EE 41440, University of Notre Dame
Forest Fire Prediction
Final Report

May 8, 2023

Authors:
Joseph Canfield
Verlee Richey
Kailee Saunders
Ed Stifter
Mitchell Turner

1. Table of Contents

2. Introduction

The Great Fire of 1910 took out more than three million acres across Idaho and Montana. It killed 84 people and injured many more. Luckily, there was enough warning for most of the women and children to escape. While these great fires are devastating to the land and its people, forest fires are nature's way of clearing out the dead leaves and branches that can overtake the forest floors. Over the last century, people have misconstrued this importance, and the public tends to see forest fires as an inherently bad thing. The result has been a buildup of underbrush that is now more prone to catching fire than before. Along the West Coast and the Inland Northwest specifically, wildfires have become fairly out of control over the last few years. This is mostly due to the forest mismanagement previously mentioned. People have been trying to prevent fires despite them being natural to large forest environments. In these regions, the forest undergrowth, consisting of bushes, grass, and small trees, has been able to grow in the absence of frequent fires, increasing the probability of huge forest fires that can cover upwards of 300,000 acres. With this buildup, forest fires become more dangerous and more likely. Therefore, there needs to be a way to warn people of potential danger, as they did leading up to the Great Fire of 1910.

To begin thinking about how to warn the public, one must first think about the conditions necessary for a forest fire to occur. First and foremost, forest fires require a dry environment and an ignition source. The probability of ignition largely depends on humidity and temperature. Some ignitions are naturally occurring, such as a lightning strike, but most ignitions are caused by humans, such as campfires and cigarettes. Once there is an ignition source, the probability the fire will spread is dependent on

temperature, humidity, and wind speed. The current measurements as well as trends are both important for calculating this probability. While the factors are relatively straightforward, the main cause of danger in regard to forest fires is the lack of detection. Those that live in nearby areas and would be in danger due to fires need to be given ample notice to protect themselves and their property. Keeping track of trends can also help advise firefighters as to when to perform a controlled burn. While forest fires are natural and healthy for foliage, they can also cause a lot of damage if they become uncontrolled. This is why early predictions are necessary.

Our proposed solution comes in the form of many weather stations placed around high risk areas, i.e. areas where forest fires are most prevalent. These sensors send signals to a centralized station that analyzes the information. A probability calculation is done that correlates temperature and humidity measurements with ignition probability and also overall trends that are correlated with large fires. Obtaining the parameters for this calculation is accomplished by sensing the key indicators of an environment and conditions that will likely cause a forest fire, keeping everyone ahead of the curve. Each weather station will have the capabilities to measure temperature, air pressure, humidity, wind speed, and wind direction. The weather station communicates with the receiver at the centralized station over LoRa. LoRa is a public frequency band at about 915 mHz that was chosen due to its extensive documentation and popularity among hobbyists. In practice, government frequencies can be used as agencies like the US Forest Service will likely be the biggest customer for our product. We also only have one weather station for the project demo, but in reality many would be needed to predict forest fires across a region.

The weather station itself is built atop a PVC pipe structure, since the station will need to be high enough to be out of reach of animals and clear of large trees or other structures that could interfere with measurements. The wind vane and anemometer are exposed to the elements atop arms connected to the center station. The rest of the sensors are encased in a ventilated enclosure to protect them. The entire station will be solar charged to avoid frequent maintenance.

The weather station we designed can accurately detect temperature, relative humidity, air pressure, and wind speed up to two decimal places of precision as well as wind direction in all eight cardinal and ordinal directions. While working on the project, we realized that wind direction and air pressure were actually insignificant factors in forest fire probabilities. That being said, having the measurements can still be insightful. Wind direction in particular can help residents determine the likelihood of the fire heading towards their region next. Air pressure can also be an indicator of changing weather patterns, with high pressure usually indicating stable weather conditions. Our design successfully measures, communicates, and reports to our webpage, so it meets all our expectations in that sense. Originally, we wanted to include an alert system that would notify the appropriate residents if a forest fire became likely, but the cost of a system like this didn't make sense for our demonstration. Despite not having an alert system, the website still displays the likelihood of a forest fire occurring as well as current and historical data. This still allows residents to be aware of current forest fire risk. That being said, the alert system is something that should be implemented in a further iteration of the product.

3. Detailed System Requirements

Our design consists of a weather station, a receiver, a web server, and a website. The weather station is the physical component that will actually be placed in the field. It records all the data and sends it to the receiver. The receiver then receives the data over LoRa and repackages it to send it to the web server over WiFi. The web server then interprets the data. In an actual implementation, the receiver would receive data from multiple weather stations and send it all to the server to be analyzed together. Then, this information is displayed on a website. The website features current measurements, weather trends, and a forest fire probability indicator.

The portion of the design in the field is the weather station. It needs to be able to detect surface temperature, air pressure, wind speed, wind direction, and relative humidity. These measurements need to be accurate to the two decimal places. The sensors for temperature, pressure, and humidity must be encased in a weatherproof enclosure with ample ventilation so as not to obscure measurements. The anemometer for wind speed and the wind vane for wind direction must be attached on the outer parts of the station so they are exposed to the elements. Therefore, these aspects must be weatherproof in their own right. The weather station must also have the capability to communicate over LoRa with the receiver. The weather station itself must have the capabilities to be solar powered.

To elaborate on the necessary enclosure, the station will be mostly encased in a waterproof enclosure and must be able to withstand temperatures consistent with electronic devices; i.e. -20 degrees to 80 degrees Celsius. In order to accurately detect

wind measurements, the device must be elevated in a large clearing. It will be installed atop a structure similar to those used in other weather detection devices. Ideally, the enclosure containing the sensors will be installed at least 33 feet high and at least 50 feet away from any tall vegetation. For the purpose of our project, we will only make the top of the structure where the station will be connected. In order to protect animals and the environment, the design is minimalistic and takes up as little space as possible.

The station will need to be solar-powered to ensure long-lasting life. Since the station will be installed in the middle of a forest, it is important for it to be self-sustainable. The battery and associated regulation will need to be capable of supplying 3.3V.

As previously stated in the introduction, we will be using LoRa frequency ranges for communication with the receiver rather than the government frequencies that would likely be used in actual implementation. For wireless communication, only one device will need to be supported as there is only one device detecting data. For future applications, multiple devices will need to be supported, but we will not focus on that for the scope of this project. We deemed an acceptable range to be 500 meters based on other field research. The weather station will use this communication to send the data to the receiver. The receiver will then use WiFi to send that data over to the web server. The web server will then analyze the data collected by the device and compare it to trends in the environment. The server will be referencing a table that contains information on the likelihood of forest fires. It will use the current measurements, trends, and database to make a prediction on the likelihood of a forest fire. In a real application, it would also have access to the locations of the weather stations. The web server will

then automatically upload the analysis to a website that can be accessed by people both inside and outside the communication range of the device.

Finally, the website will allow viewers to look at current statistics and trends over hours, days, or months. It will also feature a warning to the probability of a forest fire as well as resources for forest fire protection.

4. Detailed Project Description

4.1. System Theory of Operation

The system works by measuring weather data at a remote weather station, namely temperature, wind speed, wind direction, relative humidity, and air pressure. The wind speed is measured via a reed switch and the direction is measured using hall effect sensors. The temperature, humidity, and pressure are measured by two on-board sensors. This data is then sent via LoRa to a receiver at a centralized station. The receiver repackages the data and sends it over Wifi to the web server. The web server analyzes the data against a table that predicts forest fire trends and displays it to a website. The website shows current statistics, historic trends, and the likelihood of a forest fire occurring.

4.2. System Block Diagram

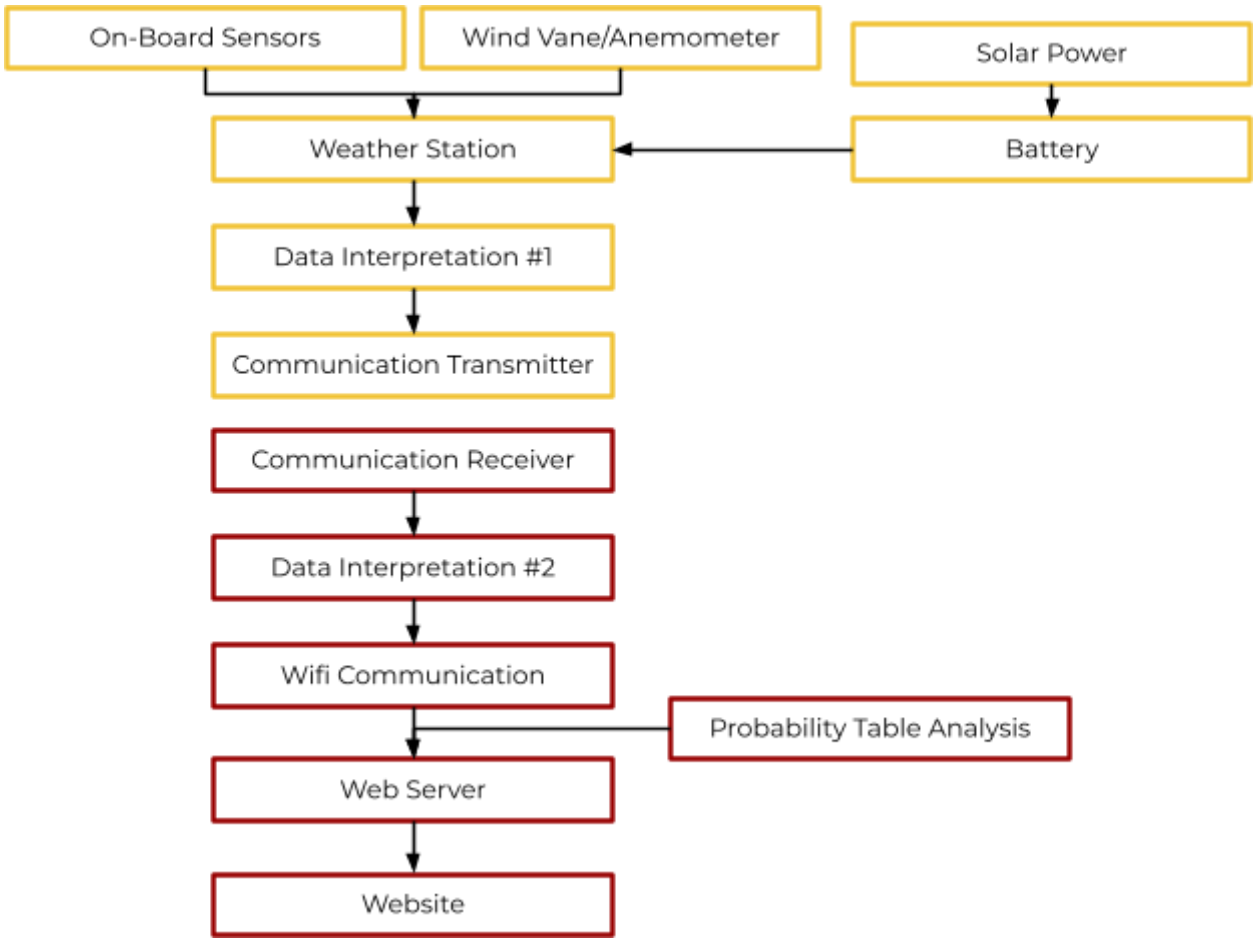


Figure 1. System Block Diagram

Figure 1 is a flowchart that outlines the major subsystems and how they integrate with each other. The yellow blocks are representative of the field components and the red blocks are representative of the local components. The three major systems are the weather station, the receiver, and the web server. A solar panel charges a battery, which powers the station. The station reads in data from the sensors, interprets it, and sends the data via LoRa to a receiver at a centralized location. The data is repackaged and sent over WiFi to a web server.

A table look-up system is used to estimate the risk of a forest fire based on previous research done by Texas A&M university. This information is then uploaded to a website that is available to the public.

4.3. Subsystem 1: On-board Sensors

One major subsystem of the weather station is the onboard sensors. These sensors must accurately detect temperature, humidity, and air pressure to the second decimal place. These sensors must be able to withstand temperatures consistent with electronic devices; i.e. -20 degrees to 80 degrees Celsius. These sensors will be encased in an enclosure, so there is no requirement for them to be weatherproofed. We aimed to find smaller sensors, so both the board and the enclosure sizes could be minimized. This also helped to minimize our cost. Since a real world application would require several weather stations across a region, it was important to make them cost effective. With the ESP32 microcontroller we chose, we needed the sensors to communicate through an I2C interface. Therefore, another requirement for the sensors was that they had a native I2C interface. The overall function of this system is to accurately detect all temperature, pressure, and humidity measurements.

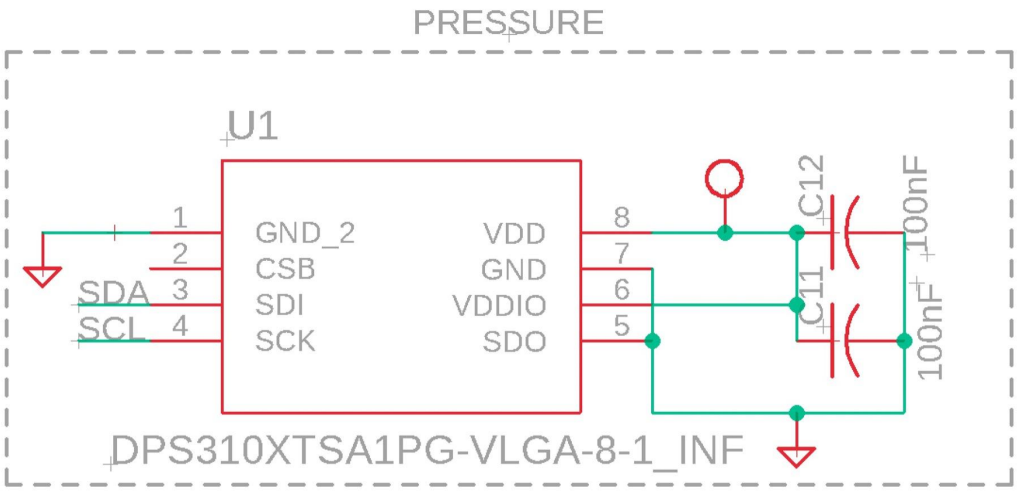


Figure 2. Schematic for Pressure Sensor

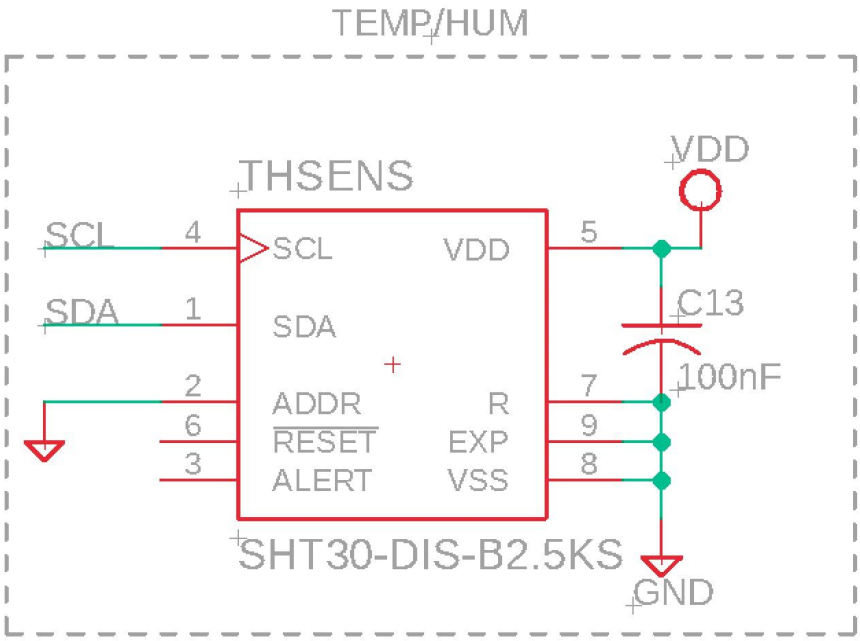


Figure 3. Schematic for Temperature and Humidity Sensor

Figures 2 and 3 show the schematics for the on-board sensors. The capacitors on the voltage sources are the bypass capacitors. In the board design, the capacitors are placed right next to their respective sensors because of the nature of bypass capacitors. While not pictured in the above schematics, the I2C interface includes pull-up resistors for its appropriate function.

We chose these specific sensors because of their native I2C interface and for being cost effective. As previously stated, it was important to minimize the costs of this design to make real world application economically feasible. We chose to use an I2C interface because it is easy to connect multiple sensors and eases the programming required. An I2C interface was also compatible with the ESP32 that we were already familiar with, making this type of interface an easy choice.

We were not able to test our subsystems individually due to the nature of our project. The sensors were too small to reasonably test them without assembling them onto our board. As the sensors themselves were not all that complex, we decided it was best to wait until we could get our board printed and assembled to begin any testing. Once the board was assembled, we used the serial monitor on VsCode to ensure the sensors were working properly.

4.4. Subsystem 2: Anemometer

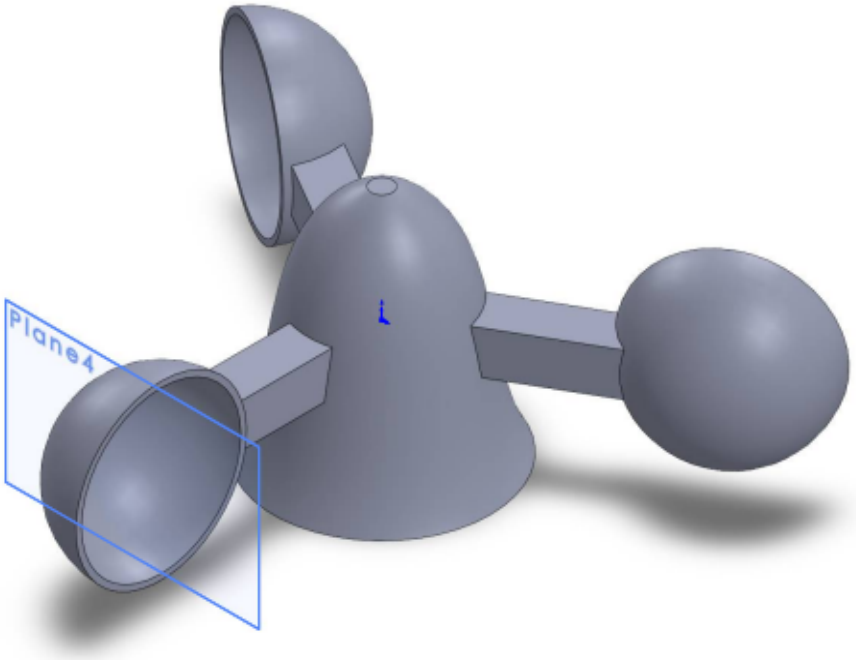


Figure 4. Anemometer Solidworks Design

The second major subsystem is the anemometer. The main requirement for this subsystem is that it accurately detects wind speeds. We decided to design our own anemometer instead of buying one off the shelf to increase the complexity of our project.

As the wind blows, the air is caught in the cups and causes the fixture to spin. The number of rotations per minute is measured using a reed switch and a magnet affixed to the spinning portion of the anemometer. The reed switch closes the circuit every time the magnet passes over it, allowing the ESP32 to

detect each rotation. The number of rotations per minute is then converted to miles per hour using a multiplier we determined by measuring the diameter of the anemometer.

We chose to use a reed switch rather than a hall-effect sensor because of its simplicity. Hall effect sensors would add complexity to our code and circuit design because they require that we interpret analog values rather than digital ones. We knew that hall-effect sensors were unavoidable for the wind vane, so to reduce the number of analog inputs we would need, a reed switch made the most sense.

We 3D printed the anemometer with holes in the bottom to leave space for a small magnet to be glued into its bottom. There was also space in the bottom of it for a ball bearing, which was press fitted in. We then press-fit into the bearing a wooden dowel that was glued to the container for the anemometer. In the container wall was a reed switch with two long wires going to the board back in the main enclosure.

Our script averages the frequency of the magnet passing the read switch over 10 seconds. This time is then converted to revolutions per minute, which is then multiplied by the radius of the anemometer and 2π radians (one whole rotation). Finally, this speed is converted to miles per hour which is sent to the receiver and website. One end of the reed switch is pulled low, with the other end connected to an analog input on the ESP32. When the magnet passes the reed switch, the analog input pin will read low and use this interval time for the

calculations mentioned above. The code for this system is interrupt driven, only operating if the magnet rotates over the reed switch.

4.5. Subsystem 3: Wind Vane

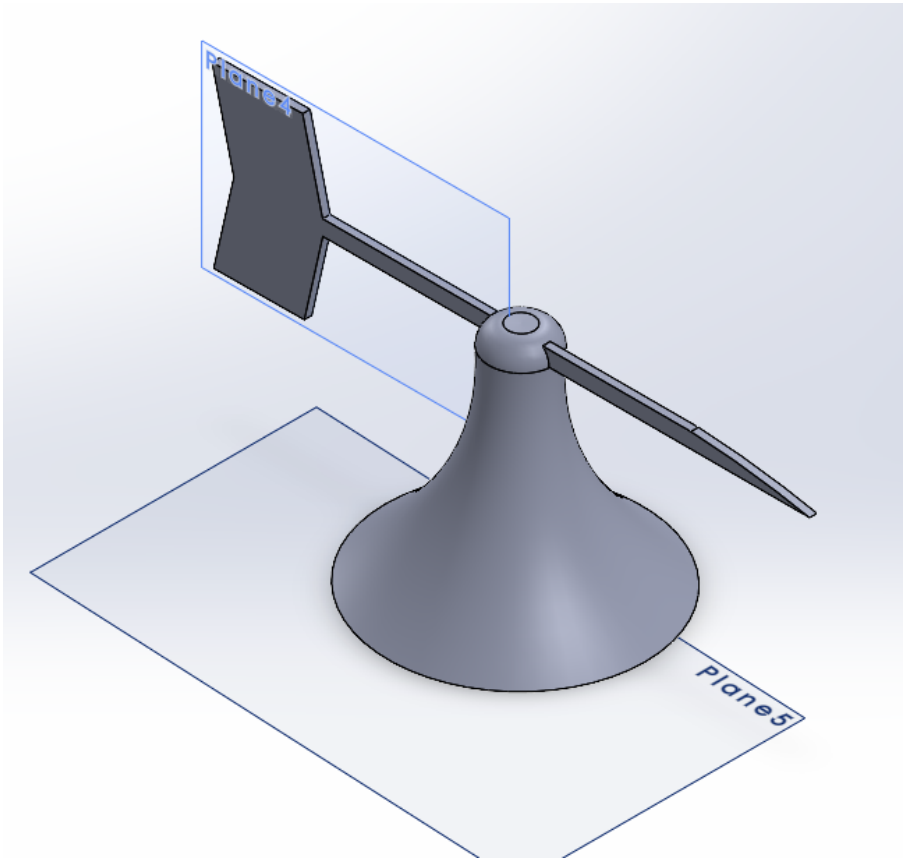


Figure 5. Wind Vane Solidworks Design

The wind vane is required to give wind direction. This data is recorded via hall effect sensors, which give a strength of magnetic field, which for our purposes is a measure of how close a group of magnets in the direction of the tip of the wind vane is to each to a group of four hall effect sensors. It operates similarly to the anemometer with a ball bearing and the wooden dowel, with the

difference being in the type of sensor used and the amount of signals that are being sent back to the board. We accidentally had some pins on the board that were supposed to be analog for the wind vane signals, but instead were digital pins. This caused some issues and required a workaround to include an external ESP32 to be used for analog to digital conversion, with the analog outputs being sent to our main board. These signals are then used to determine which direction the magnet and coincidentally the wind vane are pointed, with two high signals giving cardinal directions (N,E,S,W), and one signal being higher than others by a factor of two giving ordinal directions, (NW, NE, SW, SE). This was tested and a northern direction was labeled on the whole system to be set up with a compass. A breadboard mockup of the electronics inside the windvane is shown in Figure 6.

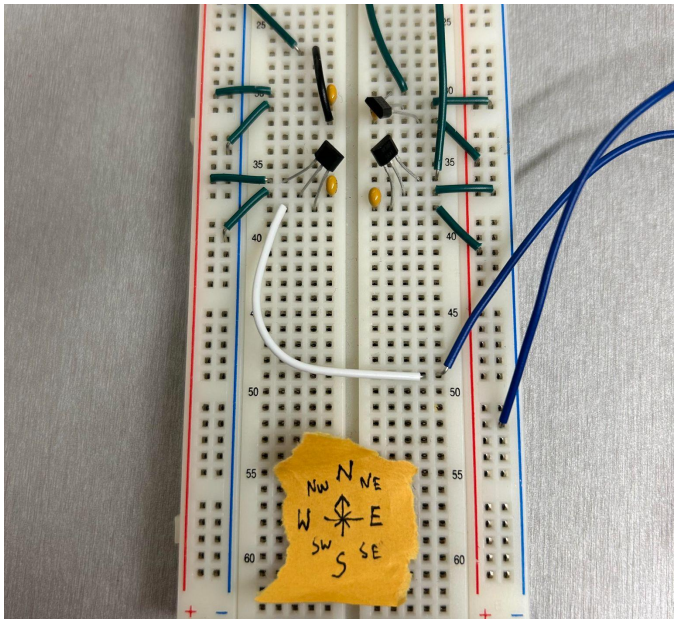


Figure 6. Wind Vane Breadboard Design

A group of magnets is drilled into the top of the wind vane, with the four hall effect sensors soldered inside the bottom wind vane assembly on protoboards. Each hall effect sensor is orientated on the ordinal directions. The hall effect sensors return magnetic field strength (south pole) through an analog signal of 0-4096 to the ESP32. If any hall effect sensor reads above 2000, the ordinal direction of the given hall effect sensor is sent to the receiver. Each cardinal direction was calibrated by changing the thresholds of two hall effect sensors. For example, if the northwest sensor and southwest sensor both read above 800, the direction west would be transmitted to the receiver from the weatherstation.

4.6. Subsystem 4: LoRa Communication

Because our weather station will be placed in remote locations, we needed a way to communicate with it wirelessly over moderate distances while conserving battery power. We opted to use the LoRa protocol because it best fit this requirement. Other options like Bluetooth, WiFi, and LTE were either too limited in range or required additional supporting infrastructure to operate.

LoRa operates on an unlicensed band at 915 MHz so there is no need for government approval to use it. The principle selling point of LoRa is that it uses a very narrow bandwidth to communicate. This means that the noise in the signal is very small, so the transmit power can also be very small. In the case of a battery operated circuit, this lower power usage is essential. Because of the

narrow bandwidth, LoRa data rates are much lower than other protocols, but our application did not require high throughput so this was a non-issue.

Another benefit of LoRa for this application is that it is well-supported in the Arduino/ESP32 ecosystem. We chose the ubiquitous RFM96 transceiver because there were several available libraries to interface with it from the ESP32, and because all of the LoRa modules in the lab were based on the same transceiver. This made testing code before our board was built possible.

Once the boards were built, we tested the communication between them at various distances. We demonstrated that communication still worked at about 100 yards, which was our target distance for a demo. Signal strength was still quite strong at that distance, so the station could be placed even farther without any changes to the monopole antenna design.

4.7. Subsystem 5: Power System

The power system on the remote station was required to provide 3.3V to the board from a solar panel and lithium-ion battery. We also opted to design the circuit from scratch, so prebuilt solar controllers or battery chargers were not an option.

The first stage of the power system is the solar charge controller. There are numerous options on the market for solar charging, but many of them did not fulfill all of our requirements. We needed an IC that would handle not only charging the battery but also providing power to the circuit in cases where the solar panel is generating enough energy to run the circuit. Many options on the

market do not have this load-switching capability. After much research, we decided on the BQ24210 from Texas Instruments. It had all of the features we required in addition to having thermal protection for our battery with the use of an external thermistor. It required several passive components, including several capacitors, resistors, and indicator LEDs. The datasheet specified a recommended board layout that we followed closely when designing our board.

The second stage of the power system is our switching regulator. Because this device is battery powered, we wanted to avoid using a linear regulator for a few reasons. First, linear regulators waste significantly more power than comparable switching ones. This not only puts pressure on our solar panel to keep up to replenish the wasted power, but also creates unnecessary heat in the enclosure. Second, there is a distinct possibility that our battery voltage drops below our desired supply voltage if its charge level diminishes. We want our weather station to work even if there are a few cloudy days, so this problem made a linear regulator a non-starter. There was also the issue of dropout voltage on most linear regulators we looked at. Single cell lithium-ion batteries have a voltage of 3.7V, and our board supply voltage was 3.3V. Most regulators that we looked at had a dropout voltage close to 1V, which meant that even in ideal conditions we would not have sufficient voltage to regulate properly. Special low-dropout regulators were expensive and hard to find, in addition to oftentimes having dropout voltages very close to our absolute maximum desired dropout.

The switching regulator IC we chose is the TP63030 from Texas Instruments. Its sibling, the TP63031, was our ideal choice because it is programmed at 3.3V off the shelf, but due to extremely limited availability we opted to go with the TP63030 instead. It is a variable regulator, so we had to design a resistor network to set the output voltage appropriately. Because the resistors we used did not perfectly match their labeled values, our actual voltage (3.41V) strayed slightly from the desired 3.3V, but was well within the margins acceptable for each of our sensors and the ESP32. We followed the layout recommendations given in the datasheet very closely given that the frequencies that switching regulators operate at make layout important. We also followed the datasheet's recommendation on type and size of the supporting capacitors and inductors.

The final choice we made was our solar panel. We opted for a 6V, 2W solar panel for a few reasons. First, the prices of solar panels we found were very similar for different wattages, i.e. the 1W and 2W panels were close in price. We decided that we could take advantage of this price difference to introduce some margin into our power supply. Second, we wanted a panel that was rated for outdoor use. Some panels we looked at were not rated for prolonged exposure to the elements without some external protection. Finally, our solar controller was rated for an input voltage between 3.5V and 18V. There were several panels we evaluated early on in the design process that had voltages lower than this threshold, so once we nailed down our charge controller we were able to narrow

down our panel options. After all of this consideration, we landed on a panel from a company called Voltaic that fulfilled all of our requirements. Figure 7 below shows the solar panel on top of the enclosure in our outdoor testing environment.

Figure 7. Solar Panel on the Enclosure

4.8. Subsystem 6: Enclosure

The enclosure had to house all of the weather station's components and keep them safe from rain while still allowing the sensors to collect data about the

current weather conditions. For this reason, we decided to use an enclosure that was sealed at the top and sides and that had ventilation at the bottom.

We chose a premade waterproof enclosure to fulfill these requirements for a few reasons. First, the cost of most premade units we looked at was significantly lower than a comparable 3D printed one even without the consideration of the time it would take to design the enclosure. Second, the off-the-shelf item gave us confidence that our enclosure would actually be waterproof without extensive testing on our part. This allowed us to focus our time on other essential features of the project.

The enclosure we ultimately decided on was a waterproof junction box from LeMotech. It is IP65 rated, so we could be sure that it would be waterproof. It is made out of plastic rated to be outdoors in sunlight all day, which beats out the materials we had access to in the EIH. It has several cutouts on the side that are sealed with rubber grommets, which made it easy for us to connect our solar panel and antenna externally without much concern for water ingress. We added several holes to the bottom of the enclosure for ventilation so that our sensors would get accurate readings as seen in Figure 8. Because the circuitry in the enclosure does not put out measurable heat due to its low power usage, our concern was chiefly that the sensors be exposed to outside air for accurate readings, not that the enclosure would overheat. Further, for the purposes of predicting forest fires, the temperature sensor only needs to read surface temperature, not air temperature.

Figure 8. Enclosure with Ventilation

The wind vane and anemometer were attached to the main enclosure using PVC pipes. We needed to set both sensors away from the main enclosure so that they had clearance to spin freely, and PVC pipe is widely available and low cost so it was a natural choice to connect them. The anemometer is secured to the PVC using superglue. The wires come from the bottom of the anemometer through the PVC pipe and up into the main enclosure where they are connected to the main board. Similarly, the wind vane is secured to its PVC pipe using a combination of superglue to hold it in place and caulk to seal it. The caulk was necessary only on the wind vane because of its larger bottom hole. The wires from the wind vane are fed through the pipes up to the main enclosure where they connect to the main board.

The final part of the enclosure is a vertical piece of PVC pipe used to elevate the station off of the ground. This pipe is hammered into the ground until

it is solidly seated, and then the rest of the enclosure can be affixed to it. We chose to raise the station off of the ground so that there was no risk of it being affected by debris, water, or critters on the ground. In a real application, the station would be placed even higher than we did for our demo to ensure accurate readings.

4.9. Subsystem 7: Receiver

The receiver was used to bridge the remote weather station and the web server. Its requirements were simple: it had to receive data from the remote station using the LoRa protocol and upload the received data to the web server using HTTP.

To achieve this, we designed and built a board based on the ESP32-C3-WROOM, the same WiFi-enabled microcontroller that we used on the remote weather station. This choice made sense because the chip has WiFi capability and can interface directly with our RFM96 LoRa module. In addition, it allowed us to reuse the knowledge we gained from designing the remote station to make the design process for this board smoother. Our rationale for choosing the LoRa module is outlined in Section 4.6.

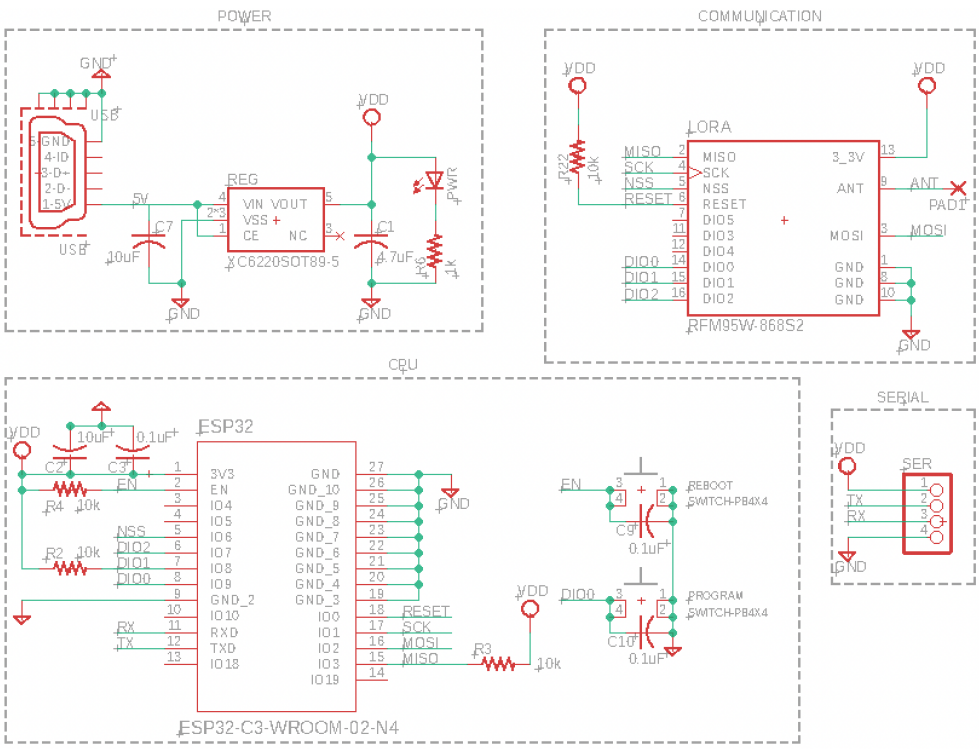


Figure 9. Schematic for Receiver

Because most of the functionality of the receiver is handled by the ESP32 and the LoRa module, there is very little else present on the board. We added the required supporting components to program the ESP32: pullup resistors on the strapping pins IO2 and IO8, buttons on the enable and strapping pin IO9, and a few bypass capacitors. We also included a 5V to 3.3V linear regulator so that we could power the board with a standard USB power supply. This choice was motivated by the ubiquity of 5V USB power supplies, and the fact that the receiver will likely be placed near the web server that to which it will be uploading data (which means there are likely available USB ports and power outlets

around), so there was no need to devise a battery system like on the remote station.

The code running on the receiver is equally simple. We used the LoRa library to interface with our LoRa module. The receiver will constantly listen for LoRa data until it receives a packet. Upon receiving the packet, the receiver will create an HTTP POST request with the appropriate headers and include the received LoRa packet as the POST data. For debugging purposes, we send some logging output using serial, but in the everyday operation of the receiver this data does not need to be viewed.

4.10. Subsystem 8: Web Server

The web server was used as the central processing hub of the project. All data collected in the field is uploaded to the web server to be logged and processed. The web server also generates an interactive dashboard showing the current likelihood of a forest fire, the most recent conditions measured, and a graph of historical values collected by the station.

The server is written in Python with a framework called Flask. Flask is a library that makes generating web servers in Python convenient by providing several functions to quickly map URLs to python functions. We chose it to save time and effort writing boilerplate for our web server so that we could instead focus on writing the logic that actually drove our project.

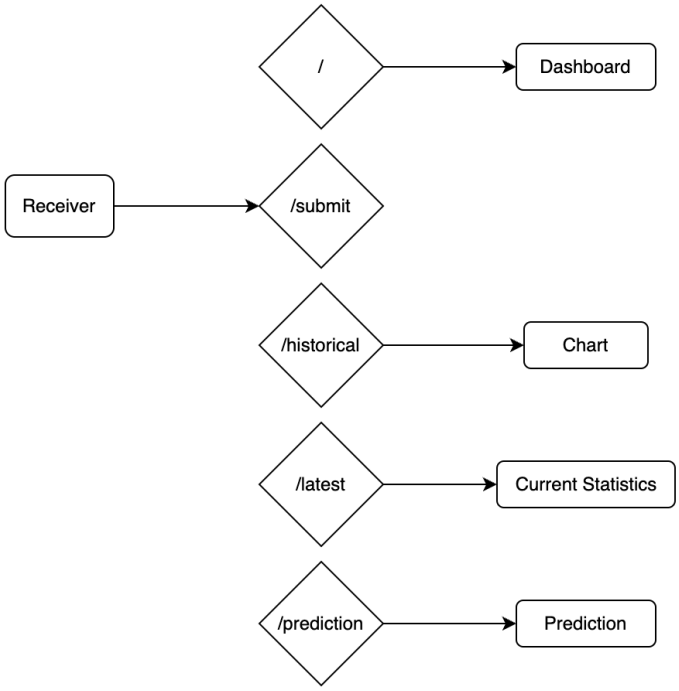


Figure 10. Block diagram for web server.

In our web server, there were a few URLs (commonly called *routes*) mapped to specific purposes:

1. `/` (base route)

   This is the URL of the base website. The web server will respond to this URL with the interactive dashboard.

2. `/submit`

   This route is where sensor data is submitted. The receiver uses HTTP POST requests to upload the data it receives from the weather station here, and then the web server will parse and store that data into a database.

3. `/historical`

   This route is used to retrieve historical data from the database. It accepts a parameter for how much data to retrieve so that a user can choose whether to show 1 hour, 6 hours, 1 day, 1 week, 1 month, or 6 months of data.

4. `/latest`

   This route is used to retrieve the latest sensor readings from the database.

5. `/prediction`

   This route is used to compute and then retrieve a likelihood prediction based on the conditions measured. In order to estimate the likelihood of a large forest fire, three different probabilities are calculated: the probability a small fire will ignite, the conditional probability a large fire will occur given a small fire has occurred, and the unconditional probability a large fire will occur. The probability of a small fire ignition (for example from lightning or a cigarette) increases with higher temperatures and lower humidity. This calculation was guided by research done by Texas A&M[1]. The conditional probability a large fire will occur given the ignition of a small fire increases with higher wind speeds, higher temperatures (particularly when higher than the location's monthly average), lower humidity, and more severe drought conditions. After estimating these

[1] "Reference Fuel Moisture Adjustment Table." *Texas A&M AgriLife*, https://agrilife.org/rxburn/weather-fuel/reference-fuel-moisture-adjustment-table-2/.

probabilities, the unconditional probability of a large fire occurring is calculated by multiplying the ignition probability and the conditional probability. This value is then assigned to a prediction which can be one of *Very Low*, *Low*, *Moderate*, *High*, *Very High*, or *Extreme*.

The dashboard itself was written by hand using HTML, CSS, and Javascript. The dashboard will poll the web server for new data every ten seconds (which is the time between sensor readings) and display new data when it is received. The graph widget is provided by a Javascript library called d3.js. The buttons above and below the graph that allow selection of the displayed data type and the duration of data were written separately by our group. Clicking a button will send a request to the web server to obtain the data type and duration specified, and will then generate an updated chart with the new data.

We chose to write the website by hand (instead of using a site generator or a larger web framework like React) because we had only a few pieces of functionality that all existed on a single page. Tools that aid in website generation are mostly helpful in keeping things consistent across several different pages, but would only add overhead in our project. Our approach kept our website small, focused, and easy to modify for people who might not be familiar with more complex web design tools.

5. System Integration Testing

5.1. Integration Testing

Due to the nature of our project, we were not able to test the subsystems before they were integrated. Therefore, our integration testing included some subsystem testing as well. To ensure the sensors were reading properly and communicating with the ESP32 microcontroller, we logged their output on the serial bus and then monitored the serial output. Since we were receiving data that matched with what we expected, we knew the on-board sensors were working as expected. To test wind speed, we hand spun the anemometer to emulate wind speeds while inside a lab. For direction, we used a compass alongside our wind vane to make sure the directions were matching up as expected. When the website was complete, it was easy to re-verify these measurements. For the forest fire probability, we manipulated the code to force high temperatures and low humidities in order for a higher likelihood to display. With the conditions in South Bend and in the lab, it was impossible to get a high likelihood authentically, so these modifications were necessary to ensure that the decision code was working.

5.2. Demonstrated Requirements

We knew the integrated system was working together when the website displayed accurate weather measurements from the sensors. This meant that the sensors were accurately making measurements, the receiver was getting all the

information from the station, the web server was analyzing the data as expected, and the website was displaying it properly. We made sure that the website was updating at the appropriate interval to determine whether the entire system was working properly.

6. User Manual

The weather station consists of three separate systems that communicate to produce the full working system: the remote weather station, the receiver, and the web server. The remote weather station and the receiver are both programmed already, and require very little setup. The web server requires that users download and run software, and some configuration is required.

6.1. Remote Station

To set up the remote weather station, a user must first find a suitable location to place it. Ideally, the station would be placed in an open field so that measurements reflect the actual conditions of the area. The station is powered by sunlight, so the location should get at least occasional sunlight. Locations with frequent shade are not ideal. After the location is chosen, the user can install the station by inserting the vertical pipe solidly into the ground and then affixing the actual station to that pipe. Once placed into the ground, the station is ready to transmit.

6.2. Receiver

To set up the receiver, the user should obtain a 5V USB power adapter, commonly used to charge phones and other small electronic devices. Once the device is powered up, it will begin receiving messages from the weather station and transmitting them to the web server using the local WiFi network. The receiver will need to be reprogrammed with the WiFi credentials and the IP address of the web server when used outside of the lab. Information on reprogramming the device can be found in Appendix 9.5: Programming with PlatformIO. Ensure that the antenna on the receiver is perfectly vertical. The device can then be connected to power and placed somewhere where it will not be tampered with.

6.3. Web Server

With the transmitter and receiver in place, the web server can be set up. Most of the software can be downloaded from the team website in a zip archive and extracted into a directory of the user's choice. Not included in that zip archive is the database to store the collected data (so that users will have a fresh one).

The database must be created from scratch with sqlite3, which comes standard on most operating systems. The following command should be used exactly to create the database in the same directory as the other files:

```
> sqlite3 weather_data.db
```

This will start the sqlite3 interactive prompt, which can be used to create the table needed to store the data. Enter the following SQL command to create the table according to the structure the web server expects:

```
CREATE TABLE weather_data (
     id             INTEGER PRIMARY KEY,
     timestamp      TEXT,
     temperature    REAL,
     humidity       REAL,
     pressure       REAL,
     wind_speed     REAL,
     wind_direction TEXT
);
```

The final step before starting the web server is to install the single Python dependency: Flask. Flask is a Python library that aids in creating web servers. It can be installed with the following command:

```
> pip install flask
```

The web server can then be started by invoking the `app.py` script using a modern (>=3.9.0) Python installation as follows:

```
> python3 app.py
```

On Windows, the command will be simply

```
> python app.py
```

Upon running the server, the terminal will begin to show a few log messages. One of these will be the IP address and port that the server is open at. Copying and pasting this full address (it should look something like

http://xxx.xx.xx.xxx:3000) into the address bar of a web browser will bring up the website which displays real-time data from the weather station.

If the setup is working properly, the website should update every ten seconds with new data. The user can see when the site was last updated by looking at the text under the “Current Statistics” heading.

6.4. Troubleshooting

If the website is not updating with new readings every ten seconds, there are a few things to troubleshoot. Ensure that the receiver is in range of the WiFi network. We have found through basic testing that the WiFi connection range on the receiver is somewhere between 50 and 75 feet from the wireless access point. If the receiver is in range of the access point and still not receiving data, check that its antenna matches the orientation of the antenna on the weather station. Typically this means the antenna should be perfectly vertical. If that still does not work, ensure that the weather station has power. Open the lid of the station and check that the green LED is lit. This indicates that the battery is supplying power to the board. Further, ensure that the red LED is lit when the station is in sunlight. This means that the battery is charging.

If all of those checks do not work, make sure that the IP address programmed into the receiver matches the IP address that the web server is running at, and that the WiFi credentials programmed into the receiver are correct.

7. To-Market Design Decisions

There are numerous decisions that we made early in the design process that turned out to be anywhere from completely wrong to somewhat impractical later in the construction of the project.

The first, and perhaps most significant, design change we would make would be the board. There were several issues relating to pin selection on the ESP32 which meant that board traces had to be cut and rerouted. First, some of the strapping pins of the ESP32 (IO2, IO8, and IO9) were connected to the LoRa module to be used as basic digital IO. When designing the board, we did not foresee any issues because the strapping pins can be used as regular IO after the boot process completes. After we got the board built, though, we found that we could not put the ESP32 into programming mode. This pointed to an issue with the strapping pins. After some troubleshooting, we found that the LoRa module's digital input pins defaulted to a pulled-down state at startup, which meant that our strapping pins were always pulled low. To put the ESP32 into programming mode, however, we needed those strapping pins pulled high.

Obviously, this was a large issue for our board. To fix it, we had to cut the traces connecting the strapping pins to the LoRa module. Luckily, our code did not use those pins on the LoRa module anyway, so we didn't need to reconnect them to other pins. If we could redesign the board, we would choose different pins on the ESP32 to connect to the LoRa module so that the strapping pins could be left isolated. One thing to note is that even after reading the entire datasheet of our LoRa module, the input pins being

pulled low is not mentioned anywhere, so we do not think it was unreasonable to build the board as we did. The issue could only be found through testing.

Another issue we had with the board was also related to pin selection. We mistakenly assumed when designing the board that analog inputs were remappable. The datasheet for the ESP32 mentions that GPIO is remappable for every pin, but we missed the key detail that this is only for digital IO. Our wind vane required four analog inputs because it was based on hall-effect sensor readings, but we did not assign it four pins that were capable of doing analog readings. Furthermore, the pins on the ESP32 that are analog capable (of which there are 6) were all connected in ways that made it prohibitive to reassign them without a complete redesign of the board.

For that reason, we had to find a way around using analog inputs on our board. The solution we came up with was to use another ESP32. This ESP32 would do the analog read, compute the direction of the wind, and then send a three bit digital signal to the main ESP32 on our board. It increased the power draw of our station significantly (from 20 mA to about 90 mA), but we had designed our power supply with enough margin that this was not an issue.

In addition to connection changes, we decided during the board design phase to change one of the sensors we had selected. One of our chief concerns when choosing components was cost, and because of that we opted for a very small, low-cost pressure sensor. When designing the board, we found that the sensor was so small that the pin pitch was too narrow for our board house (OSH Park) to produce. We had to select a new pressure sensor with a larger pitch that had the same features.

Another design decision we made for our demo, but that we would change in a real product, is our enclosure system. We used a premade plastic enclosure from LeMotech and various pieces of PVC pipe to house the wiring between our main board and the wind vane and anemometer. We used caulk and superglue to connect and seal the enclosure in various places. The result is good enough for a demo, but more care would have to be taken in a commercial system to ensure protection from water ingress. Additionally, the main enclosure should also be redesigned to have built in ventilation as well as designed holes for the solar panel wiring and the antenna to come out of. In addition to these practical design decisions, the to-market product will also need to be more visually appealing than our design, with matching colors and higher quality printers for the objects in order to make it appear professional.

More optimized 3-D prints for the anemometer and wind vane should be implemented in order to guarantee smooth spinning as well as more precise measurements. The wind vane in particular is not especially precise, giving only the 8 cardinal and ordinal directions. It might be useful in a commercial product to have a more accurate reading on wind direction (perhaps showing direction in degrees) so that fires can be accurately pinpointed and predicted. This could perhaps be achieved by using more hall effect sensors in a larger ring so that the readings are more granular.

A new feature that would be too expensive and logistically challenging for our demonstration is the alert system, which will need to be implemented into the to-market product. There are numerous companies that offer SMS messaging services which could be used to send text messages to affected parties.

Another change that would be made in a marketable design would be that the station would have to be at least 33 feet in the air and 50 feet away from any tall vegetation, in order for proper readings to be gathered.

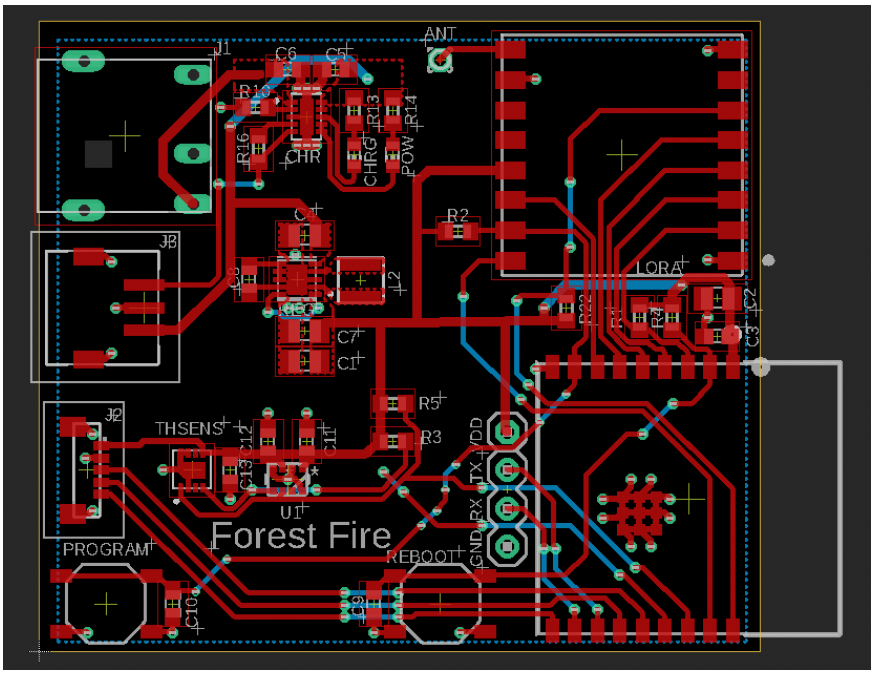
8. Conclusion

We designed a remote weather station with the unique implementation of monitoring for potential forest fires. With advanced detection, we hope to save lives and protect nature. We combined the hardware component of the sensors for the collection of data with a significant software component for the transmission and analysis of data. The result is an implementable technology complete with an easy to understand website that displays fire probability that will protect forest environments and the people who live in them.

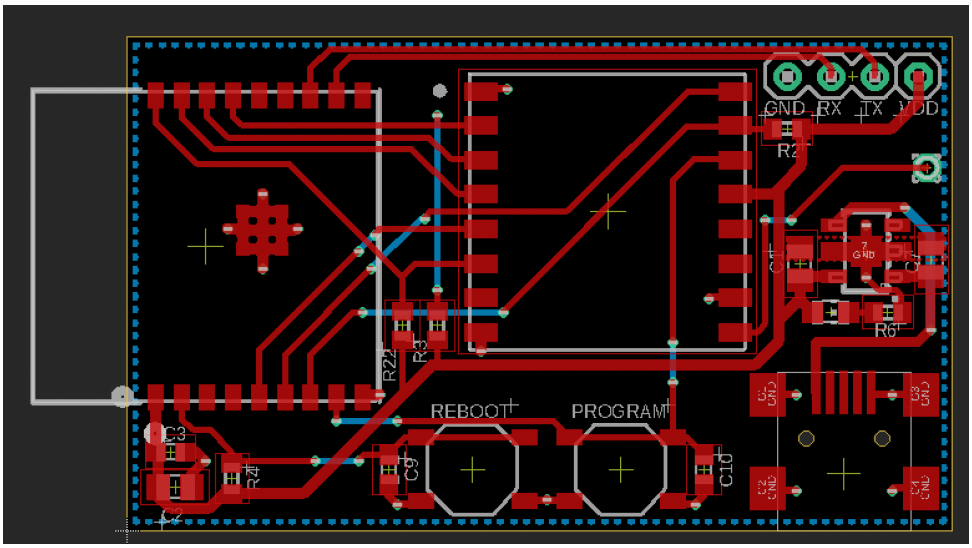
9. Appendices

9.1. Board Designs

9.1.1. Remote Station

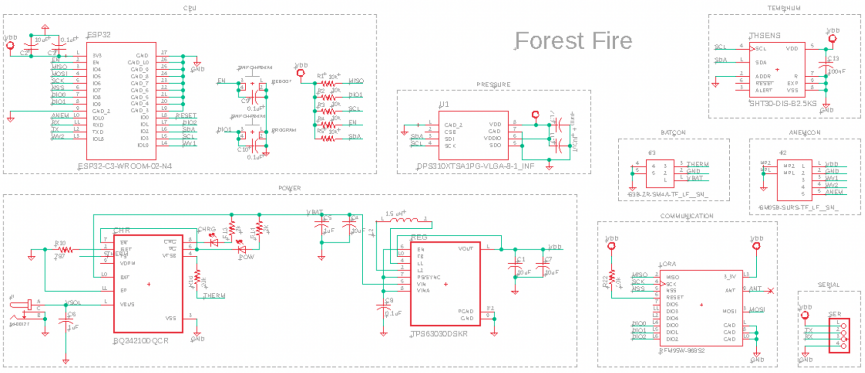


9.1.2. Receiver

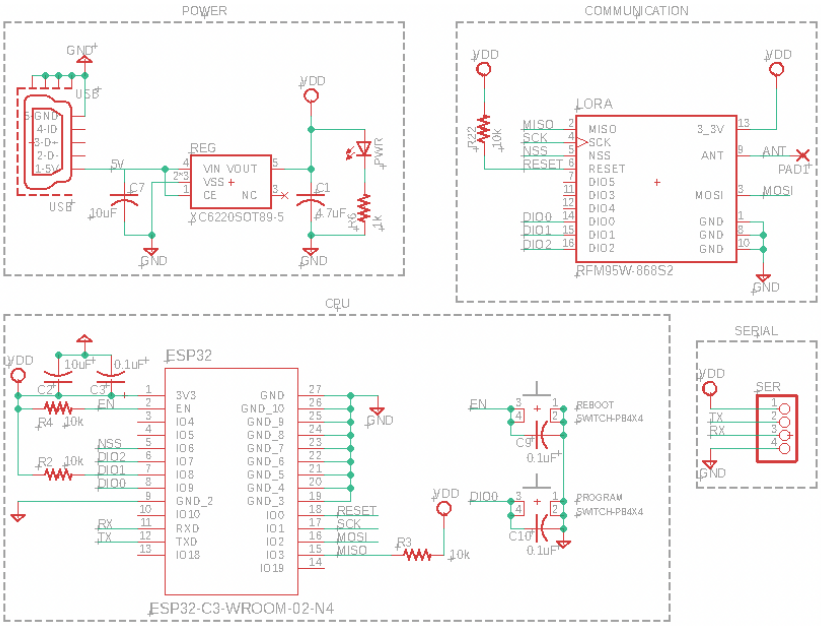


9.2. Schematics

9.2.1. Remote Station



9.2.2. Receiver



9.3. Code

9.3.1. Remote Station

```
#include <SPI.h>
#include <LoRa.h>
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_SHT31.h>
#include <Adafruit_DPS310.h>

// RFW9X pin mappings on LILYGO board
#define SCK 6
#define MISO 4
#define MOSI 5
#define SS 7
#define RST 0
#define DI0 -1

const int SCL_PIN = 3;
const int SDA_PIN = 2;
const int ANEM_PIN = 8;

const int WV0 = 1;
const int WV1 = 10;
const int WV2 = 18;

Adafruit_SHT31 sht30 = Adafruit_SHT31();
Adafruit_DPS310 dps310;
Adafruit_Sensor *dps_pressure =
dps310.getPressureSensor();

// Initialize wind speed variables
float wind_mps = 0;
```

```
float wind_mph = 0;
float rev_per_time = 0;
unsigned long firstMillis = 0;
unsigned long lastMillis = 0;
unsigned long lastIntTime = 0;
int rev_counter = 0;

void IRAM_ATTR windSpeedISR ()
{
  unsigned long intTime = millis();
  if(intTime - lastIntTime > 100)
  {
    if (rev_counter == 0)
      firstMillis = millis();
    rev_counter++;
    lastMillis = millis();
  }
  lastIntTime = intTime;
}

/* Read temp and humidity data and display it */
void readAndDisplaySensor(void *param)
{
  float humidity, temp;
  char displayStr[50];
  sensors_event_t pressure_event;
  while (1) {
    temp = sht30.readTemperature();
    humidity = sht30.readHumidity();

    if (dps310.pressureAvailable()) {
      dps_pressure->getEvent(&pressure_event);
    }
```

```
    rev_per_time = 0;
    if(rev_counter != 0)
      rev_per_time = 1.0 / ((lastMillis - firstMillis)
/ rev_counter);

    wind_mps = rev_per_time*1000*6.2831853*0.092; //
wind in meters/sec
    wind_mph = wind_mps*2.237;
    rev_counter = 0;

    u_int8_t wv0 = digitalRead(WV0);
    u_int8_t wv1 = digitalRead(WV1);
    u_int8_t wv2 = digitalRead(WV2);
    u_int8_t wdir = (wv0 << 2) + (wv1 << 1) + (wv2);

    sprintf(displayStr, "temp: %.2f\nhum: %.2f\npres:
%.2f\nwspd: %.2f\nwdir: %d\n", temp, humidity,
pressure_event.pressure, wind_mph, wdir);
    Serial.println(displayStr);
    LoRa.beginPacket();
    LoRa.print(displayStr);
    LoRa.endPacket();

    vTaskDelay(10000 / portTICK_PERIOD_MS);
  }
}

/* Make sure the sensor can be found */
void findSensor()
{
  /* Find the sensor */
  Serial.println("Looking for SHT30...");
  int retries = 0;
  while (!sht30.begin(0x44) && retries < 10) {
```

```
    Serial.println("Couldn't find SHT30");
    retries++;
    delay(500);

    if (retries == 10)
      while (1) delay(10);
  }
  Serial.println("Found SHT30 sensor");

  Serial.println("Looking for DPS310...");
  retries = 0;
  while (!dps310.begin_I2C(0x76) && retries < 10) {
    Serial.println("Couldn't find DPS310");
    retries++;
    delay(500);

    if (retries == 10)
      while (1) delay(10);
  }
  Serial.println("Found DPS310 sensor");
}

void setup() {
  Serial.begin(115200);
  Wire.begin(SDA_PIN, SCL_PIN); // set the correct scl
and sda pins on our board

  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DI0);
  if (!LoRa.begin(915e6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  Serial.println("Started LoRa");
```

```
  findSensor();

  pinMode(ANEM_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(ANEM_PIN),
windSpeedISR, FALLING);

  pinMode(WV0, INPUT_PULLDOWN);
  pinMode(WV1, INPUT_PULLDOWN);
  pinMode(WV2, INPUT_PULLDOWN);

  xTaskCreate(readAndDisplaySensor, "display sensor
data", 16384, NULL, 1, NULL);
  vTaskDelete(NULL);
}

void loop() {}
```

9.3.2. Receiver

```
#include <SPI.h>
#include <LoRa.h>
#include <Wifi.h>
#include <HTTPClient.h>
#include <Adafruit_SHT31.h>
#include <Adafruit_DPS310.h>

// RFW9X pin mappings on our base board
#define SCK 1
#define MISO 3
#define MOSI 2
#define SS 6
```

```
#define RST 0
#define DI0 -1

const char* ssid = "SDNet";
const char* pass = "CapstoneProject";

const char* url = "http://192.168.10.101:3000/submit";

char buf[1024];

void setup() {
  Serial.begin(115200);

  Serial.println("Initializing rx...");

  WiFi.begin(ssid, pass);

  SPI.begin(SCK, MISO, MOSI, SS);
  LoRa.setPins(SS, RST, DI0);
  if (!LoRa.begin(915E6)) {
    Serial.println("Failure!");
    while (1);
  } else {
    Serial.println("Success!");
  }
}

void loop() {
  // try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) {

    while (LoRa.available()) {
      LoRa.readBytes(buf, packetSize);
```

```
    }

    if (WiFi.isConnected()) {
      WiFiClient client;
      HTTPClient http;

      http.begin(client, url);
      http.addHeader("Content-Type", "text/plain");
      String data = String(buf);
      data += String("rssi: ") +
String(LoRa.packetRssi());
      int resCode = http.POST(data);
      Serial.println("Uploaded data.");
    }
  }
}
```

9.4. *Components and Datasheets*

Purpose	Model	Datasheet
Solar Charge Controller	**TI BQ24210**	https://www.ti.com/general/docs/suppproductinfo.tsp
Buck/Boost Converter	**TI TPS63030**	https://www.ti.com/general/docs/suppproductinfo.tsp
LoRa Module	**RFM96W**	https://github.com/SeeedDocument/RFM95-98_LoRa_Module/blob/master/RFM95_96_97_98_DataSheet.pdf
Microcontroller	**ESP32-C3-WROOM**	https://www.espressif.com/sites/default/files/documentation/esp32-c3-wroom-02_datasheet_en.pdf
Temp/Humidity Sensor	**SHT-30**	https://media.digikey.com/pdf/Data%20Sheets/Sensirion%20PDFs/HT_DS_SHT3x_DIS.pdf

Purpose	Model	Datasheet
Pressure Sensor	**DPS310**	https://www.infineon.com/dgdl/Infineon-DPS310-DataSheet-v01_02-EN.pdf?fileId=5546d462576f34750157750826c42242
Solar Panel	**Voltaic P126**	https://cdn-shop.adafruit.com/product-files/5366/5366_Voltaic+Systems+P126+R1E.pdf

9.5. Programming with PlatformIO

The included code for the project has two folders: one is named **tx** and the other is named **rx**. These folders contain the code for the remote station and the receiver, respectively. The included `platformio.ini` file will configure the PlatformIO environment with separate tasks for both tx and rx. Either device can be reprogrammed by selecting the task labeled “Upload” under the device’s category (e.g., rx for the receiver).