Laboratory #1 Report:

ECE:4880 - Principles of Electrical and Computer Engineering Design

Team 9: Alex Arand, Brandon Cano, Ian Kuk, Rogelio Valle

Design Documentation

Overview of Functionality

The goal of this first lab was to design a thermometer with a web interface. The general requirements for this process were: A computer used for interface and control, a thermometer sensor at the end of a 2 meter long cable, A box containing other hardware components, and lastly a cell phone that could receive messages. The hardware component box was to consist of an enclosed box with a display, button, battery, and power switch. This box was to also be with a sturdy construction that could survive being dropped.

The task for the computer was to show a graph of the last 300 seconds of temperature data from the box. The computer was also required to be able to send a message to a cell phone whenever the temperature went above a certain temperature or below another specified temperature. The box has a display that when the button is clicked shows the current temperature.

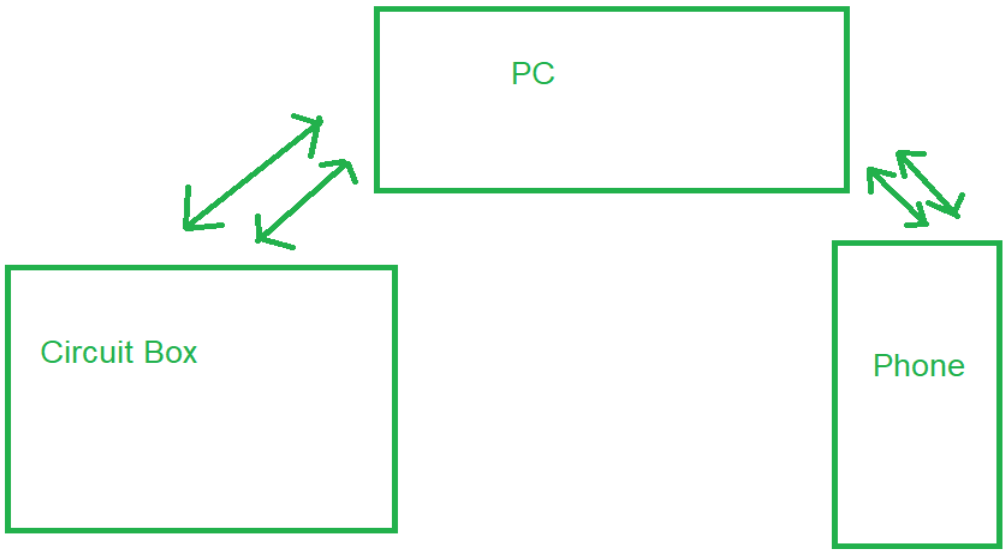


Fig. 1: Diagram of the functionality - High level view

Software Documentation

For the development of this lab, there were 3 different components to work on, the hardware, software, and mechanical design. For the software side of things, we kept all of our code inside of a git repository accessible on github so that it was easier to maintain version control and for everyone to have access to the most recent version of the code.

To begin coding we decided to use the Arduino IDE. In past classes we had experience using microchip studio with an Arduino. The languages used previously were Assembly and C. After some thinking and library availability we decided to use the Arduino language on Arduino IDE due to its ease of use and easy access to libraries. As far as coding in the Arduino language goes it is very similar to C++. Not long after, we came to the decision to use Java as our source for a UI since we found libraries that would help with different components of the lab and we used the Intellij IDE to compile our Java code. The tables below will show all the libraries and softwares utilized with a brief description.

Name	Description
Arduino IDE	- Compiles and uploaded code to the ATmega328P - Easily search for, download, and add libraries to the code
`LiquidCrystal.h`	- Library to control the LCD display - Updated text that would appear on screen
`DHT11.h`	- Library to read values from the temperature sensor - Gives error message readings to identify what the issue is
`AnalogButtons.h`	- Library which can detect button states
`SoftwareSerial.h`	- Library for serial communication - Allows to developer defined rx and tx ports to allow multiple streams of communication - Can write data to hardware devices to be read from other devices
PuTTy	- Computer software that was able to connect to a bluetooth

	device - Would read and display data that was being written to bluetooth device - Writes output to a file, which can be sent elsewhere
Twilio	- Java library used for interfacing with the API - Sends text messages from the Java interface
Javax Swing	- Java UI library - Allows us to create a simple UI which can interact with different elements at once
JFreeChart	- Java library used to create an interactive graph - Work with the Javax Swing library

Table 1: Software and libraries used

For this lab, the software section is split up into two main sections, Java and Arduino. For the Arduino development we used our ATmega328P as the source to control all the hardware and how it communicates with a PC via bluetooth connection. The main idea here was that we had our initial setup of all the hardware components. Then we loop and pulse the sensor every half second or so in case a bad reading occurs. Within the loop, we would also check the button state to see if it was low, if it was then we would turn on our display with either an error message or a temperature reading. In each half second loop we would also send data to the bluetooth device, which could either be a temperature reading or an error code. The sensor library we used would give a 253 integer value if the sensor had a time-out error, which would occur whenever the sensor was not connected or could happen randomly if pulsing for the temperature. Because of this randomness that could occur, we had to double pulse every second to reduce the small chances of the random error to appear.

From the Arduino code when we write the temperature data to the bluetooth device, we use a program called PuTTy which allows us to connect our HC-06 bluetooth device. This

program, once connected to a device, will read in all data being sent to it. In order for this program to work however, there was some setup needed for it to work as intended.

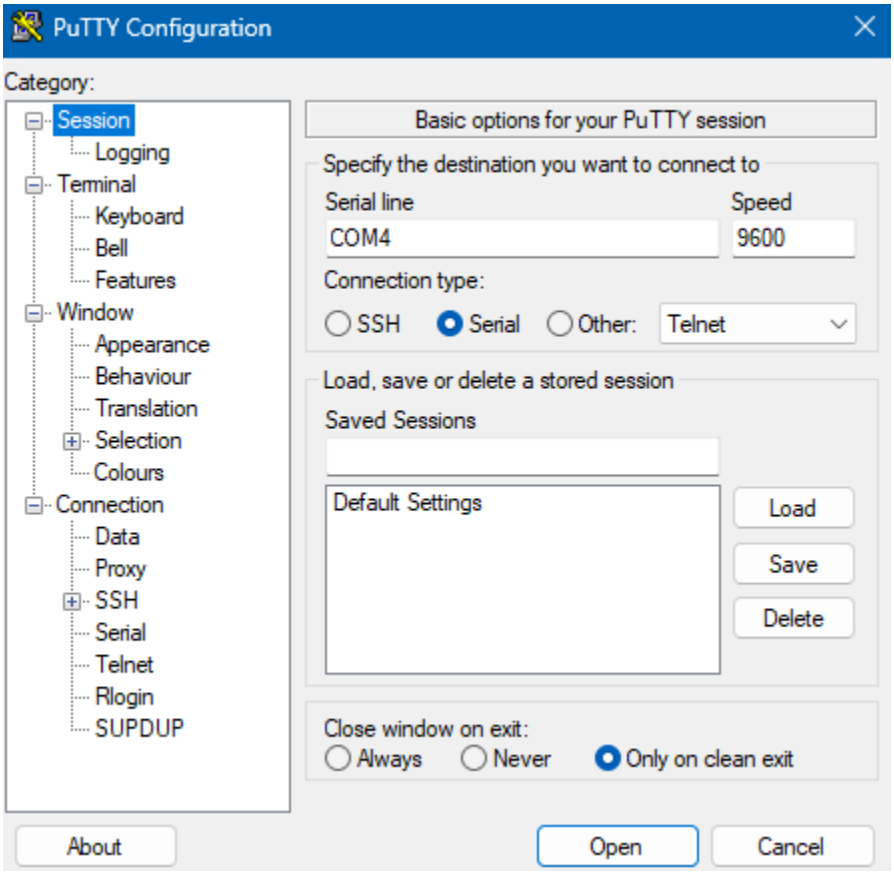


Fig. 2: Image of PuTTy interface showing configuration

The connection type we had to use was serial, and we had our code all running with a 9600 baud rate. We also had to dig into the computer's control panel in order to find which serial line the bluetooth device was on which for us was COM4. Since we write all of our temperature data to a text file, we also needed to specify the file path to where the local text file was stored at. We went with this program because we had issues getting the device to connect to the Java code with any serial connection libraries that we had found, the device would connect with some of the other libraries but the data would never be properly sent or received so to counteract it we used the PuTTy program as the mediator between the two different code bases.

For the Java code, which had the UI, there were a lot of different moving parts here to look at. To start development, we had decided to make smaller starter programs to text different aspects of the code that we were going to need. To start we created a file which can test the text message functionality and get the authentication working. We used this same idea when trying to connect the bluetooth device prior to the change to using the PuTTy program. When we start the program we split into multithreading. We had an internal class with the whole purpose of putting the graph data on it so that the program could run as normal without random pauses if they were on the same thread. On this thread, we call a function in another class `TemperatureFileReader` Which reads in the most recent data from the text file that the PuTTy program is writing to. Also to account for error code values we set up the method like this:

```
public double getRecentTempReading() throws IOException {
     String s;
     lastReadValue = "2";
     while ((s = br.readLine()) != null) {
         lastReadValue = s;
     }
     // if the 253 value is received then the sensor is unplugged
     if (Objects.equals(lastReadValue, "2")) return ErrorCodes.NOT_CONNECTED.code;
     if (Objects.equals(lastReadValue, "253.00")) return ErrorCodes.UNPLUGGED.code;
     if (Objects.equals(lastReadValue, "254.00")) return ErrorCodes.CHECK_SUM_ERROR.code;
     . . .
     return lastReadValue;
```

This reads through the file and gets the most recent value written to it, then we check that value to see if it is an error, because the different errors will correspond to different messages to display to the user.

Error Code	Message
2	- Sensor is not connected to bluetooth
253	- Sensor is unplugged
254	- Sensor has data reading error (check sum)

Table 2: Error code and their messages

Back to the main bulk of the program, we do all the setup for each of the UI elements and we create two separate graphs in the same location, one for F and one for C. We did this because when we tried using one graph it became problematic when switching between the units so having a variable that knows which unit we are reading fixed the small bugs with only having one graph since we can change visibility fairly easily. As for the graph itself, that was the main element in the display, since it had most “moving” parts to it. We noticed later during the fine tuning of the graph that there were some limitations with the library we used. The main issue is that on its own we could not have the range display from 300 … 0 and be able to see the oldest reading moving to the left. To fix this we had to add an extra list we could use to index the elements in reverse after each reading. But to do that we had to clear the data series element which is the object the graph uses to display data points. The code snippet below shows how we added elements to the data series in reverse traversal from the list.

```
dataSeries.clear();
int j = 0;
for (int i=dataArray.size()-1; i>=0; i--) {
    dataSeries.add(i, dataArray.get(j));
    j++;
}
```

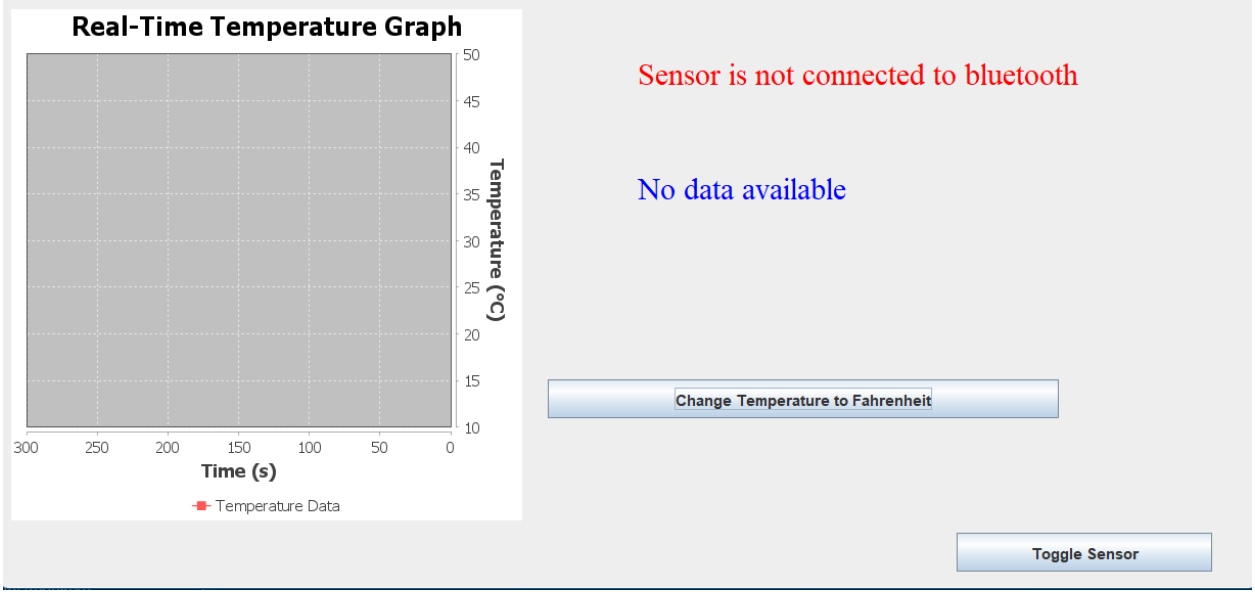


Fig. 3: Image of the UI with the graph, text, and buttons

If an error message was read instead of writing a value we wrote null to the data series since a null value will give us a gap in the graph which is what we needed to show visually that there was a gap in data readings. There was an issue we encountered during testing that disconnecting the bluetooth connection and reconnecting, the Java program would not recognize that the connect was reset so a method was added to create a new instance of the TemperatureFileReader object, this would then reset itself and now check for a new connection once the previous one is lost. We also have a counter that checks to see if it exceeds the 300 second mark, because once it does then we start removing the elements from the data series since we only want to see the 300 most recent seconds. We also set the ranges to be between 10 - 50 C (50 - 122 F) and with the zoom/scrolling capabilities it did not allow the user to scroll outside of the range specified.

We have two additional buttons on screen, one for toggling between the units and the other is meant to control the LCD screen on the arduino. For toggling, we check what temperature value is displayed, then do the opposite.

```
toggleTemp.addActionListener(e -> {
     if ( isTemperatureInFahrenheit )
         isTemperatureInFahrenheit = false;
         ...
     else
         isTemperatureInFahrenheit = true;
         ...
 });
```

For our other button, we had issues getting it to work. We had tried a few different options of setting up using different types of output streams available in the java language but it had to be a similar issue with the bluetooth sensor not being able to read in any data after connecting. After many attempts we decided to just remove the body of code that did not work to avoid any other issues.

For sending a text message, as stated earlier, we had done initial testing of that early one since it only involved a few lines of code. The next issue was that when the bounds were reached, we could send a message but then it would keep sending them until no longer in the bounds, essentially spamming the phone. To overcome this issue we added an extra boolean variable which checks if the text was sent, if it was then do not allow another text until we return to the “safe bounds” and it goes back outside of them.

```
// to send a text message
Twilio.init(ACCOUNT_SID, AUTH_TOKEN);
Message message = Message.creator(User Phone Number, +18775666567, message).create();
...
// to determine when to send a message
        if ((tempF > maxTempF) && !textSent)
```

```
        send_message()
        textSent = true;
    if ((tempF < minTempF) && !textSent && tempF != -1)
        send_message()
        textSent = true;
    // once we get back into the "safe" range we reset the text variable - avoids text spamming
    if (textSent && ((tempF < maxTempF && tempF > minTempF)))
        textSent = false;
```

All in all, all of these software parts are connected together and a lot of the individual sets of logic or code blocks are triggered by the listeners of the different UI elements which are invoked later from user interactions.

Hardware Documentation

All components of this lab were controlled by the ATmega328P arduino microcontroller. We chose this piece of hardware because of our previous experience working with the same microcontroller in previous labs. Along with our previous experience Arduinos have a vast and extensive library collection to make implementation of new parts very easy.

For the LCD we decided to use the HD44780. This decision was made because of our previous working knowledge with this exact LCD. We had several previous documents showing how to wire, and program the LCD making set up very easy.

We used a DHT11 temperature sensor because of its availability in previous class lab kits and the ability to solder wires onto it to meet the length requirement. The bluetooth module was chosen due to pure convenience, as it was the only one available in the electronics shop in the seamen center. In order to wire the temperature Sensor and the bluetooth module we used the markings that were on the parts themselves. The temperature sensor has markers for ground, vcc,

and data. The bluetooth module has markers for RX communication, TX communication, vcc, and ground. The bluetooth module used the built- in “Software Serial” library for the Arduino IDE.

In order to power our circuit we used a 9V battery along with an adapter that allows it to plug into the ATmega328P power port.

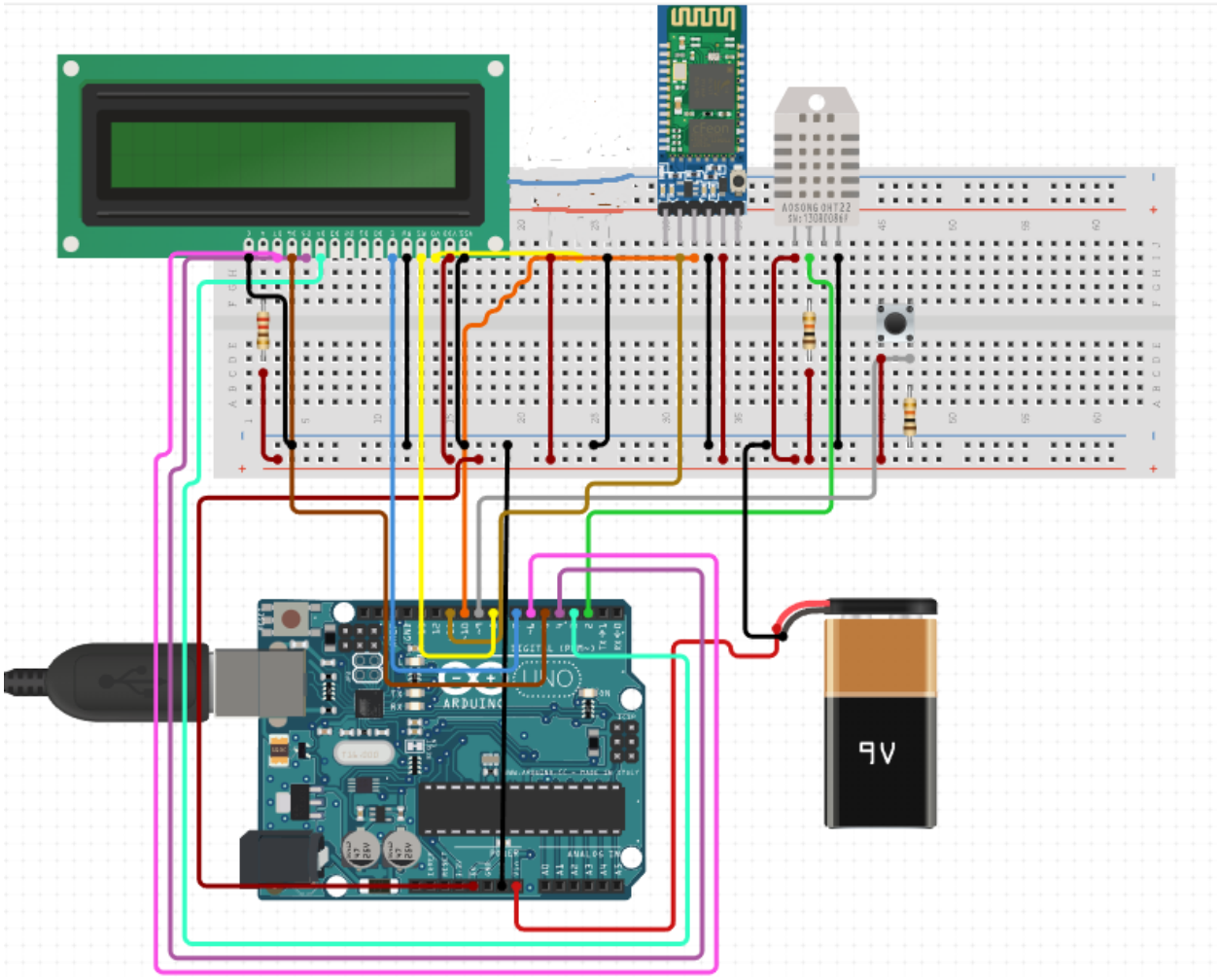


Fig. 4: Diagram of our circuit (parts from left to right: display, bluetooth module, temperature sensor, button)

A majority of the hardware work and testing was done with an Arduino ATmega328P microcontroller, HD44780 display, HC-06 bluetooth module, DHT11 temperature sensor, and a button. These parts were selected via their easy availability from our embedded systems lab kit.

Part	Qty
Arduino Atmega 328p	1
HD44780	1
Button	1
HC-06	1
DHT11	1
9V Battery	1

Table 3: Parts list for circuit design

Mechanical Documentation

The system was implemented inside of a clear tupperware container. On the exterior, small holes were made for the toggle switch, pushbutton, and a built in hole for the temperature sensor wires. The components were secured with expanding spray foam and placed opposite the microcontroller to allow the LCD screen to remain completely visible. For final wiring the battery was secured in the corner, the switch was brought through a hole in the lid, and a small plastic lego piece was inserted through one of the holes to press the push button. All parts were

placed on the spray foam before it expanded and dried, so by the time it did expand it fully encapsulated the components securing them in place.

The latches of the container makes sure the lid stays securely on leaving the final product extremely rugged, compact and shock absorbent. It can be shaken, dropped, or even thrown and still work.

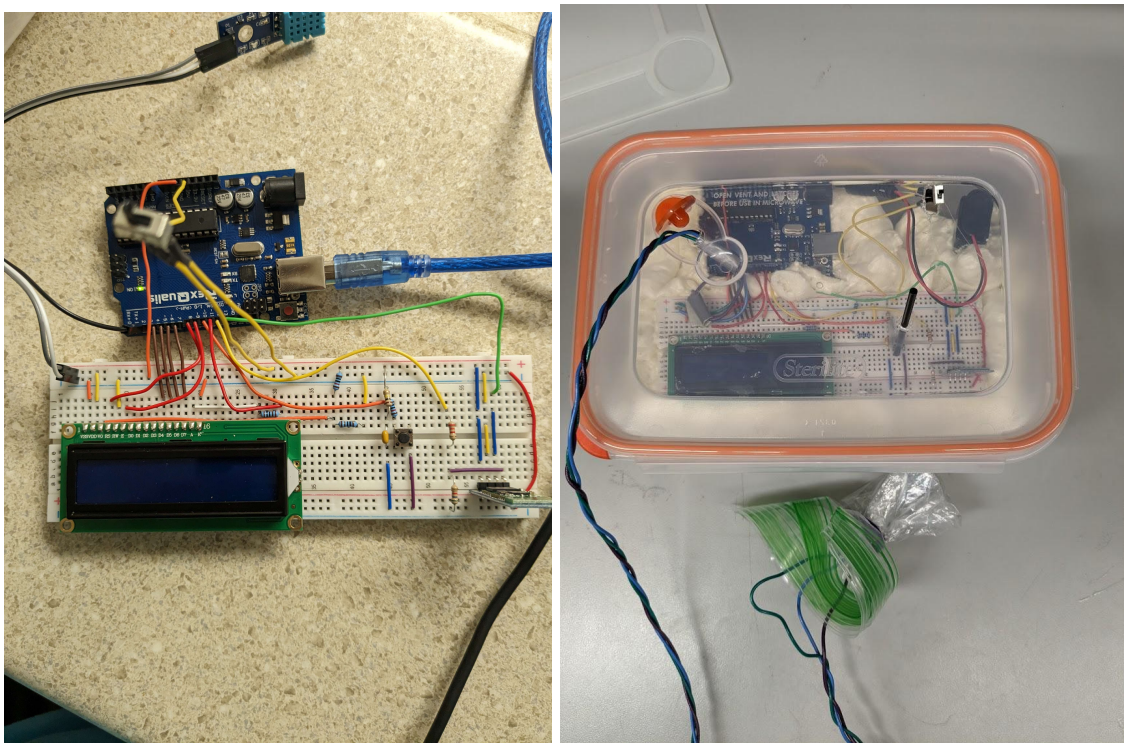


Fig. 5: The core circuit exposed and circuit in the box

Design Process and Experimentation

To begin this lab we all scraped together the lab kits we had left over from embedded systems. Each of these lab kits had all the tools we needed to begin working on the lab. For our microcontroller we used an Arduino ATmega328P since it was what we had from embedded systems. We could have done this project using C or Assembly language as in embedded, but we

chose to use the Arduino language since it had more accessible libraries. For the screen, we reuse our HD44780 which we already had experience with.

One of the biggest pieces we had debate among ourselves was the temperature sensor we were going to use. Our choices were the DHT11 that we already had in our embedded kits or use a temperature probe (ds18b20) we would have to purchase. We decided to use the DHT11 for its availability and we had 4 in case one of them broke. But the probe sensor still looked enticing due to it not needing to be waterproofed. We decided to resolve this issue by wrapping the temperature sensor in a plastic bag and tying off the end.

After deciding on what temperature sensor we were going to use, we began programming to test communication between the arduino and laptop. We first found a service called Twilio that allows us to send a SMS text message to our phones when the temperature gets too high of a reading from the temperature sensor. After importing their library for Java and testing that it works, we proceeded with writing the GUI code that was necessary to display the information we were getting through the arduino.

We did not know if we would be using wired or wireless communication at this point, but at this time we were testing using a direct USB connection to the arduino before knowing we could not. After our initial setup we decided to try out bluetooth modules as opposed to a wifi connection. We made this choice due to ease of use with bluetooth. We were also unsure of how easy or difficult it would be to use the wifi module with campus wifi and if other people could connect to it. After this choice we used the HC-06 Bluetooth module since it was available in the engineering shop.

In order to graph the information we were receiving from the Arduino's serial output, we found several libraries such as the Fazecast JSerialPort and JFreeChart, which allowed us to

transmit information and graph that information respectively. We used these libraries to come up with some initial tests for communication between the Arduino and our laptop, and make sure that the JFreeChart library was taking in the data from the Arduino's serial monitor properly. Eventually, we had to move away from some of the initial serial communication libraries we had found, as they were incompatible with the wireless communication we had set up, but did work with a wired connection directly to the Arduino. Many of these libraries would have required a lot of multithreading, which is something that could cause difficult problems to debug. We figured there would be an easier solution, so realizing this, we saved a lot of time by finding the PuTTy program, which was compatible with our HC-06 Wireless transceiver and performed the same job as the initial libraries we were using while the Arduino was communicating with wired communication.

One of the most open ended parts of this project with multiple routes was the design for our box container. We had our circuit completed before the first progress update and we pondered a lot about what to do in terms of design. From different tupperwares to tape or sealant. On a trip to a hardware store we found a container with a small hole to fit cables for the temperature sensor. It was decided that all components could be secured inside a tupperware container with expanding spray foam, and holes to each interactive component could be cut into the box. The tupperware container was decided, as opposed to a custom 3D printed box, because of its durability, cost efficiency, and accessibility. One of the larger debates was on how to secure the components inside of the box. The ideas ranged from cutting styrofoam to encapsulate it, to taping everything down. Ultimately we landed on the spray foam route. We chose to use expanding spray foam because it's soft when first spray and then extremely hard, but still

malleable, when dried. Due to the nature of the spray foam we were able to place the circuit inside while it was still soft and it perfectly formed around the circuit keeping it in place.

Test Report

Test Number	Requirement	Criteria	Pass/ Fail	Date
1	Graph Test	Get a temperature graph to show data	Pass	9/11/2023
2	SMS Test	Get a java program to send a text	Pass	9/11/2023
3	Sensor unplugged Test	Graph stops when sensor unplugged	Pass	9/18/2023
4	Switch Test	Make the system turn off with a switch	Pass	9/09/2023
5	Battery Power Test	Power the system with a battery	Pass	9/22/2023
6	Bluetooth connection test	Bluetooth module connects to computer	Pass	9/22/2023
7	Temperature sensor test	Get a reading from the temperature	Pass	9/04/2023

		sensor		
8	**LCD turn on test**	**Make sure the LCD turns on when connected**	**Pass**	**9/04/2023**
9	**LCD display text**	**Display custom messages on the LCD**	**Pass**	**9/04/2023**
10	**Computer control LCD**	**Computer sends a signal to change the LCD content**	**Fail**	**9/28/2023**
11	**Bluetooth data transfer test**	**Transfer data using bluetooth module**	**Pass**	**9/24/2023**
12	**Arduino test**	**Make sure arduino connects to computer**	**Pass**	**9/04/2023**
13	**Connectors easily connected and disconnected**	**The wiring for the temp sensor is easily disconnected**	**Pass**	**9/09/2023**
14	**Switch “On” test**	**The switch turns the system on**	**Pass**	**9/09/2023**
15	**Switch “Off” test**	**The switch turns the system off**	**Pass**	**9/09/2023**

16	**Button press test**	**Button displays temperature on LCD**	**Pass**	**9/08/2023**
17	**Temperature range test**	**The temp sensor shows a wide range of temperatures**	**Pass**	**9/09/2023**
18	**Ice Water Test**	**Thermometer shows a freezing temperature in ice water (near 0C)**	**Pass**	**9/25/2023**
19	**Room temperature test**	**Thermometer shows 20-22 C in a normal room**	**Pass**	**9/09/2023**
20	**Palm of hand temperature test**	**Temperature sensor goes up slowly when held in the palm of the hand**	**Pass**	**9/09/2023**
21	**Convert C/F test**	**Conversion from C to F is accurate and displayed**	**Pass**	**9/09/2023**
22	**Third box enclosed**	**The box is enclosed and fall ready**	**Pass**	**9/25/2023**

23	**Third box drop test**	**The box survived a drop from our countertop**	**Pass**	**9/25/2023**
24	**Third box can be turned upside down**	**The box can be turned upside down and keep functionality**	**Pass**	**9/25/2023**
25	**Thermometer at the end of a 2m cable**	**The thermometer is at the end of a cable we measured to be a little over 2m**	**Pass**	**9/25/2023**
26	**Error conditions test**	**If there is an error that message would display on the LCD**	**Pass**	**9/11 /2023**
27	**No data available test**	**If there is no data, a no data message would be displayed**	**Pass**	**9/11/2023**
28	**Past readings graph test**	**The graph showed 300 seconds of past readings**	**Pass**	**9/18/2023**
29	**Sensor unplugged and replugged test**	**The graph would have a gap in info**	**Pass**	**9/18/2023**

		if the sensor was unplugged and replugged in		
30	**Scalable graph test**	**The graph is scalable when resized**	**Fail**	**9/22/2023**

Table 4: Test cases with results and dates

Project Retrospective

Outcome: Our project met nearly every requirement. The one requirement we were unable to fulfill was changing the information on the LCD from the web app. After the checkoff we felt that even though not everything was completed our design process was still effective. Our team communicated, and worked extremely well with each other. Everyone in the group had their assigned tasks and completed them, leading to everyone in the group contributing equally. We believe our failure to complete every requirement was due to the changing of information about the project. We allocated a set time slot to work on the bluetooth communication, but then we changed our plans after hearing the circuit may be wired. It was not until a week later that we were informed that it had to be an ethernet connection and not the USB connection we were currently using. This caused our team to scramble to try and get the bluetooth portion up and running. The bluetooth connection proved challenging leaving us very little time to develop a way to change the LCD from the computer. In the future we plan to establish better communication with the product designers in order to create a more foolproof work schedule.

Development Ideology: Our group used an agile development method, we believe this was the perfect development method to use. Due to using some unfamiliar components we knew we would have to pivot and change our approach to many problems. We broke up our time into three main sprints. The first sprint was getting the main hardware components and displaying basic information on the LCD. The second sprint consisted of making sure the temperature sensor could send data, making the web app and displaying the information from the sensor on a graph, and sending SMS messages and certain values. The third and final sprint was about establishing bluetooth connection, sending all necessary information over bluetooth, having the web app interact with the LCD, and securing all components inside the box.

Ian (CSE): Researched necessary components. Responsible for most of the implementation of hardware requirements. Resized, soldered and connected all components where it was deemed necessary. Secured circuit in place to make sure it could withstand the drop test.

Brandon (CSE): Researched a way to write data to a bluetooth component and be able to read it in from a computer. After that most of the work was on adding to the arduino and java code getting everything working and communicating with each other, then also fine tuning bugs and other small issues that were encountered.

Alex (CSE): Researched initial libraries and tools that would be required to write initial versions of the Java code, established SMS communication, created the GUI and altered both Arduino and Java code to get initial proof of concept code when we were only using the direct connection to the Arduino.

Rogelio (CSE): Researched necessary components and tested components. Made the initial circuit and figured out where everything should go. Wrote the base Arduino code that was used throughout the project that enabled the LCD, temperature sensor, and bluetooth module. Fixed any hardware issues that arose throughout the project.

Gantt Chart

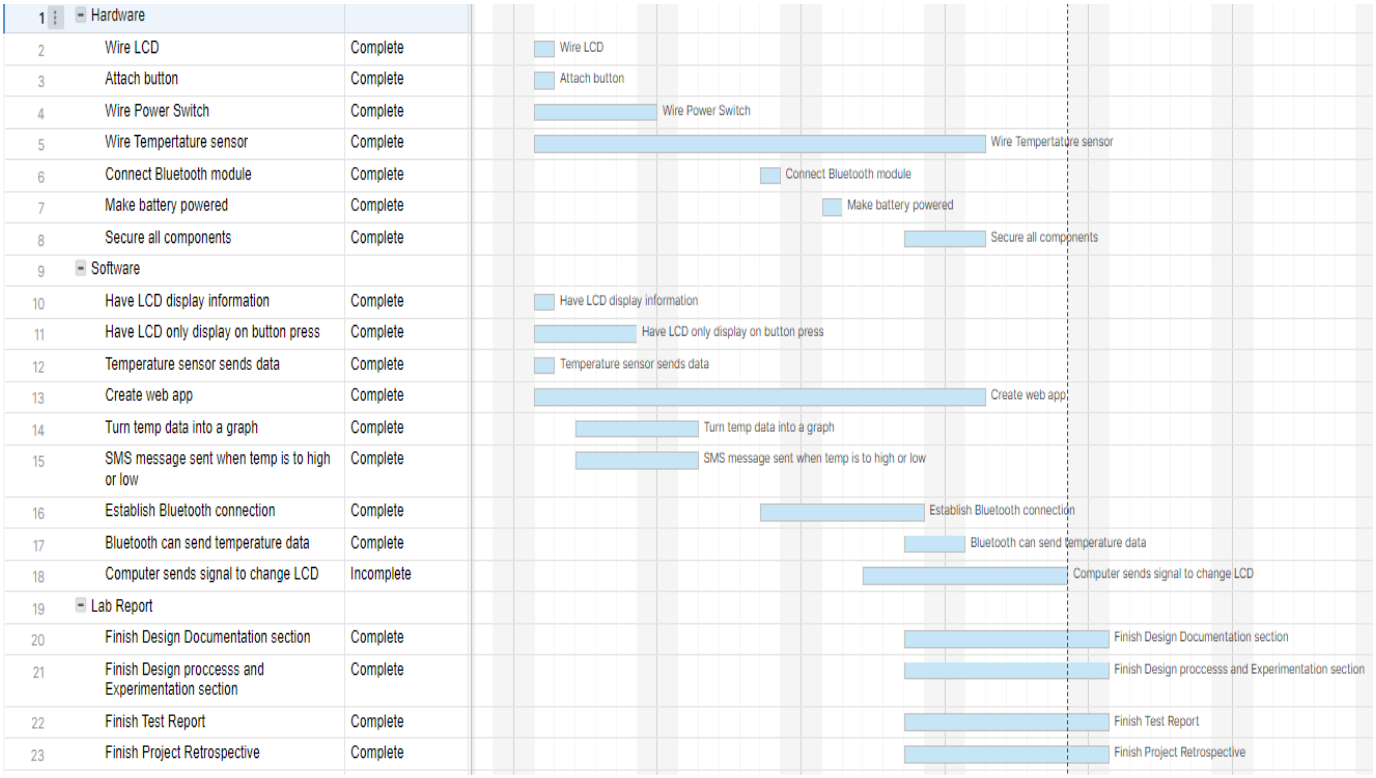
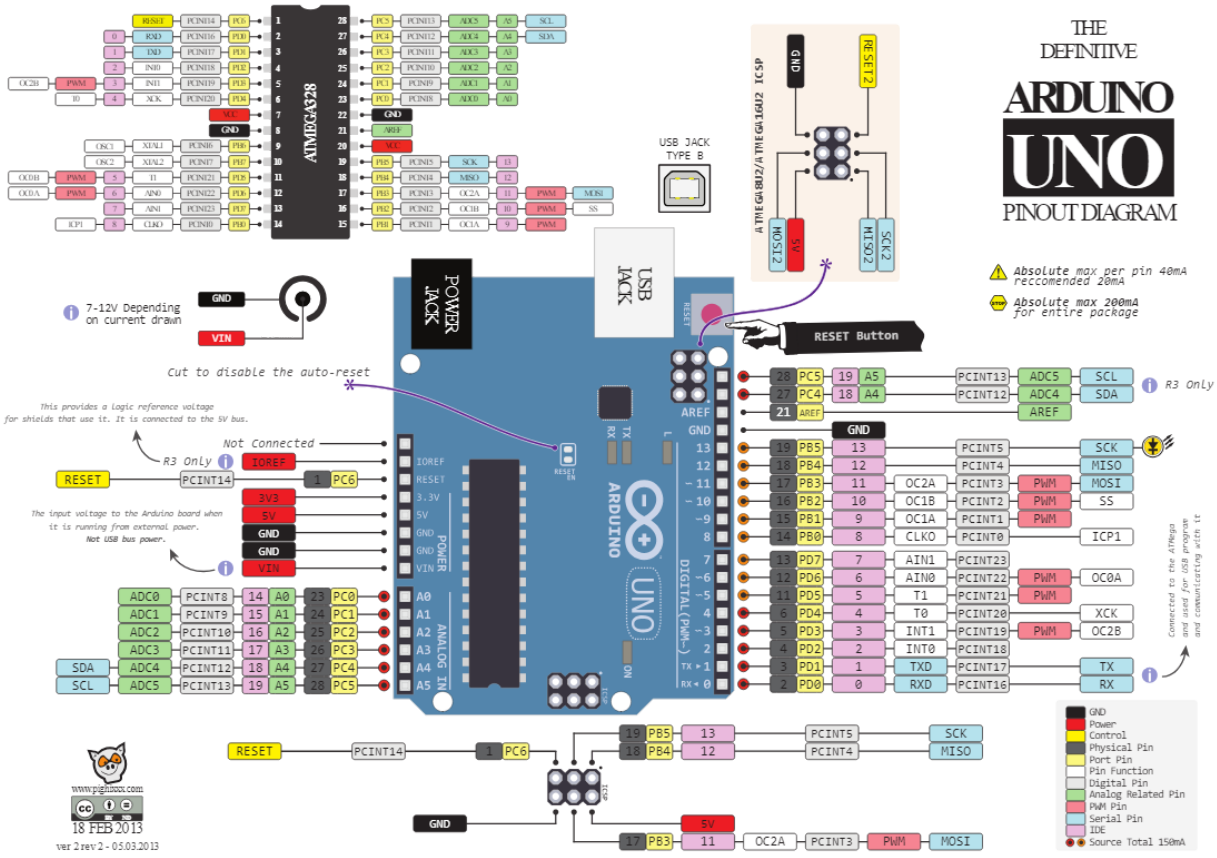


Fig. 5: Gantt chart

Appendix & References

ATmega328P datasheet:



DHT11 datasheet:
https://components101.com/sensors/dht11-temperature-sensor

HC-06 datasheet:
https://components101.com/wireless/hc-06-bluetooth-module-pinout-datasheet

HD44780 datasheet:
https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

PuTTy Software:
https://www.putty.org/

DHT11 Library:
https://github.com/dhrubasaha08/DHT11