Boston University

Electrical & Computer Engineering

EC464 Greenhouse Prototype Test Report II

by

Team # 13

GreenHouse

Olivia Dorencz odorencz@bu.edu

Laura Morgan lamorgan@bu.edu

Hok Yin Shum hyshum@bu.edu

Kiahna Tucker kiahnat@bu.edu

Qian Zhang zhang1@bu.edu

Submitted: April 16th, 2019

Table of Contents

Summarization of Equipment and Setup **2**

Conclusions based on Test Data **7**

Appendix **9**

Summarization of Equipment and Setup

Our prototype is partitioned into two, major categories: hardware and software. The hardware division consists of a heating system and the structure of the greenhouse, while the software section is composed of a web application, Arduino and sensors, as well as a series of cloud-based services provided by AWS (i.e. Amazon IoT, Amazon DynamoDB, and Amazon Lambda, Amazon Simple Notification Service).

The underlying framework of the greenhouse was setup to provide a strong depiction of the finished project. A cedar box, constructed using a BC plywood base and four 2” x 10” x 3’ weather-treated, cedar lumber walls, served as the foundation of the greenhouse. Four wheels are attached to the underside of the plywood, enabling forward and backward translation. The frame of the greenhouse was created using 1-½” PVC pipe and various different PVC joints. Supports for the frame were created by machining 1-½” inch holes into plastic blocks which where then secured to the box. These supports secure the PVC pipe to the base of the greenhouse. The frame was then covered in a double layer of 6-mil polyethylene, to allow an insulating air gap in between the layers of polyethylene. A door to the inside of the greenhouse was created using two adhesive zippers that were placed in a backward-L shape on the polyethylene.

Within this wooden semi-enclosure, a heating pad is found in its center. The platform of the pad is a 0.75” x 2’ x 2’ BC plywood base with five 1” x 1.5” x 1.5’, pine furring strips placed 2.5” apart. An 18’ incandescent rope light is looped between the furring strips, held in place with 0.5” cable clamps. Four, ½” x 8” x 2’ pieces of BC plywood placed five inches apart serve as support for the subfloor. One-inch corner brackets, fastened to the furring strips of the heating pad, are used to keep the supports upright. In the event the incandescent rope light needs maintenance, the supports can be easily removed by lifting them upwards. Finally, the subfloor rests on top of the walls of the cedar box. It is composed of two ½” x 15” x 32” pieces of BC plywood joined together by a 1-1/16” x 30” nickel continuous hinge. Four 2.5” x 2.5” squares were cut from the outermost corners of the wooden panels to fit around the PVC frame.

Our prototype was intended to test the ability to accurately obtain measurements from our various sensors, confirm that we were able to store the sensor data to the online database over a wifi connection, and visualize this data through a series of graphs in our web application. Power to the heating pad was provided using a Wemo Smart Plug which was set up with IFTTT and connected to a 2.4GHz wifi network, which is the required frequency to operate the smart plug. The heating pad is plugged directly into the smart plug. This allows the heater to be turned on and off depending on current readings within the greenhouse. The NodeMCU can be connected to either a 2.4 GHz or 5GHz wifi network, though the credentials for this wifi network must be re-flashed onto the board if the network credentials change. If the NodeMCU loses wifi connection, it will loop and attempt to reconnect. The Wemo Smart Plug, Arduino, and NodeMCU were all connected to a surge protector that is housed in the base of the greenhouse. The actual electronics are placed in a box that can be moved to either corner of the

greenhouse so the user can optimally choose where it is least likely to be damaged. During testing, we used two DHT11 temperature/humidity sensors, one photoresistor, and three QLOUNI soil moisture sensors with the Flying Fish MH Sensor Series module. We chose to use two temperature sensors as the user should be able to see the temperature information both inside and outside of the greenhouse. We chose to use three soil moisture sensors so the user can customize how many plants they measure the moisture for. This could be expanded to up to six soil moisture sensors, but we felt that three was enough to demonstrate functionality. Our customer also indicated that having an excessive amount of soil moisture sensors was unnecessary as soil moisture would likely be fairly consistent across the different plants in the greenhouse as he would likely water them all at the same time.

The greenhouse is connected to Amazon IOT through a NodeMCU ESP8266 Wifi board, which receives data from the Arduino UNO over a UART connection and sends it to Amazon IOT. NodeMCU has only one analog input, whereas our final product requires multiple analog inputs as the soil moisture sensors require an analog input connection. In the previous testing, we connected the two soil moisture sensors to the single analog input through a multiplexer and set a delay so the soil moisture data could be collected sequentially. However, in the final prototype testing we connect all the sensors to the Arduino UNO board. The Arduino UNO has six analog pins, so we are not limited by the same constraints that the NodeMCU has. As a result, un our final product, all of our sensors will be connected directly to the Arduino UNO. The TX and RX pins of the NodeMCU and Arduino are connected allowing for serial UART communication between the boards. The various sensor readings are packed into a string which the NodeMCU then parses.

When the Arduino and NodeMCU are powered, the sensors collect data every 5 seconds, transfer them to the NodeMCU, and then transfer them to Amazon IoT Core using the MQTT protocol with a formatted JSON package. Some calculations are done on the Arduino to convert raw readings into usable data, prior to transmitting the sensor readings. The NodeMCU sends all readings as a single string to AWS IOT due to issues we were facing with the allocation of dynamic JSON buffers when sending variables separately. When AWS IOT is triggered by the MQTT request, it extracts the sensor readings from the JSON package and sends them to Amazon Lambda. Amazon Lambda provides several different functionalities to our system. The first is to check that the readings received are valid and extract them from the delineated string if valid. At our first testing, we had an issue with sending some readings that did not actually contain accurate data. Through further informal testing, we identified that these packages being sent were just empty packages, due to the NodeMCU reading from the serial connection more often than the Arduino was writing to the serial connection. As perfect synchronization is highly unlikely even when using the same sleep times, we decided to implement a different software solution for this issue. In Amazon Lambda, we added a flag to check whether the data was actually valid data or whether it was a completely empty packet, which we would then ignore. This is either set to 1 if the data is valid, or is 0 if the data is not valid. The data are then extracted from the single string and further processed. The second function is to insert this extracted data into the DynamoDB table in the format of JSON, including the current timestamp. The third function is to check whether enough time has passed to send a notification to the user with the current greenhouse readings. Our client can modify the frequency of these notifications through the web application, including choosing not to receive notifications at all. These are additional notifications reporting the current state of the greenhouse, and are not the same as emergency

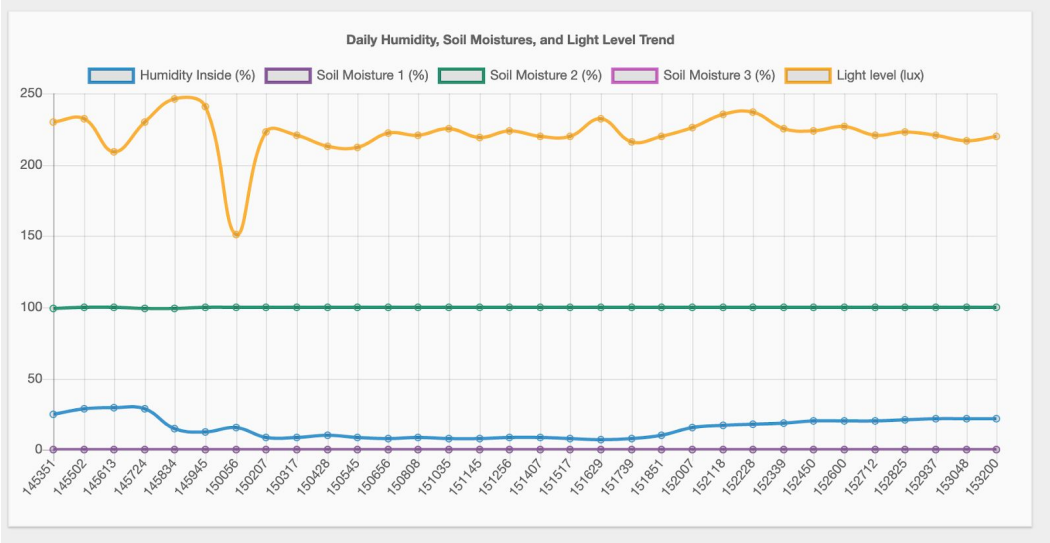
notifications. The fourth function is to read the temperature inside the greenhouse and compare it to the threshold temperature set by the user. There are a few possible cases. 1. if the temperature inside the greenhouse is lower than the threshold temperature setting, and the heater is off, then an alert is sent to the customer's mailbox, and the heater is turned on. 2. If the temperature inside the greenhouse is higher than the threshold temperature setting, and the heater is on, then the heater is turned off. 3. if the temperature inside the greenhouse is higher than the highest temperature setting, then an alert is sent to the customer's mailbox. The fifth function is to record the running time of the heater and store this value in the database. The user can then view their calculated to-date power consumption by month.

The web application is implemented using Python as the backend and Flask as the web framework. During testing, we demonstrated that the web application can establish a database connection to DynamoDB. On the front end, the web application displays graphs of temperature both inside and outside of the greenhouse on a linear plot. It also displays air humidity within the greenhouse and soil moisture. The user has the ability to select which readings are displayed on the graph at a given time. These graphs are implemented using Chart.js. A sidebar allows the user to manually switch the heater on and off as well as set the threshold temperature at which the heater will automatically turn on. The user can also view their calculated to-date power consumption and cost of the greenhouse based on their local electricity costs and the amount of time the heater has been operating.

Description of Measurements

Testing on April 2

After the greenhouse was powered, we first examined the data in the database and checked the plot on the web application. The data were collected successfully and plotted on the web, which proved the successful connection between NodeMCU and Amazon IOT. The DHT11 sensors read an average value of the room temperature of 24°C with humidity of 15%. Since the two sensors were running in the same circumstance, there is not significant difference between two readings.



For testing the regular update message, we set the specific time to be the moment what we were testing added one minutes. After one minute, we received an email from GreenhouseTemAlert, which contained the following information:

Current Time: 20190402-165357
The Temperature inside the greenhouse is: 69.80 degrees.
The Temperature outside the greenhouse is: 71.60 degrees.
The Humidity inside the greenhouse is 30.00 %
The Light level inside the greenhouse is 194
Soil moisture 1 is 80
Soil moisture 2 is 0
Soil moisture 3 is 0
The total power consumption in Apr is 0.2088 Kwh

This email contained the time, temperature inside and outside the greenhouse, the humidity inside the greenhouse, light level, and the soil moisture. Since only the first soil moisture sensor was in the pot, the sensor data was reasonable.

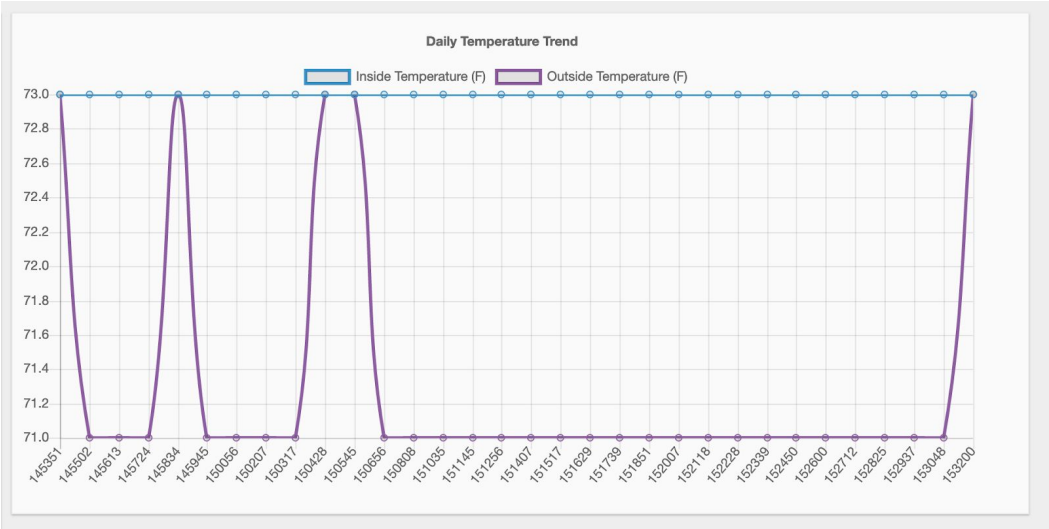
For testing the notification and heater, we needed to increase and decrease the temperature inside the greenhouse. However, for the sake of testing in the controlled environment of the senior design lab, it was hard to change the temperature for the purpose of testing. As a result, we decided to modify the lowest temperature setting on the web application to see the outcome. We first set lowest temperature setting to be 80 degrees, which was higher than the temperature inside the greenhouse. The heater automatically turned on, and the alert message is sent to the mailbox, which contained the message of Low temperature Warning: the current temperature in your greenhouse is 71.60
Then, we set the lowest temperature to be 60 degrees on the web application. The heater turned off automatically, which provided the successful communication between the heater and central system

For testing the function that record the running time of the heater. We recorded the running time of the heater by a clocker, and compared the number to what plot on the web application. The two numbers are identically. By multiplying the time by the power of the heater(140W) and the electricity rate, we were able to estimate the electricity cost of the heater. This number was also included in the reguler update message.

Even though we successfully demonstrated most functionalities, however, we did notice that there were abnormal data in the temperature plot as the plot shows below. There was one temperature reading that was equal to zero. However, most data was reasonable, it was a few data points that jumped abruptly up or down. As a result, we were required to debug it and retest our project on Apr. 11. However, because all inaccurate data points occurred at the same time, we were able to determine that the cause was a problem with our data transmission and not with our sensors. The mechanisms we chose to employ to fix these errors are described in the previous section.

Testing on Apr.11

This time we were only required to demonstrate the successful data collection on the web application. To thoroughly test this functionality, we decided to power the greenhouse and kept collecting data two hours before the testing time. The plot was attached below



This time we proved that we were able to collect the temperature data smoothly and consistently. Though it appears there are some abrupt jumps in the graph above, this graph only shows a two-degree temperature range. We are also fairly confident those “jumps” are just a result of our team members touching the sensor and transferring heat to it, as they occurred at times where we were moving or adjusting circuitry.

Conclusions based on Test Data

We successfully demonstrated that we could read accurate data from all of our sensors, transfer this data over a UART connection from the Arduino to the NodeMCU, send this data from the NodeMCU to AWS over wifi, store the data in the database, display data in a web application, send a notification message to our customers, and control the heater using Smartplug remotely. Compared to the second prototype, this time we included the timestamp with the sensor data, so we were able to plot the data versus the time on the web application. We believe that we have developed full end-to-end implementation of our initial product design.

For controlling the heater, our initial plan for this was to achieve this using an electronic relay. In order to wire our heater to the relay, we would need to physically cut open the power cable of the heater, correctly wire it to the relay, and reseal the cable using electric tape or heat wrap. The risks of doing this incorrectly include completely destroying all of our electronics as well as potential harm to ourselves as our heater will be plugged directly into the wall socket. As a result, we switched to use the Smart Plug. Prof. Hirsch raised the concern that it is possible that the heater could not be switched on or off because of the wifi interference, and he recommended us to use a relay instead. However, Prof. Pisano suggested us not to worry about the internal disconnectedness, so we decided to keep use Smart Plug to control the heater. As a group of all computer engineers whose experience with power electronics is very limited, we felt uncomfortable pursuing the relay as a design solution and had previously informed Prof. Pisano, Prof. Alshaykh, and our TA of our intent to use a Smart Plug. At the end of the day, we decided that the safety of our team was more important than any possible worst-case situations this device may be placed into.

The biggest flaw of our testing on Apr.2 was the abnormal data of the temperature reading. After thorough debugging, we concluded that there were a few possibilities that caused it. 1. The NodeMCU has limited memory that it can allocate to a dynamic JSON buffer. As a result, the NodeMCU occasionally sends an empty data package to AWS IOT. 2. When we were testing the DHT11 sensor using the heating gun, we accidentally damaged the sensor, which resulted in getting abnormal data. This provided a reading that was equivalent to the sensor being disconnected.

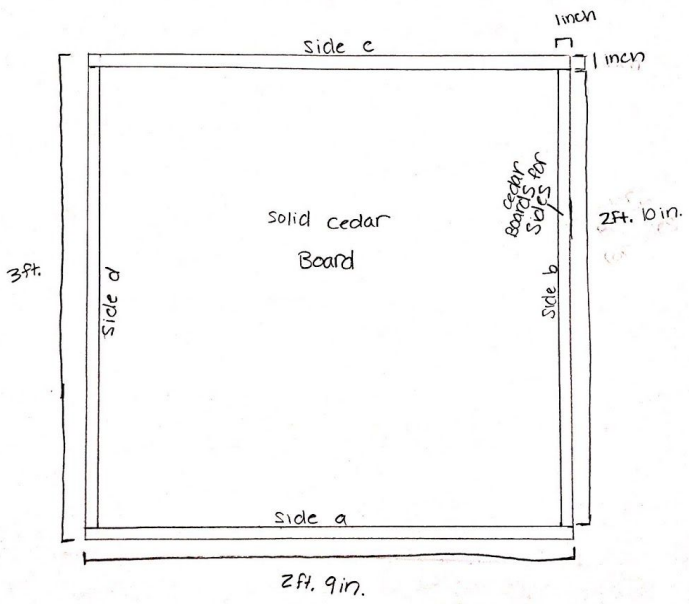
To combat these errors, we developed a few new strategies. 1. In our previous code, we created a variable for every corresponding sensor data. For cutting back the use of memory in the NodeMCU, we combine all the data into a string and send it as a single string to AWS IOT. 2. We includes a constant variable in the data package. This variable is set to be one and sent to AWS IOT from NodeMCU with other sensor data. When Lambda function receives the data package, it examines this variable first. If its value is 0, Lambda function rejects this data package. Lambda function only accepts this data package if the value is 1. 3. We replaced the sensor which we suspected that it had been damaged, and we stopped of

using heating gun to test. After we applied these strategies, we had have not read the abnormal data again, and the greenhouse ran smoothly while we were retesting it on Apr. 11 and during pre-testing that we conducted the weekend prior.

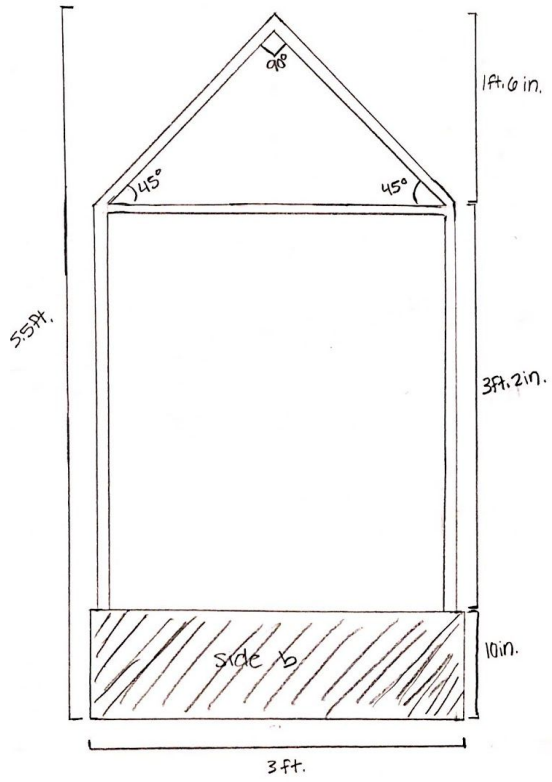
Appendix

1. Photo of the finished, physical structure

2. Photo of an aerial view of the bottom box of the greenhouse structure.



3. Sketch of a planar view of one side of the greenhouse structure. This sketch shows the PVC pipe frame placed in the bottom box. All sides of the frame follow this same design.



4. Photo of the subfloor within the greenhouse.

5. Photo of the wooden subfloor supports and heating pad.