Weather Box

Tina Maurer, EE, Anthony Mendez, CSE, Stephan Kim, CSE, and Christian Norton, CSE

Abstract—**As part of pre-flight preparations, drone operators must check the local weather conditions to ensure a safe and successful flight. While commercial weather stations can effectively collect data for a specified area at the macroscale, weather conditions in that area at the microscale can vary greatly. Since flight conditions can be greatly affected by these constraints, drone operators need a more accurate localized weather map reading for the area of flight. Weather Box will create this localized map in a network of battery powered sensor modules to provide drone users with the required information via a website and application. Our product will allow operators to quickly decide whether the conditions are suitable for safe drone flight.**

I. Introduction

Weather forecasts for certain areas are not the most accurate since weather stations could be placed miles apart from each other. Interpolation between stations can be used to calculate outdoor weather in areas between stations but depending on the distance this method can be incredibly unreliable.

A. Significance

Weather has a massive role in many modern infrastructures and systems, either directly or indirectly. That is why it is so important to monitor and record weather data. Weather can affect a huge range of systems such as transportation, the operation of power systems, and even the ability for public services to enforce the law or give assistance. Every year, weather delays cost airlines and customers several billion dollars [1] [2]. Power systems can be shut down due to issues like icing or heavy wind [3]. These shutdowns can cost as anywhere between 30 billion dollars and 130 billion dollars annually [4]. Wind farms often run their turbines based on wind and weather predictions to optimize their output to their costs of operation. In many cases where law enforcement and public services want to use drones in their operations they cannot because the weather is too inclement for the drone to operate safely. For most drone operation, weather can have drastic effects on flight stability and flight duration. Cold weather causes batteries to drain faster than usual leading to shorter flight times. To combat high winds, the blades need to spin faster to compensate. This causes the battery to drain faster as well shortening the flight duration even further [17]. High humidity can cause a build up of moisture and could possibly damage the drone's electronics [18].

Overall, weather can have a huge impact on the performance and function of many modern infrastructures and systems. In many cases, a far more accurate and small-scale rendering of weather can have a massive impact on the design and operations of these systems. Airlines can have fewer delays as they can see exactly where an approaching storm is. Power grid developers can put more infrastructure in place to reduce the damage in locations they know will have harsher conditions. Officers, EMS, and firefighters can be better equipped to handle, or avoid, harsher weather conditions during operation and to see what their options are during an operation.

B. Context and Existing Products

There are a few products on the market that help increase the resolution of weather forecasts. A common solution for this problem is the use of weather balloons. There are about 800 locations around the globe that do bidaily releases of weather balloons to gather atmospheric data in addition to specially requested launches. Each year the NWS releases about 70,000 weather balloons to gather data [13]. Although weather balloons are relatively easy to deploy, they only give a single vertical line of measurements, can only be used once, have a chance of the payload never being recovered, and have limits on where they could be deployed. Another solution would be to have people on the ground with various sensors to take measurements themselves. Although this could be incredibly accurate, it is highly inefficient since it requires people to be present on site and record values at various points when that might not even be entirely necessary. We refer the interested reader to *Appendix Section A* for more details.

C. Societal Impacts

One of our main areas of concern is the public safety aspect of drone flight. In some cases, it is much too dangerous to fly a drone outside due to weather conditions such as wind speeds. Since weather forecasts cannot give an accurate reading of the weather at a very small area, our system would

Requirement	Specification	Value
Portable	Weight	< 1lb
	Battery powered	24hr battery life
Accurate measurements	Wind speed	Accurate with 95% confidence
	Temperature	
	Humidity	
	Atmospheric pressure	
	Air quality	
	Dust	
Send data remotely	Connect to Wi-Fi and send data to web server	2.4 GHz band to connect to Wi-Fi
Cost	Maximum cost for manufacturing a sensor package	$120

Table 1: Requirements and Specifications

help drone operators better judge if an area is safe to fly through. In addition, our system can help weather stations provide a more accurate forecast to specific areas.

D. Requirements Analysis and Specifications

We required each Weather Box enclosure to be portable, able to accurately measure wind speed, temperature, humidity, atmospheric pressure, air quality, dust, and send data remotely.

II. Design

A. Overview

For our solution we have three independent, battery-powered enclosures that sends weather data via Wi-Fi to our website. Below is the block diagram of our proposed solution.

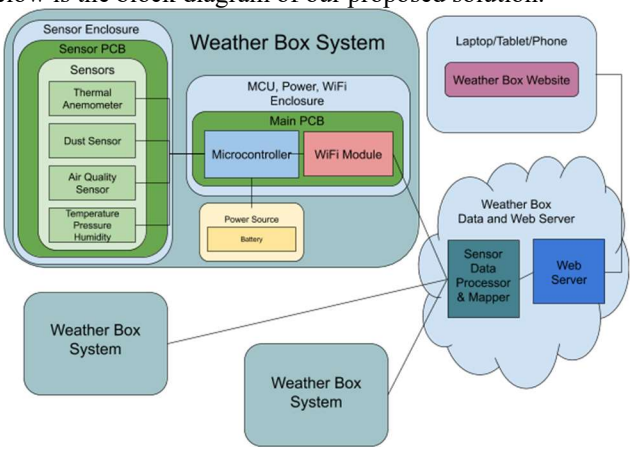

Figure 1. Block Diagram

Our block diagram includes four main blocks. The first of which is the sensor enclosure. The sensor enclosure houses the sensor PCB with a thermal anemometer, dust sensor, air quality sensor, and a 3-in-1 temperature, pressure, and humidity sensor. The next section is the main PCB. This PCB contains the microcontroller, the power circuit, and the Wi-Fi module. The main PCB provides power and communicates to the sensor PCB via ribbon cable. The main PCB is powered by an external battery pack. The Wi-Fi modules communicate with the Azure Weather Box web server using the HTTP protocol. The third block, the web server, handles receiving data from the Weather Box systems. The last section is the website. The website displays the map of the sensors and their readings. This data is retrieved periodically from the web server.

The given specifications in Table 1 presented an interesting challenge for us. We need a sensing package system that would be light-weight, power-efficient, measure accurate data consistently, for under $120. Because we are trying to do so much with so little money, we had some tradeoffs. The first of which is the accuracy of the sensors. For example, the temperature sensor has a range of accuracy of ± 1° Celsius. The dust sensor had the largest range of inaccuracy. The dust sensor has a voltage representing no dust between 0 – 1.5 volts. Outside of the sensors, we opted to go with an AA battery stack for each system's power storage. It was much cheaper and easier to implement than a lithium ion battery. However, it was slightly heavier than a single lithium ion battery would have been. Tradeoffs such as these caused us to make compromises to achieve a well-functioning system that was also within budget and power constraints.

B. Power Source

One of the specifications of our network of weather sensor packages is a battery life of at least 24 hours. To meet this specification, we are using a +12V battery stack of AA batteries. This creates the voltage line necessary for the thermal anemometer, as it requires a supply voltage of 9 to 12V. By creating the battery stack, we allow ourselves a buffer for the batteries in case they drop below the ideal stack voltage. From the +12V stack, we previously used an LDO [19], or low dropout regulator, which is a linear regulator that steps the +12V down to create a +5V line. We then used a similar LDO to again step down the voltage and create a +3.3V line for our MDR prototype. One consideration that we had when developing a printed circuit board is the constant power consumption of a linear regulator. Though the linear regulators we previously used could handle up to 1.5A and allow us to meet the current power specification (as we are drawing approximately 110mA and have a battery life of about 28 hours), a switching regulator allowed us to optimize our sensor packages [19]. This is because the switching regulator is not always in the "on" state, thus does not constantly regulate, theoretically using less power, unlike the linear regulator. So, for CDR, seen in Figure 4, we used switching regulators from +12V to +5V and from +12V to +3.3V. Though the switching regulators gave us less of a ceiling in that the maximum current they could handle is 500mA, it was still well within range for our electronics. We were not able to test battery life or power consumption with the new regulators due to the unexpectedly shortened time of the project, but still created a working product for CDR using the switching regulators in our printed circuit boards.

C. Microcontroller

We are using an STM32 microcontroller in which the firmware is written in C using ST Microelectronics' own HAL API [6]. For this block, we needed to learn the HAL API as well as the locations and functions of different pins on our microcontroller. This part was like Computer Systems Lab, where we had to read datasheets on our microcontroller and figure out the different quirks of our specific one. Our microcontroller takes the data from the sensors utilizing multiple protocols for serial communication, and then sends the data via the Wi-Fi module to our website. Before the data is sent, the microcontroller performs some processing and formatting on the data. For the wind sensor, the output is a voltage, so we take the voltage and utilize the microcontroller to calculate wind speed using an equation given by Modern Device to get the wind speed [7]. For temperature, pressure, and humidity, we used Bosch's calibration formulas to get our temperature in Celsius, humidity in relative humidity percentage, and atmospheric pressure in hPa [8]. The ams air quality sensor has a built in ADC, so we just needed to read in the values from the sensor, and it is already ready for transmission [10]. The Sharp dust sensor is also an analog

sensor, so we connected the output of the sensor to the ADC and then calculated the dust density in percentage by using a formula given by Sharp [9]. We tested our firmware using STM's own CubeIDE which is an Eclipse-based IDE and used the debugger to check values coming in from the sensors.

D. Sensors

We have four different sensors connected to our microcontroller. Much like the microcontroller, the techniques used to create this block are very similar to what we did in Computer Systems Lab, where we would read the various datasheets of each sensor and figure out how to get our microcontroller to interact with the different sensors. For wind speed, we are using a Modern Device Thermal Anemometer which uses two thermistors and compares how much power it takes for the exposed thermistor to reach the same temperature as the idle one and gives an output voltage based off that [7]. The output voltage is then input into a formula to get wind speed. For MDR, this sensor's output voltage is read via an external ADC that utilizes the SPI protocol. For CDR, we switched to using the microcontroller's internal ADC. We did this because the internal ADC is rated for 12 bits, compared to the external ADC's 10 bits. Also, to simplify the circuit and save on power usage. For MDR, we are also using a Bosch BME280, which is a 3-in-1 temperature, pressure, and humidity sensor [8]. This sensor utilizes the I^2C protocol. Our other two sensors are the SHARP GP2Y1010AU0F dust sensor and the ams CCS811B-JOPD500 air quality sensor. These have been implemented for CDR. The ams air quality sensor has an internal ADC that it uses to convert its voltages to the digital values, so no math needs to be done on the microcontroller side. The digital values were read through I^2C [10]. It is important to mention the manufacturer does not provide a C driver file for this sensor. We had to create our own C driver file. Much like the ams air quality sensor, the Sharp dust sensor also did not include any drivers, so we had to read the analog output and calculate the dust density in our code [9]. For testing, we set up different environments to simulate changes in weather and compare measurements from our system to other sensors in the same conditions for MDR. However, due to COVID-19 delaying PCB manufacturing, we were not able to test for CDR. We refer the interested reader to *Section III. The Product D. Product Performance* for more on COVID-19 effects and to *Appendix Section C* for more on testing.

E. Wi-Fi Module

For enabling Wi-Fi connectivity, we are using a development board from SparkFun which utilizes, Espressif Systems' ESP8266 chip for Wi-Fi connectivity [11]. For this, we had to learn how AT commands for modems worked and send them to the Wi-Fi module via UART from the microcontroller. In addition, we had to learn how to format an HTTP POST request in order to send data to our website. To implement this block, we had to utilize knowledge from our Computer Networks course for the HTTP messages as well as techniques from our various programming courses in learning the commands to program the module. For testing this, we sent some data to our website and checked the website's logs to see if the request would go through. In addition, we tested reading data from our website into our microcontroller. Using Direct Memory Access (DMA) we could access everything read by the Wi-Fi module as well as everything sent to it by starting and stopping the DMA at certain points in our code. We did this to read a timestamp from our website since our microcontroller does not have a separate clock and battery to keep track of time on the Weather Box.

F. Weather Box Data, Web Server, and Website

An integral part to our project is the Web Server, which hosts the communications with the SQL database, Weather Box systems, and delivering the website. For our hosting provider, we are using Microsoft Azure [12]. Microsoft Azure provides $100 credit for students, along with some free resources. The web server is programmed in Python, utilizing the Flask library to perform web server functionalities [21]. The website shows a map of the Weather Boxes on the map using the Google Maps API in JavaScript [22]. The Maps API has marker objects and information window objects. We place the markers where the Weather Boxes are, and the information windows attached to the markers. The way the website retrieves information for each system is as follows.

The JavaScript running on the website will ping the server every 10 seconds with a HTTP GET data request. The Python Flask web server would receive this request and retrieve the most recent Weather Data for each system. The server then programs this data into a JSON format that can be easily understood by the JavaScript. The JavaScript client receives the JSON message, parses it for the weather data, formats the plot, and displays it on the information window for each system. For retrieving the information from each Weather Box, it is as follows.

The Weather Boxes are currently programmed to measure and send data every 10 seconds. The Weather Boxes pings the server for a HTTP POST data request. The server acknowledges the Weather Boxes data request, then measures the data from each of its sensors. Afterwards, the microcontroller formats the data into a JSON message. The JSON message is then sent to the server. The web server parses the message and stores it into the SQL database with a timestamp.

The website enables verified users to make changes to the systems locations on the table. To verify a user, we implemented the Google Sign-In API [23]. Once a user signs into their Google account, a JavaScript function is called and sends a token of the user's account to the web server in a HTTP POST data request. The web server receives this request, gets the Google email address from the token, and then checks if it is a verified user. If the user is verified to modify the systems table, it keeps a copy of this truth in the user's session information. The web server sends data back to the user with the result of the verification. If they are verified, the page reloads, and the user has access to the table. They are able to modify, add, and delete entries to the systems table.

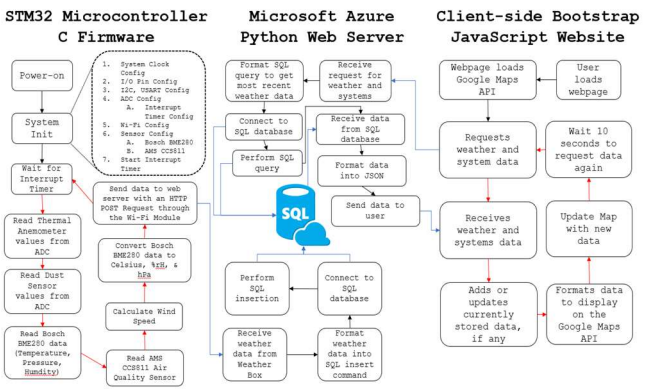

Figure 1: Software Flowchart of Weather Box

The Google Maps API allows for the creation of markers on the map with any data. These markers consist of a "latlng" object, objects from the Google Maps API that store latitude and longitude and a block of text that is formatted with HTML. When a user accesses the website, a function generates a set of "latlng" objects which are used to represent each of the weather boxes retrieved from the server. These points are combined with the data of each respective Weather Box formatted using HTML. Each marker on the map will specify the package's system id, the latitude and longitude, and the last stored data readings on the web server. Clicking on a marker will reveal all the information about the system to the user. The system data readings are updated by a JavaScript function that runs every ten seconds. The map updates the system information every minute.

Users may select between having all weather data for each system displayed in a marker on the map or they may select the individual data types. If a user selects the individual data types the website will render the data as a heatmap in addition to the discrete markers. The weather data heatmap is rendered using the Google Maps API's built in heatmap functionality. When a user accesses the website, a function generates a set of "latlng" objects which are used to represent each of the weather boxes retrieved from the server. The heatmap is initialized with all points having a null weight as the user has not chosen what they want to have displayed. When the user chooses the weather data they want shown, a function then takes the set of weather data that was most recently retrieved form the webserver, scales the data, and then sets the weights for each "latlng" object. In addition to using the data points gathered from the weather box systems, data points are

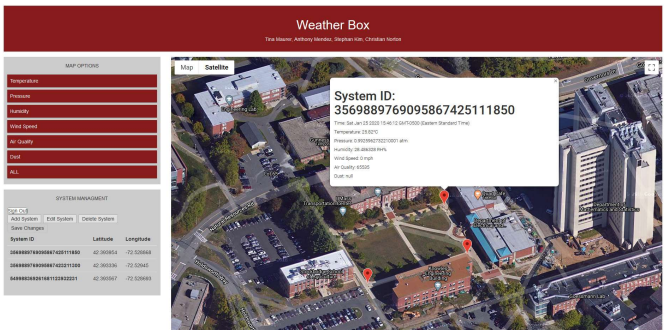

Figure 3: Weather Box Website

interpolated between the systems as well. This is achieved using a function that takes the set of "latlng" objects and weights from the heatmap and generating the averages between all combinations of points. These new objects are then added to the data set of the heatmap. Like the markers, the heatmap will update and interpolate every time new weather data is retrieved or if a system is added or removed.

III. The Product

A. Product Overview

Our product, depicted in a product sketch in Figure 4, consists of two circuit boards enclosed in a 3D printed mechanical enclosure, with a website displaying weather data. The Software Flowchart in Figure 2 describes what happens in the system. Starting with the Weather Box microcontroller enclosure. It handles powering on the device, retrieving the data, and then sending that data to the web server. The sensor enclosure handles taking measurements with the sensors onboard. This data is sent to the microcontroller enclosure. The sensor data processor and mapper handles storing the received data into the database. The web server handles sending that stored data to the client. On the client's computer, it handles placing the received data points on a map and interpolating the data.

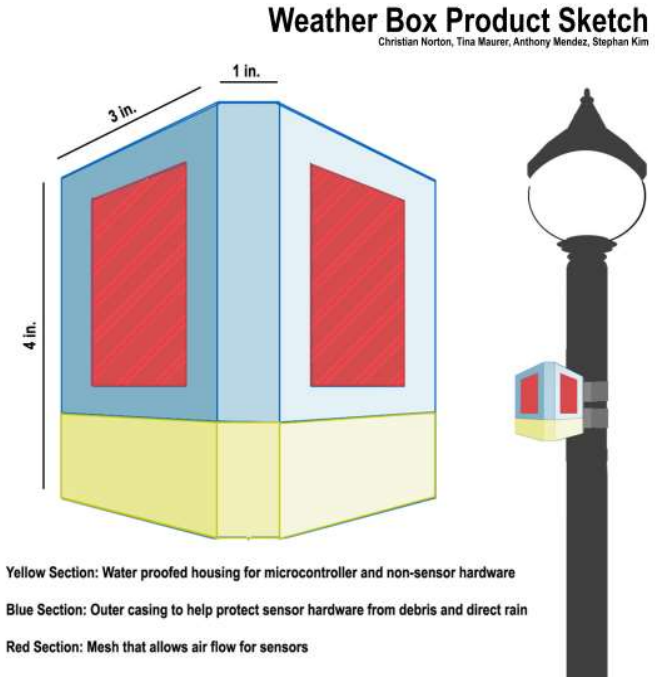

Figure 4: Weather Box Product Sketch

B. Electronic Hardware Component

The electronic hardware component for Weather Box consists of two printed circuit boards, one to house all weather sensors and one to house the microcontroller and other computing units. During the initial hardware design, we first found sensors according to the types of measurements we wanted to obtain and then chose them based on cost, size, and peak power consumption. Once the sensors were chosen, we chose the microcontroller accordingly. From prior experience,

an STM32 microcontroller was a clear choice as it is straightforward to develop firmware for as well as relatively simple to develop hardware designs with. After we had chosen the Wi-Fi as well, all major electronic blocks were chosen, and we could create a detailed hardware design.

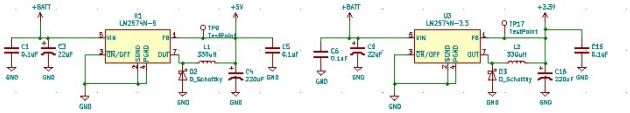

Figure 5: Switching Regulator Power Circuitry

The first major design components after the sensors and external electronics were selected was the power circuitry. From all datasheets of components in the system, we concluded that the necessary rail voltages were +3.3V, +5V, and +9 to +12V. As discussed in our *Design* section, we had previously used linear LDO regulators in our prototype to create those lines from a AA battery stack, despite the high power consumption. However, we moved to using switching voltage regulators, shown in Figure 5, to increase the power efficiency of our product after MDR when we were moving from a working prototype to a well-functioning product.

After creating a functioning breadboard prototype, the next design component was to get rid of the microcontroller development board. The STM Nucleo board we were using contained all the peripheral hardware to flash and run the firmware successfully on the microcontroller using a micro-USB connector. When attempting to flash and run our program using just the MCU chip, we chose an ST-Link V2 connector to flash the chip. Using the datasheets and other documentation provided from ST Microelectronics, we successfully implemented the necessary peripheral hardware to use the microcontroller without the development board. This consisted of a 20-pin connector to the ST-Link with filter capacitors from the supply lines to GND. The signals of SWDIO, SWCLK, and NRST provided the data line, clock signal, and reset signal, respectively.

Though the entire Weather Box prototype was contained in one physical system, we decided to split the product into two separate printed circuit boards. The main board contains the power regulation circuitry, microcontroller circuitry, and Wi-Fi module while the sensor board contains all sensors and their required peripheral circuitry.

We made the decision to make two boards to protect all non-sensor electronics. Since each Weather Box system would be set up outside, we wanted to protect the electronics that did not need to be exposed to weather, while exposing sensors to obtain the desired weather data.

We completed all schematic and layout design for both Weather Box printed circuit boards using KiCAD software. As we got various sensors working on our breadboard, the corresponding section of the system would be added to the master schematic. To connect the two boards, we used two headers as shown in Figure 6, with one being a connector for voltage supply signals and the other being for data signals. We chose to keep the connectors separate as well as separating the analog and digital data signals by a ground pin to minimize the amount of noise in our data signals. Some other design considerations we implemented in the boards were 0.1uF filter capacitors on all voltage supply signals to control noise and reset buttons for both the Wi-Fi module and microcontroller.

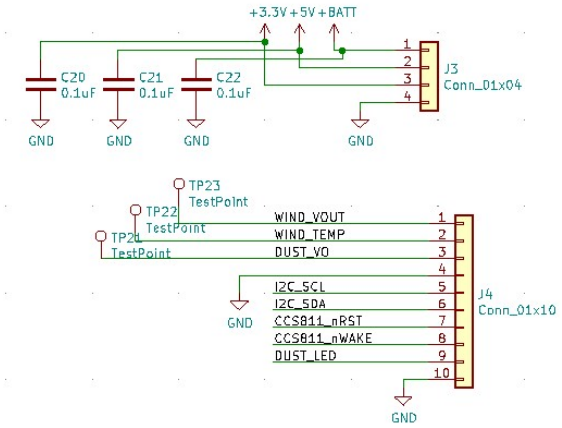

Figure 6: PCB Power and Data Signal Connectors Schematic

When completing layout for each PCB, we had to pay close attention to the placement and orientation of the connectors for both power and data signals so that the boards would fit in the designed mechanical enclosure as intended. We also had to carefully layout the power circuitry for each switching regulator as noise can sometimes be a problem with these types of integrated circuits. Both boards ended up being four layers, with one of the middle layers being a ground plane to simplify the routing of the boards. Shown in Figures 7 and 8 are the layouts of the main PCB and sensor PCB, respectively.

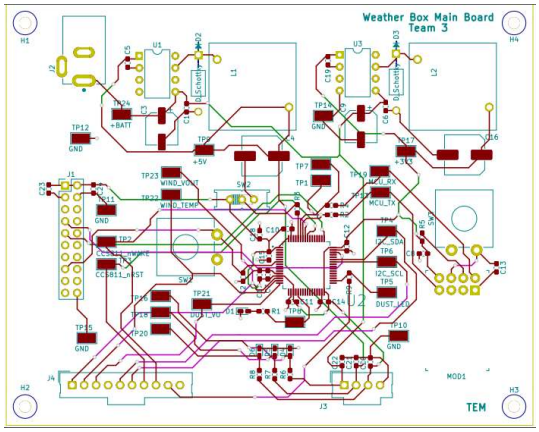

Figure 7: Main PCB Layout

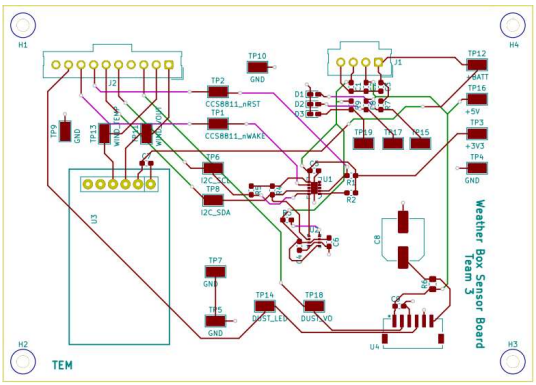

Figure 8: Sensor PCB Layout

In addition, Figures 12 and 13 in *Appendix F* show the 3D model of the PCB from the KiCAD layout software.

Our boards were manufactured from PCB Way and we hand-soldered all components onto the boards to populate both boards. Most of the components were surface mount due to the size difference between the surface mount and through hole versions of the components. During population, extra care was taken when soldering the microcontroller chip as we had previously learned that the chip can get damaged from being exposed to the heat of an iron for too long. We used microscopes from M5 to verify solder joints and performed continuity tests with multimeters to ensure successful connections in cases where we could not visibly observe the joint.

One of the priorities when designing these boards was the ability to test and debug them, as we wanted to be able to address any problem that could occur with the boards. For that reason, a test point was placed on every signal (both supply voltage and data signal) so that we may measure and monitor the signal to check functionality and debug problems with any part of the board. In addition, I put multiple ground test points on each board to make it easy to probe ground when needed. I also placed indicator LEDs on each power line so that we could easily determine if a power line were present without having to probe the test point, thus making the testing process more efficient. Before powering either of the printed circuit boards, we performed a few preliminary tests to rule out any power issues that may break either board upon power up. First, we performed multimeter continuity tests to first ensure that no supply lines were shorted to each other or to ground. We then visually checked all pins on the microcontroller with a microscope to make sure no pins were shorted there. Upon powering the boards, all indicator LEDs were on, showing us that our power lines were working as expected. At first, we kept the main and sensor PCB disconnected so that we could test one part of the system at a time. We successfully connected the main PCB to Wi-Fi and could successfully flash the board with our firmware. After testing the main board, we connected the sensor board and checked the website data to see that we got the correct sensor data. To check this, we simply compared it to the confirmed correct data we got with our prototype. Shown in Figures 9 and 10 are the pictures of our working boards turned on and sitting in their 3D printed mechanical enclosures.

C. *Product Functionality*

By CDR almost every part of our block diagram in Figure 1 functioned as intended. Both custom PCBs worked as expected, with us being able to confirm the power circuit accurately regulating the battery stack voltage. We could confirm that our anemometer, temperature, pressure, and humidity measurements were accurate since we had other tools that took the same measurements and compared our sensor measurements to that of the other tools. We could not get an additional dust and air quality sensor to prove that our measurements were accurate. However, we got numbers from the sensors that we would expect from certain conditions. For example, when covering the dust sensor to emulate an obscene amount of dust, our reading jumped from zero to max. Also, we could not exactly test the accuracy of the sensor for any other measurements. The ams air quality was not perfect either. The lowest the eCO2 the air quality sensor could detect was 400 ppm. Our microcontroller and Wi-Fi module both functioned since we saw that our system connected to our hotspot, and on our website, we could read the sensor data in the JSON format, which is how we programmed our microcontroller to format the data.

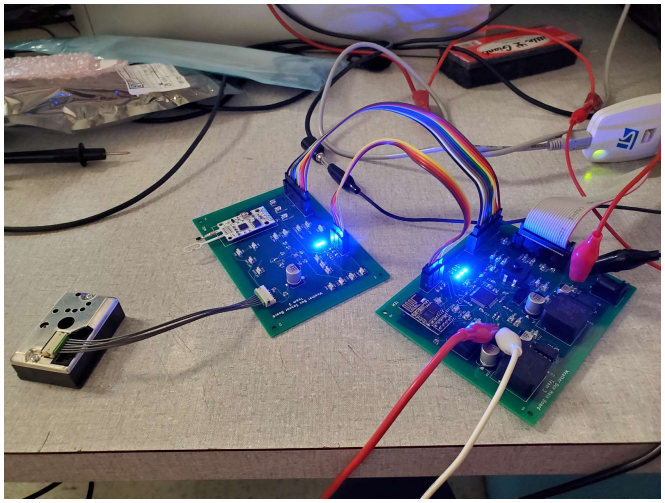

Figure 9: Main and Sensor Custom PCBs

Figure 10: Main and Sensor PCB with enclosures

When the sensor packages send their data to the webserver, the webserver checks the data type before storing it in the database, preventing any malicious or bad data from being entered. We did have one issue with our website not functioning during our presentation, however, this problem was only due to our Microsoft Azure plan not providing enough computation time for our product.

The website allows users to select what weather data is displayed on the map at any given time and it updates as soon as the user clicks the button. The website is integrated with the Google Maps API so users can see the data points for all sensor packages as a set of markers. In addition to these markers when users select an individual weather measurement, it displays a heat map of the data. Also integrated with the website is the Google Sign-In API, allowing the website to verify authenticated users to modify

the addition, removal, or placement of the sensor packages on the map.

D. *Product Performance*

For MDR, we were able to meet the 95% confidence intervals for temperature, pressure, humidity, and wind measurements. We performed an endurance test as well and it lasted over 35 hours. This is discussed further in *Appendix C*. We were not able to properly test and take measurements for CDR or FPR. For CDR, there were multiple factors involved. Tina was working at her co-op at Bose, and the COVID-19 epidemic was affecting the manufacturers in China. While Tina was able to work on some electronic and mechanical design on the weeknights after work, she could not work on in-lab prototyping with the rest of the group until the weekends. Once the PCBs had arrived, there was very little time to populate the boards, and to perform tests. The PCBs worked and were able to successfully transmit data to the web server. The data received on the web server end looked valid, so we assumed the sensors worked perfectly fine due to our time constraints. We expect that if we had been able to populate two more sets of Weather Box systems, we could have created an entire map of collected weather data since the system we populated functioned as expected.

For FPR, any in-lab portions were terminated early because of the COVID-19 pandemic. The entire system, with the battery stack, weighed approximately 1.49 lbs, failing to meet the desired weight specification. This was mainly due to the battery stack weight as the system without the battery stack weighed about 0.94 lbs. We would have used the rest of the lab time to weatherproof the enclosure and investigated alternative battery and enclosure options to decrease the system weight. Additionally, this in-lab portion includes testing the dust and ams sensors for accuracy as well as leaving the Weather Box on overnight to see how consistent our readings were for those sensors. So, we were unable to test power consumption and battery life for our system. We also had one issue where we could not flash our code and keep it running after disconnecting the Weather Box from our computers, so we did not have the opportunity to debug that.

IV. Conclusion

We met our goals for both MDR, and CDR. We had to work many more hours programming the STM32 microcontroller in MDR than in CDR. Yet, this made prototyping of our custom PCBs more streamlined. We were sure of which pins to use on the microcontroller for layout and design. This gave us more time to develop other aspects of the project for CDR. We were able to design and populate working printed circuit boards for CDR, which could sense weather data and send that data to the server. On the web server side, we have the first iteration of a weather interpolation algorithm up and running. There were many options for an interpolation algorithm. For CDR, we went with inverse distance weighting. Both methods provided a decent result for the distance between the system packages [20]. We also implemented a registration and relocation protocol for sensor packages on the website. So, each time we add a new Weather Box system or need to change its location, we do not have to write SQL queries manually and can do it all from the website. For MDR, we were able to test the systems and show that it was sending reliable data to the web server, and the sensor package lasted over 35 hours. For CDR, we were not able to perform adequate testing due to COVID-19. However, we were able to verify the populated custom PCB worked perfectly.

Acknowledgment

We would like to acknowledge and thank our advisor, Professor Zink, for his continued support and assistance throughout SDP. We would also like to acknowledge and thank our evaluators, Professor Tessier and Professor Pishro-Nik for their constructive criticism and feedback as well as Professor Holcomb for stepping in as an evaluator for CDR. We would also like to extend our thanks to the M5 staff allowing us to use their tools and components. We would like to acknowledge and thank the Provost office for aiding in the collection of additional funds as well as Apoorva Bajaj for assisting us in the initial research. Finally, we would like to acknowledge and thank our families and friends.

References

[1] J. Erdman, “The Most Weather-Delayed Major Airports in the U.S.,” *The Weather Channel*, 21-Nov-2018. [Online]. Available: https://weather.com/travel/news/2018-11-19-most-weather-delayed-major-us-airports. [Accessed: 18-Dec-2019].

[2] T. A. Press, “Flight delays are costing airlines serious money,” *Mashable*, 10-Dec-2014. [Online]. Available: https://mashable.com/2014/12/10/cost-of-delayed-flights/. [Accessed: 18-Dec-2019].

[3] Y. Liu, “Power Grid Operation Weather Security: NCAR Research Applications Laboratory,” *RAL*, 2019. [Online]. Available: https://ral.ucar.edu/projects/power-grid-operation-weather-security#nogo. [Accessed: 18-Dec-2019].

[4] K. H. Lacommare and J. H. Eto, “Understanding the cost of power interruptions to U.S. electricity consumers,” *ERNEST ORLANDO LAWRENCE BERKELEY NATIONAL LABORATORY*, Jan. 2004.

[5] M. P. J. Ashby and L. Tompson, “Does a good cop really never get wet? The impact of weather on stop and frisk,” Jun. 2018.

[6] STMicroelectronics. (n.d.). STM32L073RZ - STMicroelectronics. [online] Available at: https://www.st.com/en/microcontrollers-microprocessors/stm32l073rz.html [Accessed 21 Jan. 2020].

[7] Modern Device. (n.d.). Wind Sensor Rev. P - Low Cost Anemometer | Modern Device. [online] Available at: https://moderndevice.com/product/wind-sensor-rev-p/ [Accessed 21 Jan. 2020].

[8] Bosch Sensortec. (n.d.). BME280. [online] Available at: https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/ [Accessed 21 Jan. 2020].

[9] Sparkfun.com. (n.d.). GP2Y1010AU0F Compact Optical Dust Sensor. [online] Available at: http://www.sparkfun.com/datasheets/Sensors/gp2y1010au_e.pdf [Accessed 21 Jan. 2020].

[10] Ams.com. (n.d.). CCS811 | ams. [online] Available at: https://ams.com/ccs811 [Accessed 21 Jan. 2020].

[11] Cdn.sparkfun.com. (n.d.). ESP8266 Module (WRL-13678). [online] Available at: https://cdn.sparkfun.com/datasheets/Wireless/WiFi/ESP8266ModuleV1.pdf [Accessed 21 Jan. 2020].

[12] Azure.microsoft.com. (n.d.). Cloud Computing Services | Microsoft Azure. [online] Available at: https://azure.microsoft.com/en-us/ [Accessed 21 Jan. 2020].

[13] US Department of Commerce and Noaa, “Radiosonde Observation,” *National Weather Service*, 14-Aug-2017. [Online]. Available: https://www.weather.gov/upperair/factsheet. [Accessed: 23-Jan-2020].

[14] Telaire, SMART Dust Sensor SM-PWM-01C Appl. Note, pp.4.
[15] Sharp, "Compact Optical Dust Sensor," GP2Y1010AU0F datasheet, Dec. 2006 [Accessed Jan. 2020].
[16] "ZX 300 Onshore Wind Lidar for remote wind speeds and TI," ZX Lidars. [Online]. Available: https://www.zxlidars.com/wind-lidars/zx-300/. [Accessed: 25-Jan-2020].
[17] M. LaFay, "Tips for Flying Your Drone in Sub-Optimal Weather Conditions," dummies. [Online]. Available: https://www.dummies.com/consumer-electronics/drones/tips-for-flying-your-drone-in-sub-optimal-weather-conditions/. [Accessed: 25-Jan-2020].
[18] I. Lee, "Tips for Flying a Drone in Hot Weather Conditions this Summer," UAV Coach, 20-Apr-2019. [Online]. Available: https://uavcoach.com/fly-drone-high-temperatures/. [Accessed: 25-Jan-2020].
[19] B. Schweber, "When should I use an LDO versus a switching regulator?," *Power Electronic Tips*, 23-December-2016. [Online]. Available: https://www.powerelectronictips.com/use-ldo-versus-switching-regulator-faq/
[20] "Interpolation methods for climate data ," "https://www.snap.uaf.edu/attachments/Interpolation_methods_for_climate_data.pdf" De Bilt, Wilhelminalaan 10 , 2009.
[21] "Welcome to Flask," Flask Documentation (1.1.x). [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/. [Accessed: 27-Apr-2020].
[22] "Google Maps Platform Documentation," Google Maps Platform. [Online]. Available: https://developers.google.com/maps/documentation. [Accessed: 27-Apr-2020].
[23] "Integrating Google Sign-In into your web app," Google Sign-In for Websites. [Online]. Available: https://developers.google.com/identity/sign-in/web/sign-in. [Accessed: 27-Apr-2020].

APPENDIX

A. *Design Alternatives*

When looking into the subject of monitoring weather at a micro-scale, there is a range of products that satisfy some of the individual aspects that one would be looking for, like portability, scale, resolution, or data storage and analysis. However, many of these products do not cover all the aspects that someone would want out of them. Our design process focused on trying to deliver a product that had as many of these aspects as possible within our price range.

One of the early designs that we had conceived used LIDAR (Light Detection and Ranging) to measure the rate that particles in the air moved in order to create a map of wind speed, rainfall and dust density. This would have been used in conjunction with other sensors in order to generate a full map of weather conditions. However, this design came with many draw backs. LIDAR systems are not cheap and to get a basic unit that had the resolution needed for such calculations was out of our budget. In addition to this, systems that already implement LIDAR for similar applications, like those used in windfarms for measuring wind speed, often come in larger packages, like the ZX 300 Lidar Weather Station for example is a cubic meter in size and weighs 55kg [16]. These two major restraints were the reason this design was ultimately scrapped. Unfortunately, because of this, it was clear that having multiple three-dimensional maps of wind was far out of our price range.

We had looked at many existing sensor packages at different price ranges and noticed that although they had many of the sensors that we were looking for, they lacked the ability to access the data from alternative locations. This was the main criteria we wanted to have implemented in our project. Because of this many of our design choices came down to budgeting our sensor packages with our power systems and communication systems. This is what ultimately aided us in making the major design choices in our project. Some sensors were dropped in favor of having higher quality sensors and communication systems. We specifically chose Wi-Fi to communicate since the Weather Boxes would be tested around campus, therefore we were essentially guaranteed a connection all over campus, which also means less power consumed than if we were to use cellular data because cellular signal is not as strong on certain parts of campus. For our microcontroller, we chose the STM32 because Tina had experience using it in her internship and that it has good documentation for development [6]. It was this process of optimization that lead to our final design.

For the most part, our design process was swapping out almost equivalent sensors in order to optimize the cost of production and operation of the sensor packages with the necessary power systems to run them. For example, one of the sensors that we had swapped because the power requirement was the Amphenol Advanced Sensors SM-PWM-01C dust sensor. This sensor required more than four times the power of the SHARP GP2Y1010AU0F dust sensor for comparable performance [14][15]. In other cases, sensors or parts were changed due to the cost of them being too much and putting us slightly out of the given range we wanted for each sensor. This happened a few times while we were deciding the best WIFI module to use for our project.

B. *Technical Standards*

Some examples of standardized hardware and software that we used include: Wi-Fi, HTTP, ANSI C, Python, JavaScript/Jquery, STM32 MCU, and electronic schematic symbols.

The IEEE standard we used in hardware design was IEEE 315-1975: IEEE Standard for Graphic Symbols for Electrical and Electronics Diagrams. This standardizes are symbols and their reference designators used in our schematics for the printed circuit boards. While it governs the more well-known components like R for resistor, C for capacitor, and L for inductor with their respective symbol drawings, the standard details all standard component symbols for schematics. For example, a connector would have the J reference designator and an integrated circuit would have the U reference designator. Through the PCB CAD tool we used, KiCAD, we were able to easily adhere to this standard. This allows anyone familiar with the standard guidelines to be able to interpret our hardware schematics.

For Wi-Fi the IEEE standard that it adheres to is IEEE 802.11 which is a part of the IEEE 802 standard for local area network (LAN) protocols, and it specifies the different protocols for implementing wireless computer communication in various frequencies, most notably 2.4GHz and 5GHz for most Wi-Fi enabled devices. This also falls under the FCC rules 47 CFR Part 18 for the legal operating ranges for radio frequency (RF) devices.

For ANSI C, Javascript/JQuery, and Python, we utilized the IEEE 754 standard which specifies how a language handles

floating point arithmetic since we needed to use floating point arithmetic in our code. Since many machines are created with IEEE 754, the floats in our code were mapped to this standard. Our STM32 microcontroller also follows this standard since ARM microcontrollers follow this standard when calculating floating point values.

HTTP does not follow any IEEE standard, but instead follows RFC 7231 which describes the semantics of how HTTP messages are expressed. We utilized this protocol and standard when we formatted an HTTP message to send to our website via Wi-Fi.

C. *Testing Methods*

Our testing methods for MDR consisted of starting a Wi-Fi hotspot on a laptop or phone. The Weather Boxes would connect to this Wi-Fi hotspot, and to send their data every 10 seconds. One this setup is done, if we want to test the wind speed, we grabbed a fan and let it run over the wind sensor for about 20 minutes. After, we turn off the fan and perform another 20-minute session with no wind. To test the temperature sensor and overall operating temperature, we went outside for about an hour where it was about 5 degrees Celsius. The Weather Box performed well and did not suffer from any issues.

In order to analyze our results from the various testing sessions, we calculated confidence intervals of 95% for each of the testing sessions. This displayed the accuracy of our sensor packages for each session. We completed both a few shorter tests (around 20 minutes) and a longer test to show that we met the specification for our battery life. This shows that our project can both accurately measure weather data as well as be able to meet our power/battery specification. Below is the test data we collected for each test session, shown in Table 4. For session 1, we had 121 data points for system 1 and 115 for system 2. For session 2, we had 120 points for system 1 and 116 for system 2. Furthermore, in Figure 11, we show the battery life testing of one of the weather boxes. During this task, one of the sensor packages was able to take measurements every 10 seconds for approximately 35.5 hours.

Testing for CDR was not executed as originally planned. When the orders were made for the custom PCBs, Chinese manufacturers were being hit hard by the COVID-19 pandemic and ordering from manufacturers in the United States was not within our project budget. The PCBs were delayed by two weeks, and by the time they arrived, we had limited time before our CDR presentation. After populating the boards, we were able to test that the power circuit, and the sensors worked. For more detail on the hardware test procedure, please see the *Electronic Hardware Component* in the *Product* section. With the boards functioning as expected, we were able to do some sensor data transmissions to the web server, and have a working prototype for CDR. We had planned to perform tests after CDR and use this data as a starting point to refine our product. However, in-lab portions were cancelled in March, so this portion of the project, was finished.

System, Test Session	Measurement Type	Confidence Interval
System 1, Session 1	Temperature	[24.37, 24.40]
	Wind Speed	[1.0695e-5, 8.889e-5]
	Air Pressure	[1006.33, 1006,34]
	Humidity	[22.85, 22.88]
System 2, Session 1	Temperature	[24.063, 24.0904]
	Wind Speed	[3.446e-5, 1.0722e-4]
	Air Pressure	[1006.69, 1006.703]
	Humidity	[22.62, 22.71]
System 1, Session 2	Temperature	[23.51, 23.54]
	Wind Speed	[2.586, 2.663]
	Air Pressure	[1005.997, 1006.02]
	Humidity	[24.78, 24.83]
System 2, Session 2	Temperature	[24.49, 24.51]
	Wind Speed	[0.01371, 0.01785]
	Air Pressure	[1006.34, 1006.36]
	Humidity	[22.99, 23.04]

*NOTE: Temperature (˚C), Wind Speed (mph), Air Pressure (HPa), Humidity (%)

Table 2: 95% Confidence Intervals

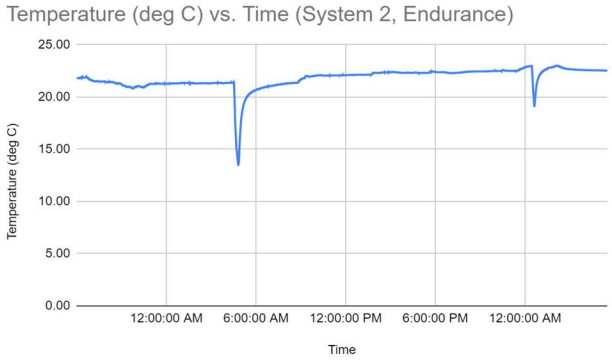

Figure 11: Endurance Testing for System 2

D. *Team Organization*

Anthony was the Software Lead since he has the most work experience in Software Engineering. Tina was the Hardware Lead since she is the only Electrical Engineer in the group, but also because she has research and industry hardware design work experience under her belt. Stephan oversaw most the firmware but received assistance from Anthony in programming some modules. Christian oversaw the web server and website but received assistance from Anthony in programming as well.

Overall, the team worked well together. We met our MDR and CDR deadlines, have a working and populated PCB sensor package, and many other prototyped sensor packages.

Relative to this project, each team member has their own expertise. Anthony has a good understanding of the overall system and how they should communicate with each other. Tina's expertise is in anything hardware related: power systems, components, PCB design, and enclosure design. Stephan has overseen most of the firmware and programmed the Wi-Fi module and the wind sensor with the ADC. Christian has programmed parts of the webserver and website.

While everyone has their own respective domains, we always assist one another when one is falling behind on their work. This semester, Christian had a surgery and was unavailable for about a month in the fall semester. So, we had to fill in for him and complete some of his website and webserver responsibilities.

We had a good communication system set up.

E. *Beyond the Classroom*

A. *Anthony*

Part of my responsibility was setting up a few crucial parts of this project. The first was setting up a Microsoft Azure project to host our website. I used Azure before at an internship but never dealt with creation of projects or resources, which is something I had to learn. Not only that, but I also had to learn how to setup the Python Flask website, the Google Cloud account to access the Google Maps JavaScript API and set up the STM32 project in CubeIDE. I also set up the Google Sign-In API and built the systems management table from scratch. I did most of the programming on the Bosch BME280 and ams CCS811 sensor.

In connection with my current experience as a professional, there are a few domains. In setting up the Microsoft Azure project, I used Azure in my time with Citrix as a Software Engineering Intern, so my experience there helped me with this project. I have experience programming Microcontrollers from class and my time with Zebra Technologies. Outside of that, I'm much more aware of the work that goes into make any electronic product and will be taking that knowledge with me when I start my job at Google.

B. Stephan

My main responsibility for this project is the firmware. So far, I have written the firmware for our ADC, anemometer, dust, and Wi-Fi module. This project has allowed me to further develop my skills with embedded systems and C, which I first developed in our Computer Systems Engineering class. I also had to learn the AT Command set for modems in order to program our Wi-Fi module. The main skill that I have learned is to create code that does not exist elsewhere since I could not find any other project that utilizes the exact same hardware configuration. For this project, I essentially had to write my own driver to program our Wi-Fi module via our microcontroller by serially sending commands specific to our project. Some helpful resources have been the different datasheets as well as looking at some examples utilizing similar hardware as us.

I see connections with this project and my potential future career since this project has exercised my skills in both writing firmware and debugging it. By writing code that doesn't exist, I have exercised my skills not only as a programmer, but also as an engineer by creating something that does not exist.

C. Tina

As the hardware lead of this project, it has been my responsibility to both make all major hardware decisions for the group and design our custom PCB. Thus far, this project has given me the opportunity to further develop my board-level design skills as well as learn how to take into consideration both software and mechanical constraints when making hardware decisions. I designed, breadboard prototyped, populated, and tested our two printed circuit boards as well as designed the mechanical enclosure for them. Through working with my teammates to determine the hardware necessary to successfully transmit the sensor data and then designing the printed circuit boards, I have been able to strengthen my skills as a teammate and an electrical engineer.

I have already been able to see connections between this project and my professional life. At my internship this past summer at Globus Medical I was able to use these types of board level design skills to work on a navigation board, ensuring that my board handled issues such as power line placement, sensitivity and ratings of components used, and mechanical considerations of size and cabling. In addition, being able to quickly prototype and test my ideas has been especially helpful this semester in my co-op at Bose Corporation, where I was able to take my lab prototype and debug skills to work in a unique makerspace where my job was to ideate and prototype new concepts for the company. I am grateful for the opportunities and experience that SDP20 has afforded me and I believe that the skills I have learned already have and will greatly advance me in my career in graduate school and industry!

D. Christian

My main responsibility for our project is developing the website. This involved writing the HTML and CSS that create what users see and interact with as well as the functionality involved in rendering and updating the map, data markers, and heatmap. I have had a little experience with basic HTML and CSS programming before, but this project has really helped me flush out my understanding. I had never used the Google Maps API before this project and it has been great experience using it to flush out our application. I have also learned a lot of JavaScript and python for this project. Assisting Anthony in developing some of the back has really taught me a lot about the Microsoft Azure web services.

I have also learned a lot about implementing sensors and custom hardware with the microcontroller. Spending time in the lab while helping trouble shooting the micro-controller, analog digital converter, and sensors has really flushed out my ability to use the equipment available in lab.

As a professional, as much as I do enjoy the firmware and hardware side of the project, I mainly see myself working in a software development. I think many of the skills I have learned will apply to that field well.

F. Additional Hardware Documentation

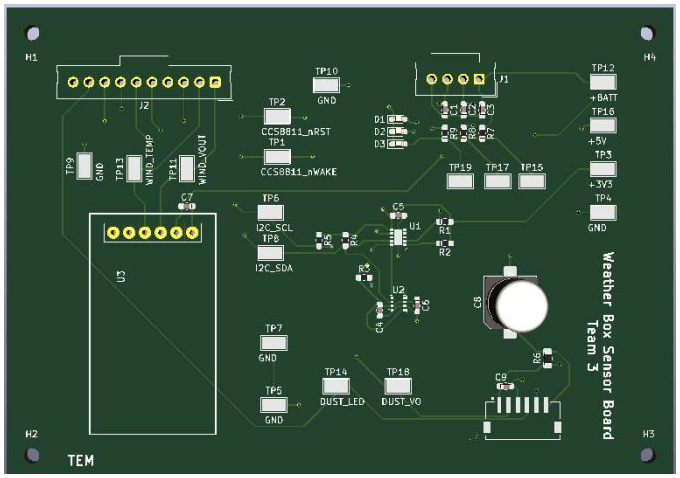

Figure 12: Weather Box Sensor PCB

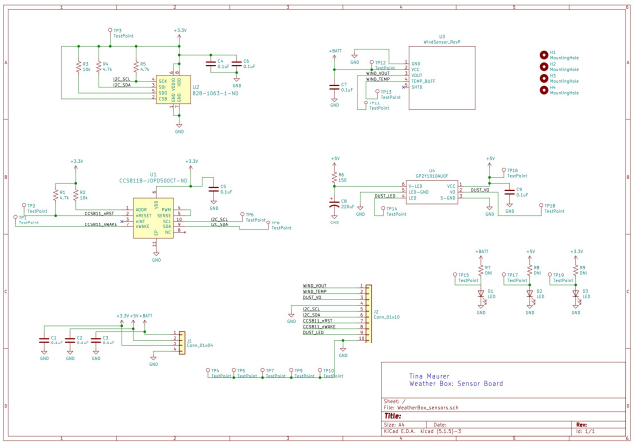

Figure 14: Weather Box Sensor PCB Schematic

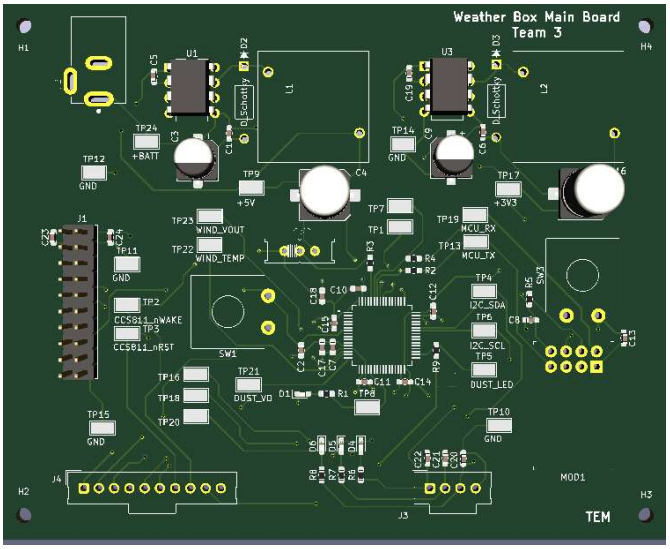

Figure 13: Weather Box Main PCB

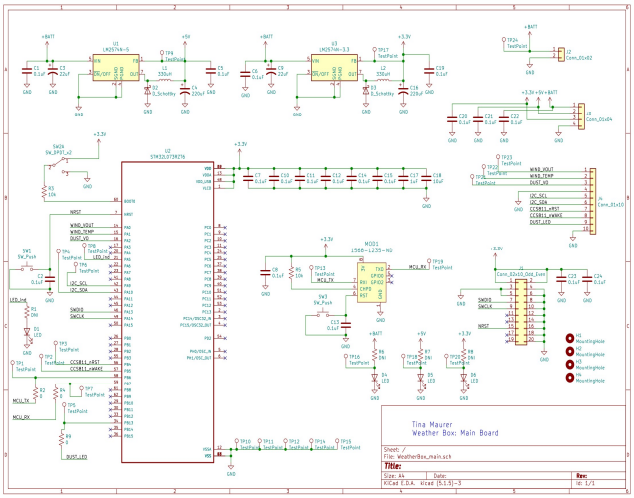

Figure 15: Weather Box Main PCB Schematic