is then plugged into the computer and does not post all of the data or

any data necessary you may refer to the replacement of the sensors for removing them to be resoldered. If the sensors still do not post the data into the serial monitor, then one or both of the sensors are bad and will need to be replaced. However, if it cannot connect to the Broker then there might be an issue with the Raspberry Pi, and appropriate further troubleshooting will be detailed later. For the data in the serial monitor for finding the specific sensor that is bad, no data can mean both sensors, no light readings, or -1 lux reading will be the BH1750 and everything else is the BME280. Replacements and reinstallation will be covered below.

Replacement of the BME280 is going to involve a small knife or possibly strong hands to remove the hot glue seal placed around the module for waterproofing. Then you will need to remove the BME280 and desolder the wires from the sensor. You will have to make sure the BME280 is the exact module you order to replace it and then you will take the corresponding wired following the wiring picture left down below. After resolving the module you should now work and will need to carefully place the sensor back into the allocated slot for it and reapply hot glue onto the sensor to recreate the seal.

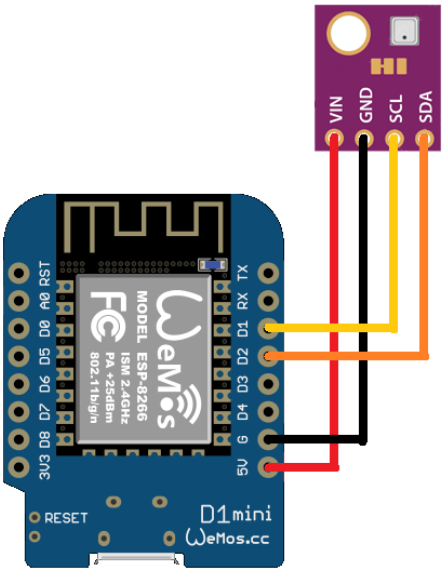


Figure: BME280 Wiring Diagram

Replacement of the BH1750 light sensor is a little more complicated and will need an Acetone and ABS goop-like solution plastic and Nail Polish topcoat for reinstallation and preferably a heat gun but Acetone can work for removal. To remove the module with Acetone you will need a q-tip and keep rubbing the goop seal on the sensor until it is rubbed away or loosened. With a heat gun, you will momentarily blast the back of the sensor where the cured ABS goop is and pull on the wires, you do not want to expose the print for too long under the heat gun or it may cause unwanted warping of the case. After about 10 seconds of flashing the heat gun on the sensor and pulling the sensor should

come loose. From here you will most likely want to double-check the wiring solder joints to ensure no loose solder joints. If the problem persists you will need to unsolder the sensor and follow a wiring diagram below for resoldering the next module.

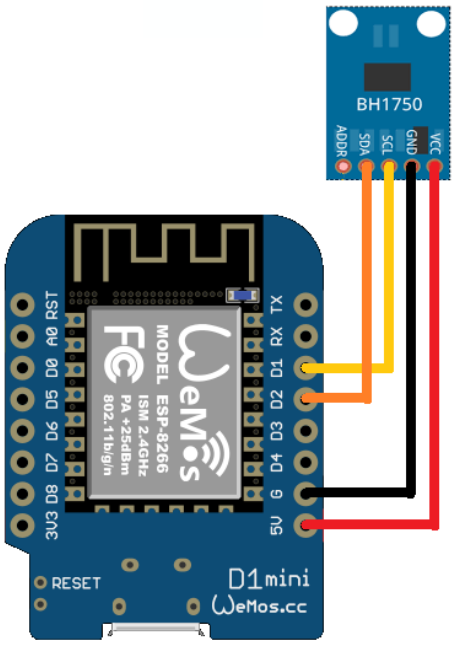


Figure: BH1750 Wiring Diagram

For placing the light sensor back into the case you should use a razor or razor-like object and carve the previous ABS goop and nail polish out of the case and while placing the light sensor you will use the wires to bend the sensor into the case the way you want it angled on a stable surface and then you will apply thick ABS goop on the back of the sensor to cover all holes and air gaps, feel free to apply multiple coats but for it to cure it can take up to 6-12 hrs. After the ABS goop is cured then you will pour some Clear Nail Polish on top of the exposed sensor on top of the case, in this step if it leaks through the other side of the sensor then you will need to apply more ABS goop. The Nail Polish will take about an hour or 2 to become stable enough to be placed in the Greenhouse again.

For replacement of the ESP8266 D1 mini, you will need to disconnect the power and then unsolder the board from the sensors. Before, resoldering the new board you will need to reprogram the new ESP. To get started the Arduino IDE does not include the right libraries for the ESP by default so you will want to navigate to File > Preferences > Additional Boards Manager URLs. From there, you will paste into it "http://arduino.esp8266.com/stable/package_esp8266com_index.json" and then hit the "ok" button. Now you have access to the ESP board libraries to download. To download you will navigate Tools > Board: > Boards Manager and you will get a window, wait until the blue download bar has disappeared to continue. From here you will type where it says in a search bar "Filter your search" and type "D1" and hit enter. Now you should be

able to find the esp8266 Libraries and you will now select the latest version and click install. Once this is done you will navigate to Tools > Board > ESP8266 Boards > WEBMOS D1 R1. Now you will navigate to the tools and have a lot of settings to make sure are correct for the board operation. The settings are posted below.

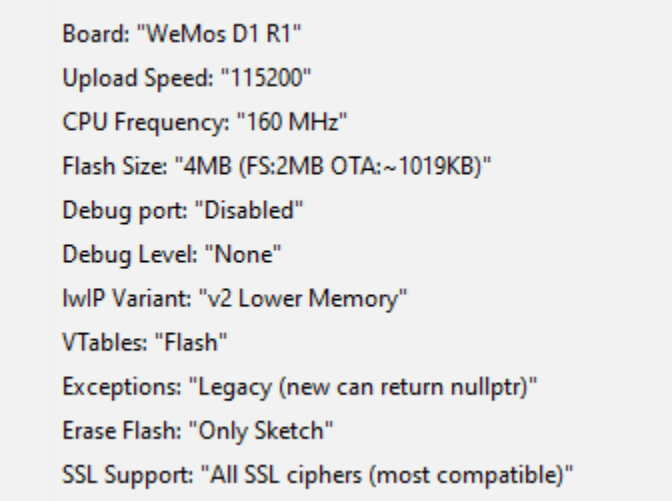


Figure: ESP8266 D1 settings

After this you will go to our Capstone website and Download the program the ESP will run under the filename “MQTT_ESP.ino”. To install the required Sensor libraries you will now have to navigate to Tools > Library Manager and then search for and install the latest versions of “PubSubClient”, “Adafruit BME280 Library” and “BH1750”. Now you are almost ready to get the sensor running. The final step is to navigate the “MQTT_ESP.ino” file after opening it and change the clientID to the ID of the module marked on the side of the case. Then plug the ESP into the computer and make sure the port option inside of tools has selected the ESP, it should be the only COM available. Now you can hit upload and re-solder the wires according to the wiring diagram and do not forget the sleep mode soldered jumper. Now it should be ready to be plugged in and be deployed.

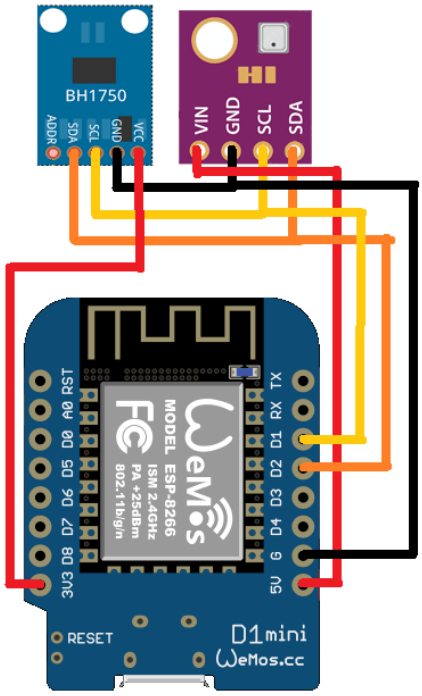
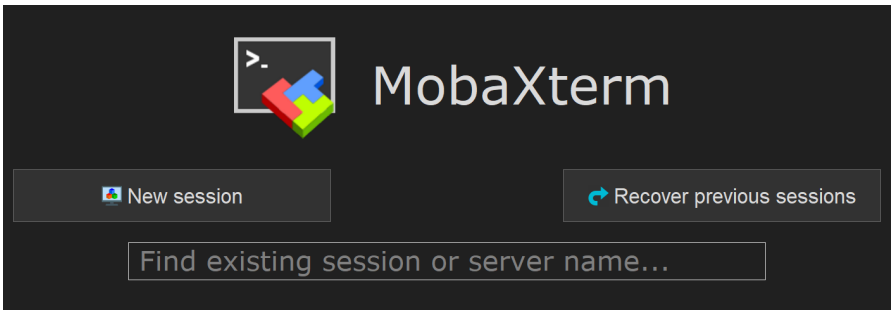
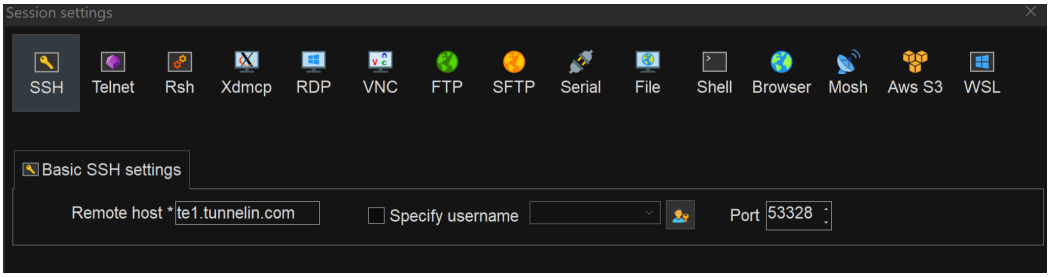


Figure: Wiring Diagram of both sensors

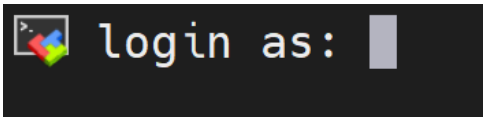
The Raspberry Pi is set up so that you should never need to interact with it. However, some cases may arise where you will need to do some minor troubleshooting with the Raspberry Pi. If data ever stops being uploaded to the website on all modules, the Raspberry Pi will need to be rebooted manually. This can be done by simply unplugging the Raspberry Pi and plugging it back into the wall outlet. All necessary scripts run on the Raspberry Pi will automatically launch on boot and you will not need to interact with it at all after the reboot. If for any reason the Pi is rebooted and data still is not being uploaded from any of the modules, you may need to SSH into the Pi using a program such as PuTTy or MobaXterm. We recommend the latter as its interface is a little easier to navigate. Utilizing a service called TunnelIn we have made it possible to SSH into the Pi from anywhere and you do not need to be connected to NAU's network. Upon launch of MobaXterm, you will be greeted with a welcome screen. From here, you should hit the "New Session" button to start creating a new SSH connection.



Once you hit the button, the screen below will appear. The important fields to fill out here are the “Remote Host” and “Port” fields. These should be “te1.tunnelin.com” on port “53328”. Once this information is filled in, hit the “Ok” button to start the remote session.



The terminal will now ask who you wish to log in as. We have left this as the default user, so you should type in “pi”.



Once logged in to the Raspberry Pi, you should now have full access to a terminal prompt and file manager in MobaXterm. On the left-hand side of the window, you will have access to the file browser. All important scripts are located at “/home/pi/Desktop/”. The main thing that you will be doing in this terminal is typing “sudo reboot” to remotely reboot the Raspberry Pi. Upon entering this, the terminal session will be disconnected and the Pi will reboot. To manage the scheduled tasks, also known as cron jobs, you can type “sudo crontab -e” to edit the current cron jobs.

```
* * * * * sudo python3 /home/pi/Desktop/movefile.py
#*/10 * * * * sudo python3 /home/pi/Desktop/upload.py
0 0 * * 0 sudo python3 /home/pi/Desktop/filemanager.py
2 0 * * 0 sudo /sbin/reboot
```

Shown here are the cron jobs that we set up to execute at specific times. The first line is the script that will move the database files to a USB called DATA every minute of every day. The second line is old and is unnecessary but kept for legacy purposes. The third line is the file manager script which will wipe each database file each Sunday at midnight. The last line is responsible for the weekly reboot of the Pi to avoid errors from the Pi being on too long. The actual script file that manages the ESPs and is run on boot is not included here as it is set up using a different method.

```
1 [Desktop Entry]
2 Type=Application
3 Name=runonboot
4 Comment=
5 Exec= lxterminal -e python3 /home/pi/Desktop/boot.py
6 Hidden=false
7 terminal=true
```

Above is the configuration for the autostart file located at "/home/pi/.config/autostart/runonboot.desktop". This is what is responsible for running the necessary script on startup to communicate with the ESPs and upload to the website. If you ever need to access the database files, those can be found at "/home/pi/Desktop/data/".

f. Conclusion

We hope that the result of our research, prototyping, and design provides you with many years of productive use. As Engineers, we could not be more proud of the work we did as the results we produced. Overall the project was very enjoyable, meeting and working with the client was an honor. We could not have asked for a better client, she was very understanding, trusting, and patient. While we move on into our professional engineering careers, feel free to contact us through our NAU emails so we can help diagnose and/or fix potential problems that may arise. We plan to stay in touch.
Thank you again.

Our Emails:

Alenn Wright (agw73@nau.edu)
Joshua Pollock (Pollock@nau.edu)
Mason Gerace (mag779@nau.edu)

Access to all Scripts on GitHub:

https://github.com/jgp77/NAU-EE486-Research-Greenhouse-Captsone

7. Appendices

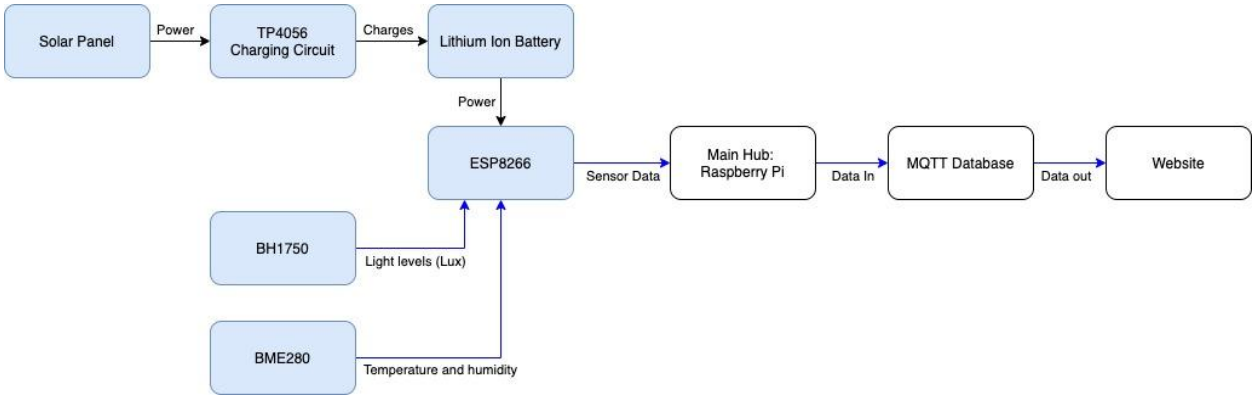


Figure 1: System architecture

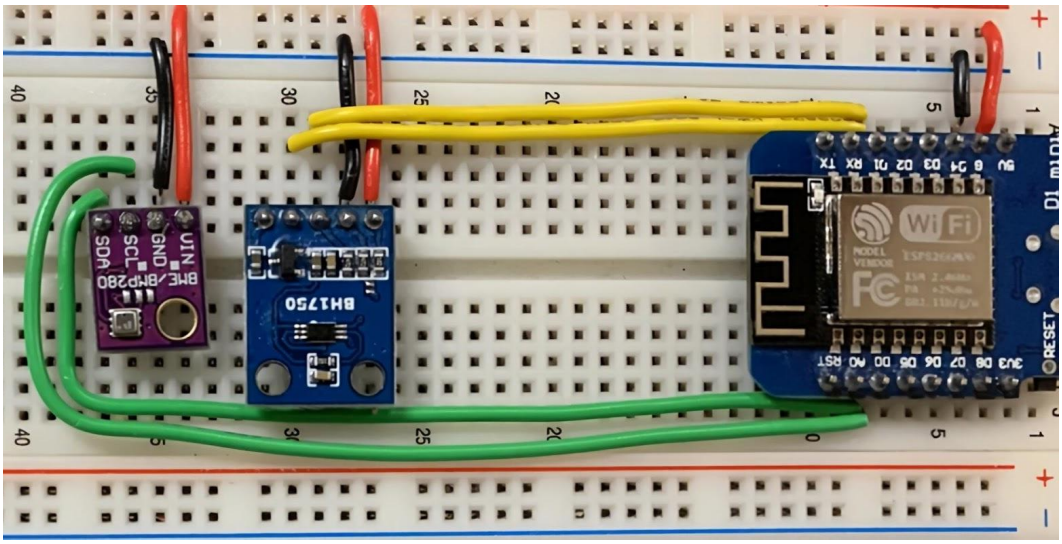


Figure 2: Sensor module on a breadboard

Figure 3: Sensor module inside of a high humidity enclosure

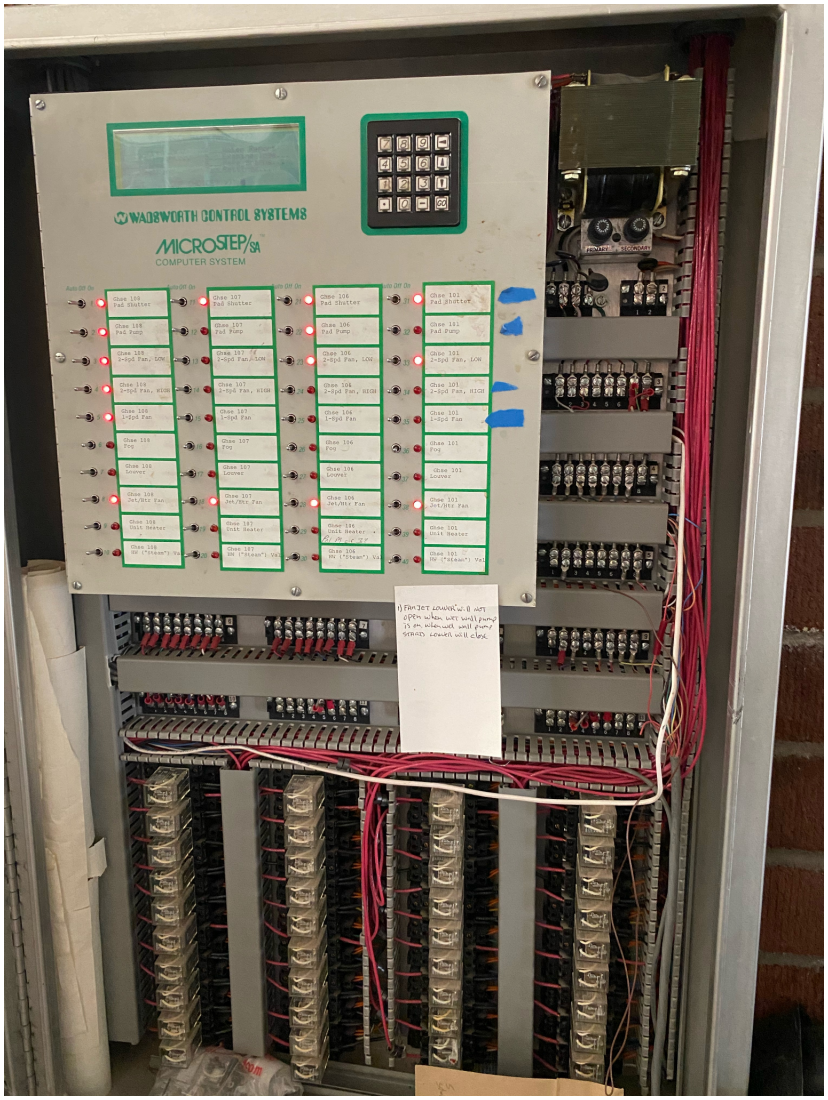


Figure 4: Temperature and Humidity control system.

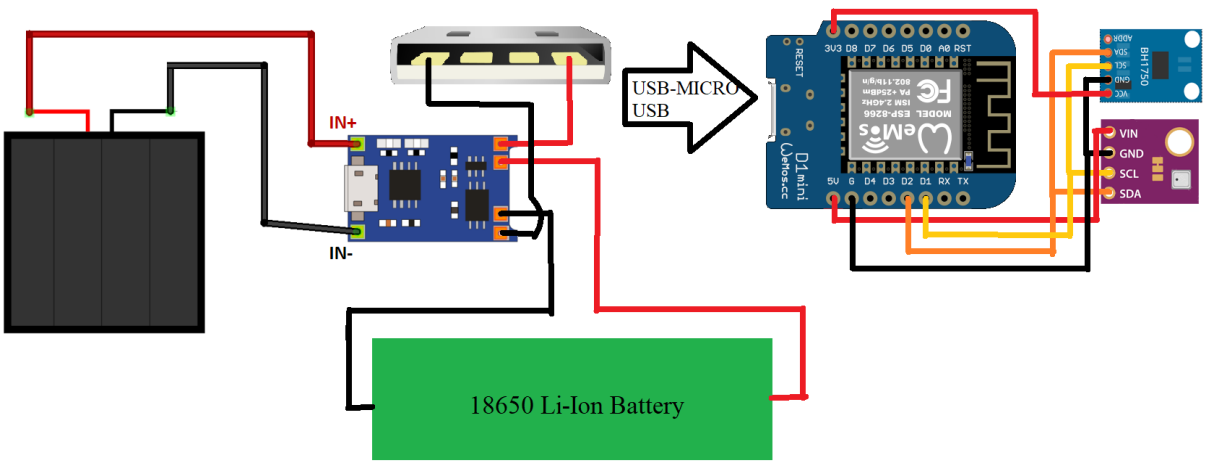


Figure 5: Wiring Diagram