Renewable Energy Lab Weather Station

Final Design Report

Chenxi Dong: CAD Engineer
Rowan McCullough: Test Engineer and Budget Liason
Ian Torp: Project Manager and Financial Manager
Shutong Wang: Manufacturing Engineer

Summer 2025-Fall 2025

Project Sponsor / Instructor: Carson Pete

DISCLAIMER

This report was prepared by students as part of a university course requirement. While considerable effort has been put into the project, it is not the work of licensed engineers and has not undergone the extensive verification that is common in the profession. The information, data, conclusions, and content of this report should not be relied on or utilized without thorough, independent testing and verification. University faculty members may have been associated with this project as advisors, sponsors, or course instructors, but as such they are not responsible for the accuracy of results or conclusions.

EXECUTIVE SUMMARY

The Renewable Energy Lab at Northern Arizona University has requested that we design a weather station for academic use. Professor Carson Pete approached us with the proposal and provided us with his requirements. The weather station collects six different types of weather data and is able to be accessed remotely at any time. The six different weather data types include air temperature, barometric pressure, humidity, solar irradiance, wind direction, and wind speed.

The weather station feeds all the sensors' outputs into a Raspberry Pi located inside the Renewable Energy Lab. This Raspberry Pi was provided and is a Raspberry Pi 3+. It collects all the data from the sensors and uploads it to our website for visibility. Each sensor has its own code within the Pi to obtain the readable data, as each sensor is calibrated differently. Each sensor uploads to our website within a timely manner. On the website, it displays daily data as well as the last ten minutes of data collected.

As for structural design, the lab has provided us with a tall metal truss which we have mounted our sensors upon. The platform atop this truss utilizes a square shaped crossbar with diagonal supports mounted to the top of this platform. The thermometer, pyranometer, and barometric pressure sensor are mounted upon this tower. The anemometer and wind vane are attached via boom arms to a separate existing tower.

This is the final report for this project. We have successfully mounted and applied seven working sensors to our Raspberry Pi. We have completed testing and have recieved client approval for each step and engineering requirements met with our project. Testing has shown solid results from our sensors and the mounting of each has gone smoothly.

TABLE OF CONTENTS

Contents

1 Background

This chapter of the report will discuss the initial importance of the project. The first section will describe the project proposal and explain why the project is important. It will also discuss the budget and fundraising targets of the project. The next section will discuss the major deliverables for the course, covering each deadline for the project. The final section of this chapter will discuss our success metrics for the project. This will include our goals as well as the major design requirements of our project in general.

1.1 Project Description

We have been tasked with assembling a weather station for the Renewable Energy Lab (RE Lab) at Northern Arizona University (NAU). This station is required to provide current, accurate weather data readings for academic and scientific use. While the RE Lab used to have functioning weather data collectors, they no longer function. Additionally, the previous collection required the professor to go into the RE lab itself and read the current data. There was no way to access this data remotely, nor was there a way to access previous data. Our aim is to fix this problem. Our project collects data to display and store online for multiple years. This has allowed us to alleviate the headache of collecting weather data from the RE Lab. While our weather station does not provide a forecast for upcoming weather events, the data collected could be sufficient to make such determinations. Weather stations are important installations, typically for the purpose of predicting future weather events, disaster preparedness, and agricultural information. Our weather station collects data on temperature, rainfall, humidity, air pressure, solar radiation, and wind speed. The classes that our project should be the most influential for are the Renewable Energy course at NAU and the Wind Power course. The students and faculty are able to access our data to perform calculations and other deliberations.

Our budget for this project includes around $3100 of preexisting equipment, an additional $500 dollars from the SCE Capstone office for additional costs, and an expected $300 in fundraising from the team. To meet this, each team member has contributed $75 from their own personal savings.

1.2 Deliverables

The project aims to deliver a functional modular weather station to meet the needs of the Renewable Energy Laboratory near NAU. The weather station has been designed to measure six key environmental variables: temperature, humidity, wind speed, air pressure, solar irradiance, and wind direction. Final deliverables include:

- A fully assembled solar-powered weather station utilizing a Raspberry Pi for data processing and wireless transmission.
- Robust sensor integration through modular mounting and standardized interfaces, following WMO siting guidelines.
- A cloud-based data display dashboard built with Adafruit IO capable of displaying real-time and historical measurements.
- Sensor calibration models and at least one working mathematical modeling example (e.g., humidity-to-voltage conversion or wind speed calculation).
- Custom data management code written in Python to read, process, average, and upload sensor values.
- A complete CAD design of the sensor support structure, paying attention to maintenance, sensor replacement, and weatherization.

- A detailed technical report outlining the design process, benchmark comparisons, concept generation, modeling, and test results.

1.3 Success Metrics

The success of this project will be evaluated based on functional performance and how well it meets the engineering goals described throughout the report. The following criteria was used:

- Measurement Accuracy: All sensor outputs should meet the manufacturer's specified accuracy standards under test conditions.
- Data Uptime: The weather station should demonstrate consistent data collection and transmission, with at least 90% successful uploads within a 24-hour test window.
- Modularity and Maintenance: Sensors must be individually replaceable and independently verifiable in the event of a failure.
- Online Visualization: All six variables must be clearly displayed on the Adafruit IO dashboard with appropriate units and labels.
- Structural Stability: The tower and sensor mounts must be able to withstand moderate outdoor wind, rain, and temperature variations typical of Flagstaff.
- Documentation Quality: All code, CAD models, and calibration methods must be clearly organized and submitted in the final report format.

2 Requirements

This section will outline the customer needs and engineering requirements of the weather station project. The customer needs were provided by our client with levels of importance assigned to each. The engineering requirements were identified as a team and are quantifiable. These were used to create the QFD which compares and identifies the relationships between each other and helps to identify what our main focuses should be. It was also used to grade our benchmarks on how well they would accomplish the needs of our project.

2.1 Customer Requirements (CRs)

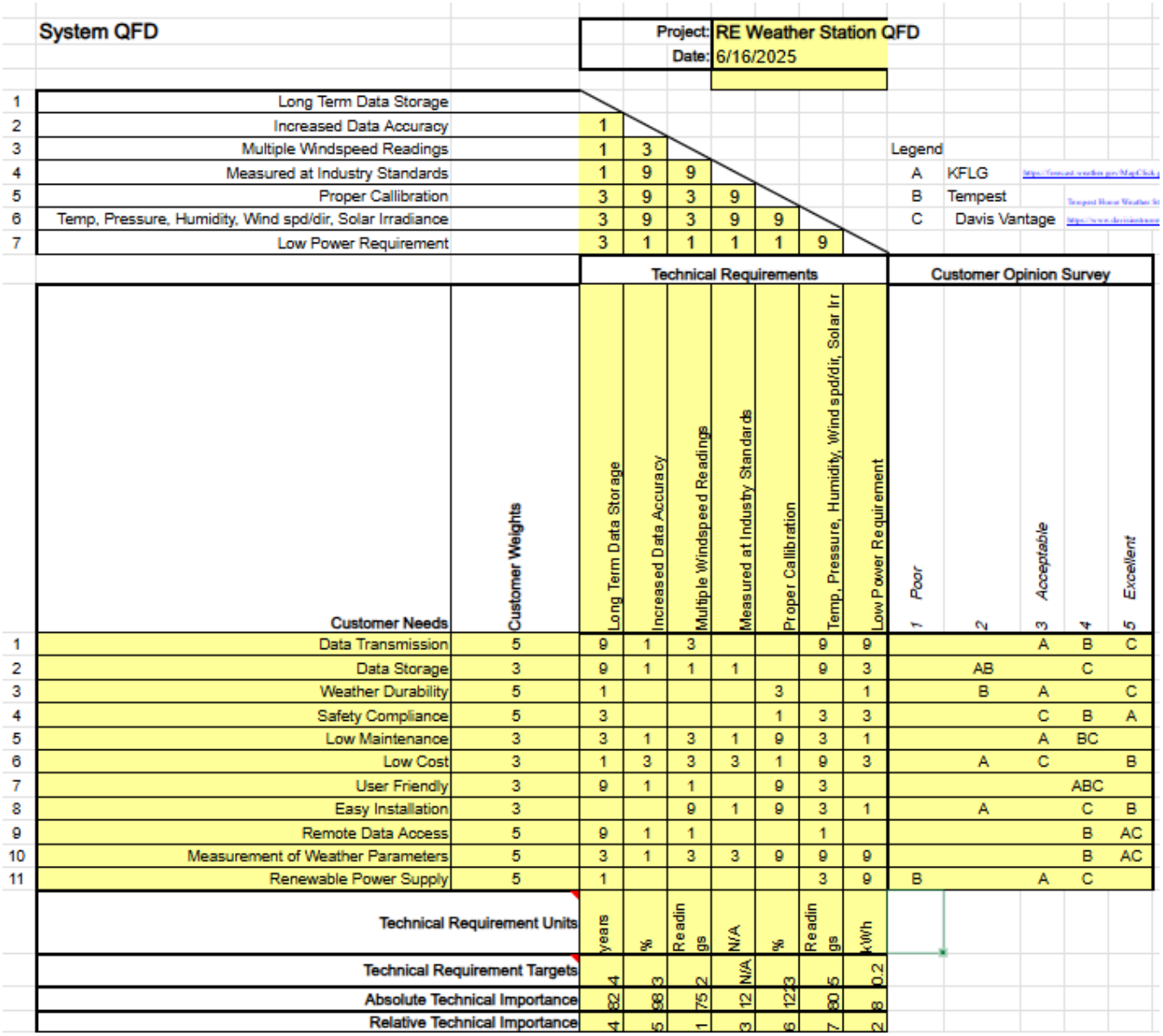
Measurement of Key Weather Parameters - Station measures temperature, humidity, wind speed and direction, barometric pressure, and solar irradiance.
Data Transmission - Data collected is transmitted via the internet.
Remote Data Access - Live and stored data should be accessible through a web interface.
Renewable Power Supply - Any components which require power must run on solar energy.
Weather Durability - Station must withstand outdoor weather conditions.
Low Maintenance - Station should require less than 2 hours of maintenance per year.
Sensor Accuracy - Sensor readings should be properly calibrated.
User Friendly - User interface should be easily navigable.
Ease of Installation - Installation should require minimal tools or training.
Low Cost - Station should be cost effective and within budget.
Safety Compliance - Must comply with relevant electrical and operational safety standards.
Data Storage - Data should be stored in an accessible and organized database for at least one year.

2.2 Engineering Requirements (ERs)

Long Term Data Storage - Database should log data in an organized manner over the course of 4 years.

Increased Data Accuracy - Sensor readings should be highly accurate, within 3-5%. Data average calculations should be properly computed.
Multiple Wind Speed Readings - Wind speed and direction should be measured at both standard height and atop existing tower at lab providing at least 2 readings.
Measured at Industry Standards - Sensors should be properly positioned according to industry standards.
Proper Calibration - Sensors should be properly calibrated to upload accurate data from raw readings within 3-5% of the true values.
Measurement of All Data Types - Station should record 6 data types including temperature, pressure, humidity, solar irradiation, wind speed, and wind direction.
Low Power Requirement - Station should be capable of fully operating under existing solar generated power means located at lab, with a target of 0.2 kWh per day or less.

2.3 House of Quality (QFD)



System QFD			Project: RE Weather Station QFD							
			Date: 6/16/2025							
1	Long Term Data Storage									
2	Increased Data Accuracy		1							
3	Multiple Windspeed Readings		1	3						
4	Measured at Industry Standards		1	9	9					
5	Proper Callibration		3	9	3	9				
6	Temp, Pressure, Humidity, Wind spd/dir, Solar Irradiance		3	9	3	9	9			
7	Low Power Requirement		3	1	1	1	1	9		

Legend: A KFLG; B Tempest; C Davis Vantage

			Technical Requirements							Customer Opinion Survey				
	Customer Needs	Customer Weights	Long Term Data Storage	Increased Data Accuracy	Multiple Windspeed Readings	Measured at Industry Standards	Proper Callibration	Temp, Pressure, Humidity, Wind spd/dir, Solar Irr	Low Power Requirement	1 Poor	2	3 Acceptable	4	5 Excellent
1	Data Transmission	5	9	1	3			9	9			A	B	C
2	Data Storage	3	9	1	1	1		9	3		AB		C	
3	Weather Durability	5	1				3		1		B	A		C
4	Safety Compliance	5	3				1	3	3			C	B	A
5	Low Maintenance	3	3	1	3	1	9	3	1			A	BC	
6	Low Cost	3	1	3	3	3	1	9	3		A	C		B
7	User Friendly	3	9	1	1		9	3					ABC	
8	Easy Installation	3			9	1	9	3	1		A		C	B
9	Remote Data Access	5	9	1	1			1					B	AC
10	Measurement of Weather Parameters	5	3	1	3	3	9	9	9				B	AC
11	Renewable Power Supply	5	1					3	9	B		A	C	
	Technical Requirement Units		years	%	Readings	N/A	%	Readings	kWh					
	Technical Requirement Targets		4	3	2	N/A	123	5	0.2					
	Absolute Technical Importance		82	98	75	12		80	8					
	Relative Technical Importance		4	5	1	3	6	7	2					

Figure 1: House of Quality

3 Research Within Design Space

3.1 Benchmarking

One benchmark we chose to consider is the Flagstaff Pulliam Airport weather station [7]. It is accessible through Weather.gov and shows a forecast for the area. While our weather station does not provide a forecast for future weather events, it provides current and accurate weather data. The airport weather station provides readings for temperature, humidity, wind speed, barometer, dew point, visibility, and heat index. It also displays the latitude and longitude of the weather station, as well as its elevation. It also clearly displays when the last update occurred for the website. These updates occur every 60 minutes. A three-day and seven-day history are both available, which we would like to emulate on our own

website. The website itself is clear and user-friendly, which are important design features for our team. It uses both line graphs and tables to display all the collected weather data, which is something we plan to do as well. This benchmark has acted as an outline for our website as well as a data confirmation system for our own readings.

Our second benchmark is the Tempest Home Weather Station, a consumer product available online [12]. This single unit device costs about $350 and claims to measure many of the weather parameters we wish to measure. The product also includes a cell phone app which allows you to access the data from the device as well as a 10-day forecast. While the device seems incredibly versatile for its cost and portability, the station does not appear to be equipped with many of the sensors needed to make the readings it claims to. For example, there is no anemometer or wind vane of any kind attached to the unit, yet it claims to measure wind speed and direction. While the app is user friendly and provides a lot of information, there is no database feature to record past weather patterns. Additionally, some reviews of the station seem to believe that many of the 'readings' it claims to make are just pulled from another source and posted as coming from the personal device.

Our third benchmark chose Davis Vantage Pro2 as a reference. This is a high-end weather station widely used in agriculture, scientific research and universities. Its main features include: integrated temperature, humidity, wind speed and other sensors; support wireless data transmission up to 300 meters; solar and battery-powered, suitable for field deployment. Its structure is tower-shaped and the anemometer is installed on the top to reduce interference. The modular design can also support additional functions such as ultraviolet rays. Its performance is very consistent with the goals of our project.

3.2 Literature Review

3.2.1 Ian Torp

[1] Setra Systems, Inc, "Barometric Pressure Sensors | Setra Systems," Setra.com, 2025. https://www.setra.com/product/pressure/barometric (accessed Jun. 18, 2025). This source is a storefront for barometric pressure sensors. For the purpose of our project, this has been used to help determine some benchmarks for the pressure sensor specifically. It displays 5 different models and compares them to one another using the criteria of pressure fittings, electrical terminations, output, media compatibility, thermal effect, compensated temperature range, operating temperature, accuracy FS, ranges in PSI, sensing technology, and sample applications. This became more useful to us as we began to tinker with our provided barometric pressure sensor. This website also explained some of the basics of barometric pressure sensors, which was useful in determining exactly why this sensor is important.

[2] N. US Department of Commerce, "Standards and Policy," www.weather.gov. https://www.weather.gov/coop/standards. This is a government-run website which has a number of useful resources. The one used for this project was NWSI 10-1302: Requirements and Standards for NWS Climate Observations. This document discusses the site and exposure standards of weather stations, as well as more specific air temperature measurement standards and precipitation measurement standards. It also discusses soil temperature measurement standards and pan evaporation measurement standards, although that is not useful information for this project. This is where we learned that thermometers must be able to measure between –20 degrees Fahrenheit and 115 degrees Fahrenheit, and that precipitation measurements must be able to reach 20 inches. The discussion of each sensors' standards has informed our decision making throughout our design process.

[3] World Meteorological Organization, Guide to meteorological instruments and methods of observation., Volume 1. Geneva, Switzerland: World Meteorological Organization, 2008. This is my

most important source so far. Each chapter in this book explains a different measurement and its standards. Each chapter discusses different sensors and tools to measure each weather feature. It goes in-depth on everything from standards to calibration. I have already utilized chapters one through seven. These are, in order: General, Measurement of Temperature, Measurement of Atmospheric Pressure, Measurement of Humidity, Measurement of Surface Wind, Measurement of Precipitation, and Measurement of Radiation. This is a vital source that has continued to contribute to our understanding and development of the weather station.

[4] ISO-CAL, "What is a Pyranometer? 10 Important Points to Consider.," isocalnorthamerica.com, Jan. 16, 2023. https://isocalnorthamerica.com/what-is-a-pyranometer/ This website was used to better understand pyranometers. It discussed what exactly solar irradiance is and what it is used for, which are things like climatology and solar power generation. It described that they measure both diffuse and direct sunlight to record the amount of solar energy reaching a surface per unit area. Typically, this measurement is in $\frac{W}{m^2}$. It also briefly mentions the two common types of pyranometers as well as the importance of calibration. It does not discuss directly how to calibrate, as they are trying to sell their calibration services. This source has allowed us to understand pyranometers in simpler terms than discussed in the book mentioned above.

[5] C. Gittins, "Considering the energy consumption of a Raspberry Pi," IOT Insider, Sep. 23, 2024. https://www.iotinsider.com/news/considering-the-energy-consumption-of-a-raspberry-pi/ This website was used to help determine how much power consumption our weather station will have overall. This was used in my first mathematical modelling exercise, which will be discussed in section 3.3 of this report. This website said that a Raspberry Pi will typically use 5W with normal load.

[6] NiuBol, "How much power does a weather station use ? ," Niubol.com, 2024. https://www.niubol.com/Product-knowledge/How-much-power-does-a-weather-station-use.html. This website was also used to complete my mathematical modeling calculations. Here, they talked about how much power weather measuring sensors use typically. It was very low, as each sensor is expected to only use .3W each hour. NiuBol is itself a company that sells weather stations, but it was a useful source for calculation and comparison nonetheless.

[7] US, "National Weather Service," Weather.gov, 2025. https://forecast.weather.gov/MapClick.php?lat=35.19814000000002&lon=-111.65112499999998 (accessed Jun. 18, 2025). I used this source as my benchmark weather station. The weather station I am choosing to be the benchmark is the Flagstaff Pulliam Airport weather station. This weather station acts as a benchmark for both our website design and each sensors' accuracy. This benchmark was already discussed above in section 3.1.

[8] J. Portilla, "Python Bootcamps: Learn Python Programming and Code Training," Udemy, 2019. https://nau.udemy.com/course/complete-python-bootcamp (accessed Jun. 22, 2025). I used this Udemy course to teach me Python. I have no coding experience in Python and needed to start from the beginning. I have continued to use this source as a means to learn how to code and will continue to do so. This project has required a great amount of coding, so this was a vital resource for me.

[9] Raspberry Pi, "Build Your Own Weather Station," Raspberrypi.org, 2017. https://projects.raspberrypi.org/en/projects/build-your-own-weather-station/0 (accessed Jun. 27, 2025). The Raspberry Pi website has been very helpful in the design aspect of this project so far. This website discusses how to make a weather station out of a specific kit that Raspberry Pi sells. We have been using

this as an outline for our own coding in the project, as it discusses the entire set up of a weather station from start to finish. We did not copy it, as we do not have the kit that was used.

[10] Raspberry Pi, "Raspberry Pi Documentation - Getting Started," www.raspberrypi.com. https://www.raspberrypi.com/documentation/computers/getting-started.html (accessed Jun. 27, 2025). I used this website to help me understand how to access and use a Raspberry Pi. When I tried to access the provided device, it unfortunately had a password on it. Luckily, this website discusses a number of workarounds, including a full reset or just replacing the SD card within it.

[11] Raspberry Pi, "Raspberry Pi OS," Raspberry Pi, 2025. https://www.raspberrypi.com/software/ (accessed Jun. 27, 2025). To actually activate the Raspberry Pi after replacing or wiping the SD card, we would need to upload the Raspberry Pi operating system. This website discusses how to do so as well as providing the operating system download and documentation. This has been used to fully access the Pi.

[38] Shilleh, "Beginner Tutorial: How to Connect Raspberry Pi and BME280 for Pressure, Temperature, and Humidity," YouTube, Nov. 20, 2023. https://www.youtube.com/watch?v=T7L7WMHbhY0 (accessed Jul. 30, 2025). This resource is a YouTube video which clearly outlined how to connect the BME280 sensor that we purchased for our prototyping to the Raspberry Pi. It outlines both the connections physically and the code needed to get it to function. I used this in conjunction with source [8] and source [9] to complete the second prototype. Without this resource, we would not have had a functional prototype.

[39] DwyerOmega, "Modular Weather Monitoring and Data Storage Stations," Dwyeromega.com, 2015. https://www.dwyeromega.com/en-us/modular-weather-monitoring-and-data-storage-stations/p/WMS-25-Series?srsltid=AfmBOopea9LGqjBDOoZJngG3fUaw1xYyItpGe5olf4VLMBsv2Ow8Cz9z# (accessed Jul. 15, 2025). This source was used in my mathematical modelling of the storage requirements for our project. Specifically, this was used to be a benchmark and validation for how much storage our station would eventually use. The station outlined in this source was also using an SD card for their storage solution.

[40] P. Keheley, "How Many Pages In A Gigabyte? A Litigator's Guide," www.digitalwarroom.com, 2020. https://www.digitalwarroom.com/blog/how-many-pages-in-a-gigabyte (accessed Jul. 15, 2025). I used this website to assist in my calculations for our storage estimate. This provided me with the conversion information between different levels of bytes. It also discussed exactly what size a byte is. A singular byte, according to this source is equivalent to one character.

3.2.2 Rowan McCullough

[3] World Meteorological Organization, Guide to meteorological instruments and methods of observation., Volume 1. Geneva, Switzerland: World Meteorological Organization, 2008. This resource has been shared between several members of our group as it contains a ton of vital information for our project. I primarily utilized this resource to find and understand the industry standards for weather measurement across all types of sensors we are utilizing. Additionally, it outlines the various subtypes of each of these sensors and their strengths and weaknesses. This source provided me with the equation for a specific thermometer output calculation shown below in section 3.3.

[12] "Tempest Weather Station," Tempest, 2024. https://shop.tempest.earth/products/tempest msclkid=1236cca887f915b3bbd3a4f8024b2f08&utm_source=bing&utm_medium=cpc&utm_campaign=

Bing-Search-B2C-NB-Broad US&utm_term=home%20weather%20station&utm_content=Weather%20Station (accessed Jun. 18, 2025). This source is the purchase page for the Tempest Home Weather Station; one of our benchmarks mentioned previously. It was used to understand the features and components of the weather station. It also includes customer reviews which were helpful in validating the product's performance.

[13] A. Overton, "A Guide to the Siting, Exposure and Calibration of Automatic Weather Stations for Synoptic and Climatological Observations," 2009. Accessed: May 01, 2024. [Online]. Available: https://www.rmets.org/sites/default/files/2019-02/aws-guide.pdf This source is a published paper from a member of the Royal Meteorological Society which outlines the calibration and long-term considerations of various weather sensors. It served as a cross-reference to other calibration research as well as a guide to the potential obstacles to maintaining accurate readings over a long period of time. It also contains a lot of information on data archiving that may be useful to us down the line.

[14] Wisconsin DNR, "Calibration & barometric pressure | | Wisconsin DNR," dnr.wisconsin.gov. https://dnr.wisconsin.gov/topic/labCert/BODCalibration2.html This website from the Wisconsin Department of Natural Resources discusses barometric pressure and the variations in readings based on environmental and elevation factors. Since the pressure data is corrected to altitude, it is important to understand this process when comparing our readings to nearby stations and when calibrating our database.

[15] R. Coquilla, J. Obermeier, and B. White, "American Wind Energy Association," 2007. Accessed: May 16, 2023. [Online]. Available: https://research.engineering.ucdavis.edu/wind/wp-content/uploads/sites/17/2014/03/AWEA-2007-Final-Paper.pdf This research paper was published by graduate students from University of California, Davis. It highlights uncertainty factors when calibrating an anemometer and the various roots of uncertainty that may arise and how to calculate them. It also contains test data from an anemometer calculation performed in a wind tunnel. This was an important resource for our understanding of anemometer calibration and a guideline for my uncertainty calculation self-learning assignment.

[16] "Login - CAS – Central Authentication Service," Udemy.com, 2025. Available: https://nau.udemy.com/course/statistics-intro/learn/lecture/35671972#overview. [Accessed: Jun. 25, 2025] This Udemy course was my primary self-learning resource when refreshing myself of statistics and how uncertainty arises and can be calculated. Not all sections were utilized, just sections three, four, eight, and ten. These gave me an important baseline understanding of statistical processes to help me build a MATLAB program to calculate uncertainty in an anemometer reading.

[17] EngineerItProgram, "Experimental Uncertainty," YouTube, Mar. 18, 2013. Available: https://www.youtube.com/watch?v=xJBta_HWTRc. [Accessed: Jun. 26, 2025] This YouTube video paired with the Udemy course gives me more specific information on how engineers calculate and handle uncertainty in their designs and experiments. This contributed to my self-learning assignment and has helped us handle the inevitable uncertainty that arose in our data down the line.

[18] RDS, Ed., "Certificate for Calibration of Cup Anemometer," SOH Wind Engineering LLC, 141 Leroy Rd - Williston, VT 05495 USA, Jun. 2021. Accessed: Jul. 26, 2025. [Online]. Available: https://www.nrgsystems.com/support/product-support/technical-product-sheets/technical-product-sheet-40 This certificate is a specific calibration test report for the anemometer we own. It was found in the NRG database using the serial number found on the box. It contains useful testing information and calibration constants that were needed for our Raspberry Pi code to produce accurate wind data.

[19] "Certificate of Calibration," NRG Systems, 110 Riggs Rd - Hinesburg, VT 05461 USA, Jun. 2021. Accessed: Jul. 26, 2025. [Online]. Available: https://www.nrgsystems.com/support/product-support/instruction-sheets/nrg-bp60-barometric-pressure-sensor-instructions This source is the calibration certificate for our barometric pressure sensor. It was found in the NRG database using the serial number on the box label. It contains the calibration constants we need for the Raspberry Pi code to produce accurate pressure data.

[20] J. Calandra, "Certificate of Calibration," ESSCO Calibration Laboratory, 27 Industrial Ave Unit #9 - Chelmsford, MA 01824 - 3618 USA, Jul. 2021. Accessed: Jul. 26, 2025. [Online]. Available: https://www.nrgsystems.com/products/met-sensors/detail/nrg-t60-temperature-sensor This source is the calibration certificate for our temperature sensor. It was found in the NRG database using the serial number on the box label for the sensor. It contains calibration constants that is needed for the Raspberry Pi code to produce accurate temperature data.

[21] J. Brooks, "Certificate of Calibration for LI-COR Sensor," LI-COR, 4647 Superior Street - Lincoln, NE 68504 USA, Jun. 2020. Accessed: Jul. 28, 2025. [Online]. Available: https://www.licor.com/products/light/pyranometer This source is a calibration certificate for one of our pyranometers. It was found on the LI-COR website database using the serial number found on the box of the sensor. While we own a more precise pyranometer that was used over this one, it is still important to have the calibration information for this sensor as a backup.

[22] E. Instruments, "Pyranometer EKO MS-60/MS-60S - Instruction Manual Ver. 8," EKO INSTRUMENTS CO., LTD, 111 North Market Street, Suite 300 San Jose, CA 95113 USA, Oct. 2023. Accessed: Aug. 04, 2025. [Online]. Available: https://eko-instruments.com/us/product/ms-60-pyranometer/ This source is the instruction manual for our better pyranometer, since I was unable to locate the calibration certificate. It has helped to better understand the sensor and to make a decision on which pyranometer was used in the final design. The calibration certificate was located through this document.

3.2.3 Shutong Wang

[23] National Instruments, "NI DAQ Sensor Calibration Guide," *National Instruments*, 2021. Accessed: May 10, 2024. [Online]. Available: https://www.ni.com/en-us/innovations/sensor-calibration.html

This technical guide outlines procedures and best practices for calibrating sensors in data acquisition systems, especially using microcontrollers like the NI USB DAQ or Raspberry Pi. It helped us understand calibration drift and recommended frequency of recalibration.

[24] Adafruit Industries, "How to Connect Weather Sensors to Raspberry Pi," *Adafruit Learning System*, 2022. Accessed: Apr. 25, 2024. [Online]. Available: https://learn.adafruit.com/pi-weather-station/overview

This online tutorial provides hardware wiring diagrams and Python libraries for interfacing weather sensors (e.g., DHT22, BMP180) with Raspberry Pi boards. It guided us through the sensor setup and initial data acquisition stages.

[25] M. A. Islam and R. Haque, "IoT-Based Real-Time Weather Monitoring System Using Raspberry Pi," in *Proc. Int. Conf. on IoT and Applications*, 2020, pp. 44–49.

This conference paper presents a complete prototype for an IoT weather station using a Raspberry Pi and discusses methods for storing and visualizing long-term weather data. It helped validate our design choices and inspired our data storage approach.

[26] S. Pandey et al., "Analysis of Weather Monitoring Systems Using Cloud Integration," *Journal of Sensor Networks and Data Communication*, vol. 10, no. 2, pp. 100–105, 2021.

This paper focuses on integrating weather stations with cloud platforms (like AWS and ThingSpeak). It supported our decision to use online dashboards for data visualization and remote monitoring.

[27] Ambient Weather, "Installation Guide for Ambient Weather WS-5000," *Ambient Weather*, 2023. Accessed: May 15, 2024. [Online]. Available: https://ambientweather.com/ws5000manual

This commercial user manual describes sensor placement, calibration, and protective enclosure requirements for outdoor weather stations. It informed our decisions on shielding and mounting components against wind and precipitation.

[28] Texas Instruments, *"Thermal Design Considerations for Enclosures,"* Application Report SPRA953, 2009. Accessed: July 14, 2025. [Online]. Available: https://www.ti.com/lit/an/spra953/spra953.pdf

This technical application note discusses how enclosure material, surface color, and exposure to solar radiation can significantly influence internal temperatures in outdoor devices. It emphasizes the importance of minimizing heat absorption and suggests design strategies such as reflective coatings and sunshades. This resource guided our material and surface treatment decisions for the weather station housing to reduce passive solar heating.

[29] IPC, *"Standard for Determining Current-Carrying Capacity in Printed Board Design (IPC-2152),"* IPC, 2009. Accessed: July 14, 2025. [Online]. Available: https://www.ipc.org/TOC/IPCT-2152.pdf

This industry-standard document outlines methods to estimate internal temperature rise of electronic systems based on their power consumption and thermal environment. It highlights how enclosure design and airflow affect component longevity and performance. We used this reference to assess internal heating risks in our weather station and to support decisions regarding ventilation and spatial layout of sensors and the processor board.

3.2.4 Chenxi Dong

[30] "Printed humidity sensors," Encyclopedia.pub. https://encyclopedia.pub/entry/7493.

This entry provides an overview of the three main types of humidity sensors: capacitive, resistive, and printed. It is particularly useful for distinguishing between their operating principles and basic applications. This resource helped me understand which sensors are best for long-term environmental monitoring. For example, capacitive sensors are known for their balance of accuracy, durability, and response time, making them a prime candidate for our station design.

[31] "Review of Printed Humidity Sensors," MDPI Sensors. https://www.mdpi.com/2079-4991/13/6/1110

This scientific paper provides a detailed technical comparison of printed humidity sensors, including their fabrication methods and sensing mechanisms. While we do not intend to use printed sensors in our designs, this paper provides an important benchmark for understanding cost-benefit trade-offs and short-term use advantages. This is particularly useful in concept evaluation and comparison with resistive and capacitive types.

[32] "Capacitive vs Resistive Humidity Sensors," Encyclopedia.pub. – https://encyclopedia.pub/entry/7493

This section specifically focuses on the comparison of capacitive and resistive sensors. It explains in detail how capacitive sensors work by detecting changes in dielectric constant, while resistive sensors work by tracking changes in the resistance of hygroscopic materials. I used this source when calculating

relative humidity using a capacitance-based formula that assumes the sensor follows linear dielectric behavior.

[33] BOMS Review – Comparison of sensing layers – https://www.ias.ac.in/article/fulltext/boms/045/0238
This resource discusses the materials used in humidity sensor construction and how they affect response time and accuracy. It is very useful when considering calibration strategies, as drift of these materials over time can affect long-term performance. It also validates our decision to consider sensor replacement cycles in our final deployment plan.

[34] "Humidity and Dew Point," National Weather Service. https://www.weather.gov/media/epz/wxcalc/dewpoint_rh.htm
This government-hosted calculator tool is used to model relative humidity and dew point based on temperature and RH data. I referenced it to validate my sensor output calculations. It also provides insight into how the sensor interprets the environmental data into meaningful weather measurements.

[35] Ma, H. et al., "Graphene-Based Humidity Sensors," arXiv. https://arxiv.org/abs/2410.02255
This paper introduces a new class of printed humidity sensors using graphene. While still in the research phase, it demonstrates future directions for ultra-sensitive and flexible sensors, which could potentially enhance smart wearables or high-resolution atmospheric monitoring. I use this to contextualize the current sensor options we are evaluating with options that may become viable in future iterations of our space station.

[36] NOAA Sensor Siting Standards – https://www.weather.gov/coop/standards
The official guide outlines the proper installation protocol for humidity sensors, including height (1.25-2 m above the ground), shielding from direct sunlight, and location relative to vegetation.

[37] WMO Guide to Meteorological Instruments – https://library.wmo.int/index.php?lvl=notice_display&id=12407
This is where I learned about long-term stability, calibration requirements, and acceptable error ranges for humidity sensors. I specifically used its advice to ensure accurate RH data in my automated weather system.

[38] Bosch Sensortec. (n.d.). BME280 Datasheet. Retrieved from https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/

[39] NRG Systems. (n.d.). #40C Anemometer Product Manual. Retrieved from https://www.nrgsystems.com/

[40] NRG Systems. (n.d.). BP60C Barometer, T60C Thermometer, LI-COR LI200R & EKO MS-60 Pyranometers Specifications. Retrieved from manufacturer data sheets.

[41] Adafruit. (n.d.). BME280 Python Library. Retrieved from https://github.com/adafruit/Adafruit_Python_BME280

[42] Python Software Foundation. (n.d.). Python Requests Library Documentation. Retrieved from https://docs.python-requests.org/

[43] SQLite. (n.d.). SQLite Documentation. Retrieved from https://sqlite.org/

3.3 Mathematical Modeling

3.3.1 Power Requirements & Storage – Ian Torp

It was important to estimate the power consumption of the weather station. To do so, we examined what the typical power consumption of both a Raspberry Pi and weather sensors would be

under normal load. A Raspberry Pi typically consumes 5W under normal load [5], whereas each sensor will typically only consume .3W per hour [6]. To figure out how much power would be consumed daily, we need to calculate how many kWh are consumed. There are 24 hours in each day, so we take that and multiply it by both the .3W per sensor and the 5W for the Raspberry Pi. Since we have seven total sensors, the equation becomes:

$$24\text{hrs}*7(.3\text{W}) + 24\text{hrs} * 5\text{W} \quad (1)$$

This means the weather station consuming .1704 kWh per day. This becomes 62.196kWh per year. This is very low power consumption. A simple, small 60 W solar panel is more than enough to support this weather station entirely every day with only 5 hours of total sunlight. A 60 W solar panel will produce about 300Wh per day with only 5 hours of sunlight, which is more than enough to support the station.

The next engineering requirement that was important to examine was the storage requirement. From our QFD, we want this weather station to be able to store enough data to last for at least 4 years. To do so, we must determine how much data will be produced by our sensors and how big a byte is. A byte is one character on a text file [40]. Based on our design, each sensor will produce seven data points per hour. Assuming each piece of data has five significant figures, each sensor creates 35 bytes of data an hour. Since we have seven sensors, the equation is:

$$35 \text{ bytes/hr} * 7 \text{ sensors} * 24 \text{ hrs} \quad (2)$$

This results in 5.88 kB/hr which can then be converted to 2.146 mB per year. Seeing as the weather station will produce less than 2 mB of data a year and that it has a 32GB SD card, the station does not have to worry about storage. Assuming only 25 GB are used for the data storage so that the rest can be processes and code, we estimate that the storage will last at least 13 years. This means we easily reach our 4-year engineering requirement.

3.3.2 Thermometer Output Calculation – Rowan McCullough

Our station is expected to use an Electrical Thermometer (PRT) which measures temperature through the resistance of pure metal. The resistance of pure metals has a nearly linear relationship with temperature across the meteorological range [3]. Different electrical thermometers are categorized based on the metals they use, with the most common being the Pt100. The Pt100 sensor also has sub-categories which are categorized by standard tolerances such as IEC 60751 or ASTM E1137 [3] [13]. These standards define the resistance and sensitivity curves which are used to convert outputs into temperature values. The electrical resistance of a PRT can be represented by the Callender-Van Dusen Equation shown below.

$$R = R_0\left(1 + At + Bt^2 + C(t-100)t^3\right) \quad (3)$$

In this equation, R_o represents the electrical resistance at 0 ºC and t represents the temperature in degrees Celsius. The coefficients A, B, and C depend on the standard specification. Assuming the IEC 60751 specification, R_0=100Ω, A=3.908x10-3 ºC, B=-5.80x10-7 ºC and C=4.27x10-12 ºC when temperatures are below zero and C=0 ºC when above zero. In temperatures above -40 ºC, the C term is negligible and thus the inverse equation for temperature can be simplified to:

$$t = -\frac{A}{2B} + \frac{A}{2B}\sqrt{1 - \frac{4B}{A^2}\left(1 - \frac{R}{R_0}\right)} \quad (4)$$

Therefore, given an output reading of 110Ω, our temperature reading would be approximately 25.34 ºC.

3.3.3 Thermal Management Considerations – Shutong Wang

Parameter	Value
Ambient Temperature Ta	35°C (typical summer)
Solar Irradiance G	800 W/m² (clear sky, noon)
Enclosure Absorptivity α	0.85 (black plastic)
Surface Area A	0.015 m² (top of enclosure)
Conv. Coefficient h	10 W/m²·K (natural convection)
Emissivity ε	0.9

Thermal Durability Analysis

We simulate how the outer surface of a small electronics enclosure heats up under constant solar irradiance of 800 W/m²

Qsolar =α*G*A=0.85×800×0.015=10.2 W （5）

This means the enclosure absorbs approximately 10.2 watts of heat per second from sunlight alone.

Heat Loss via Convection and Radiation

To determine whether the enclosure can stay cool, we must calculate how much heat it can lose to its surroundings. There are two passive cooling mechanisms: natural convection and thermal radiation.

$$Q_{\text{loss}} = Q_{\text{conv}} + Q_{\text{rad}}$$

$$Q_{\text{conv}} = h \cdot A \cdot (T_s - T_a) = 10 \cdot 0.015 \cdot (65 - 35) = 4.5 \text{ W}$$

$$Q_{\text{rad}} = \varepsilon \cdot \sigma \cdot A \cdot (T_s^4 - T_a^4)$$

$$= 0.9 \cdot 5.67 \times 10^{-8} \cdot 0.015 \cdot (338^4 - 308^4) \approx 4.0 \text{ W}$$

$$Q_{\text{loss,total}} \approx 8.5 \text{ W} < Q_{\text{solar}} = 10.2 \text{ W} \quad (6)$$

Under typical summer conditions with direct sunlight, the enclosure experiences a net heat gain of approximately 1.7 W. As a result, the outer surface temperature may rise above 65°C, and internal temperatures could exceed 75°C. This places the electronics near or above their recommended operating limits, posing a risk of overheating without proper thermal management.

Component	Max Safe Temp	Est. Internal Temp	Risk Level
Raspberry Pi 3B	85°C	~75°C	Medium
NRG BP-20 Sensor	80°C (est.)	~70–75°C	Low–Medium
MCP3008 ADC	85°C	~73°C	Low

Through research, sensors can reach 75 degrees Celsius in a short period of time.

Conclusion : The enclosure is at moderate thermal risk under summer conditions. Recommend adding ventilation openings or applying reflective coating to reduce solar absorption.

3.3.4 Humidity sensor output calculation – Chenxi Dong

For capacitive humidity sensors, relative humidity (%RH) is typically determined by the change in capacitance:

$$RH = \frac{C - C_{\text{dry}}}{C_{\text{wet}} - C_{\text{dry}}} \times 100 \tag{7}$$

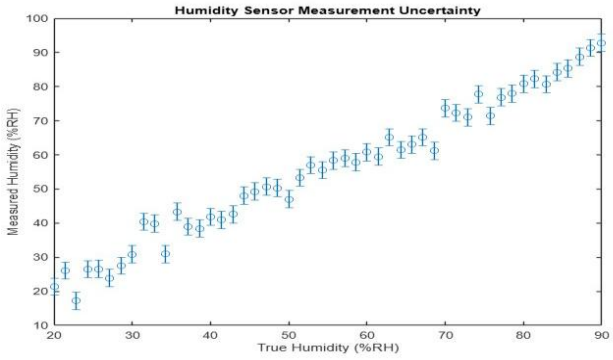
Where:
C is the measured capacitance
C_dry = capacitance at 0% RH
C_wet = capacitance at 100% RH
Example Sensor:
C_dry = 180 pF, C_wet = 250 pF, Measured C = 216pF

$$RH = \frac{216-180}{250-180} \times 100 = \frac{36}{70} \times 100 = 51.4\% \tag{8}$$

Validation was accomplished by comparing the values to the manufacturer's calibration curve. This modeling allowed us to validate sensor sensitivity to meet the sensor tolerance of ±2–3% RH and guided us in performing automated calibration to compensate for drift over time.
Humidity Sensor Uncertainty
Uncertainty Model:

$$U_{total} = \sqrt{U_{cal}^2 + U_{temp}^2 + U_{linearity}^2} \tag{9}$$

Assume a calibration uncertainty of ±2.00% RH. The temperature compensation error caused by a ±5°Cdeviation is ±0.25% RH. Sensor linearity contributes ±1.50% RH. The resulting combined uncertainty is approximately±2.51% RH. 50 humidity readings were simulated between 20% and90% RH. Add random noise according to the total uncertainty. A scatter plot with error bars visualizes the uncertainty in the sensor measurements.



(10)

4 Design Concepts

4.1 Functional Decomposition

The functional decomposition diagram is crucial for our weather station project. It breaks down the complex system into key sub-functions such as sensing, signal processing, data management, data transmission, power management, and structural protection, helping us to clarify how to implement each engineering requirement. This structured approach allows us to more efficiently assign tasks, identify technical challenges in advance (such as analog-to-digital conversion, solar power regulation, etc.), and

ensure the organic integration of subsystems. In addition, the functional diagram also clearly shows the logical path from environmental input to data output, making the design easier to verify and test. The decomposition model also has good modularity, which is convenient for subsequent debugging, upgrades, or component replacement

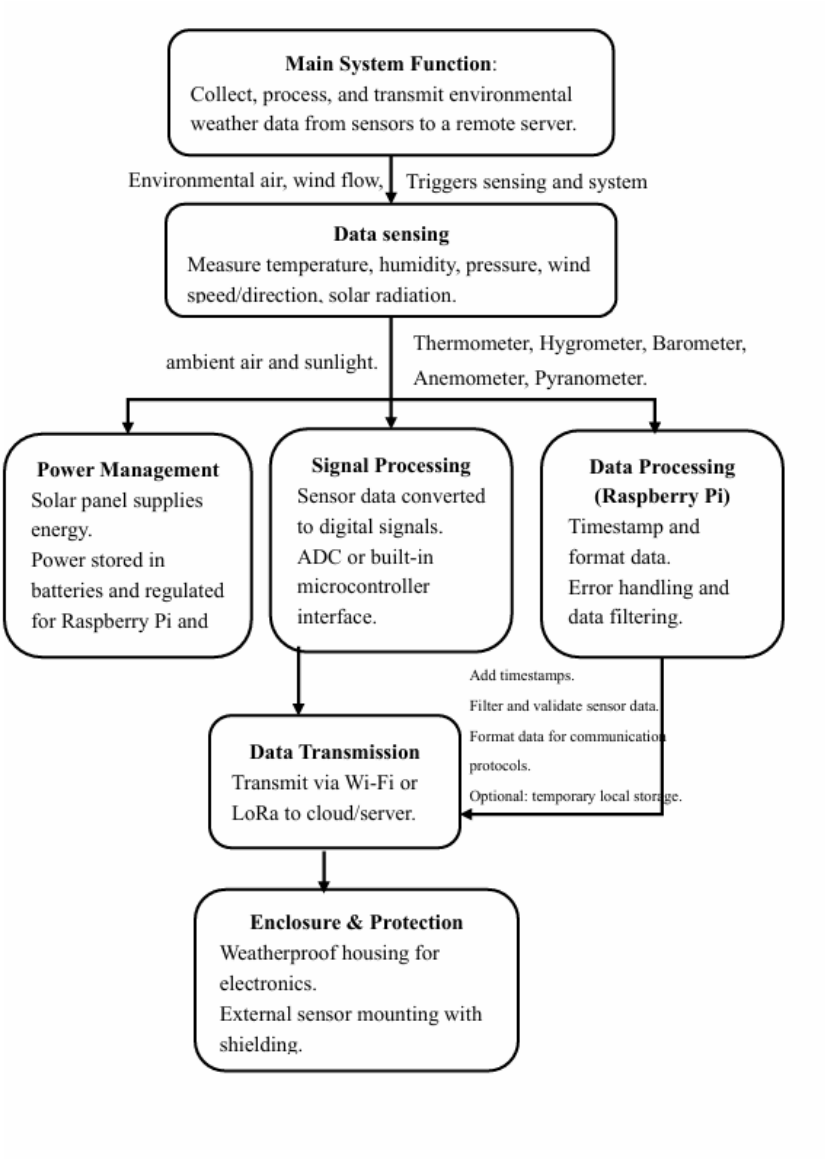


Figure 2: Functional Decomposition

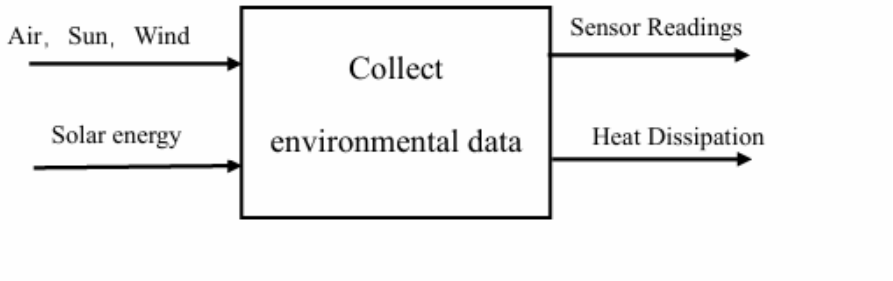


Figure 3: Black Box Model

4.2 Concept Generation

The two main systems which were applicable for concept generation in our project were the structural layout of the sensors and the coding strategy used to handle the data. Each team member designed a concept for both systems. These concepts can be seen below. The concept determined to be middle of the road was chosen as the DATUM for the others to be compared against in the Pugh Chart seen in section 4.4.

4.2.1 Structural Concepts:

DATUM Design (Rowan):

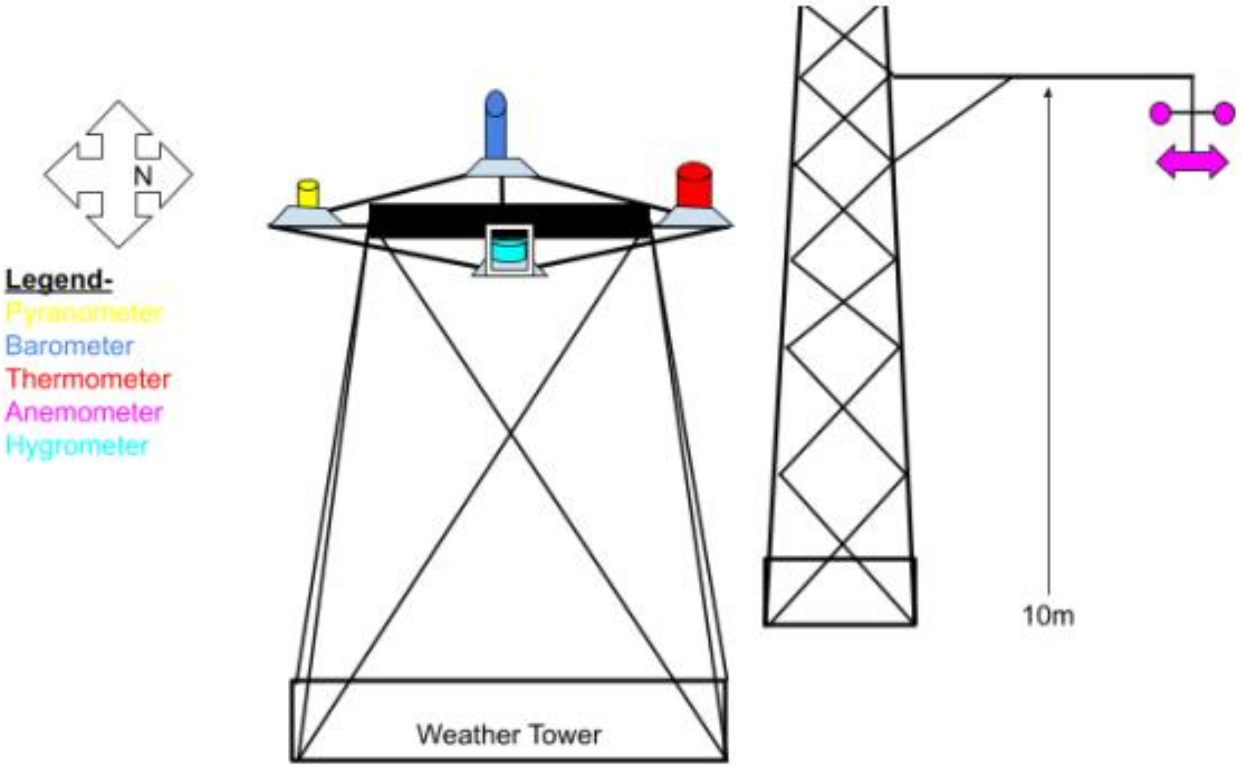


Figure 4: DATUM Design

This structural concept utilizes the small platform which already exists outside the RE Lab as well as the existing anemometer tower. The platform will utilize a diamond shaped crossbar with diagonal

supports mounted on top of the platform. The sensors which are expected to be positioned at this height, which includes all but the anemometer, will be mounted at the vertices to provide three points of contact for stability. The anemometer will be mounted on a boom at the industry standard height of ten meters on the existing tower. Any sensors which need additional housing can be housed within the appropriate material while remaining far enough from the other sensors to avoid interference. The pros of this concept include adherence to industry standards, low cost, and durability. Some cons include potential difficulty building and installing the crossbar and boom and potential safety hazards of the crossbar having corners mounted near head height.

Design 1 (Chenxi):

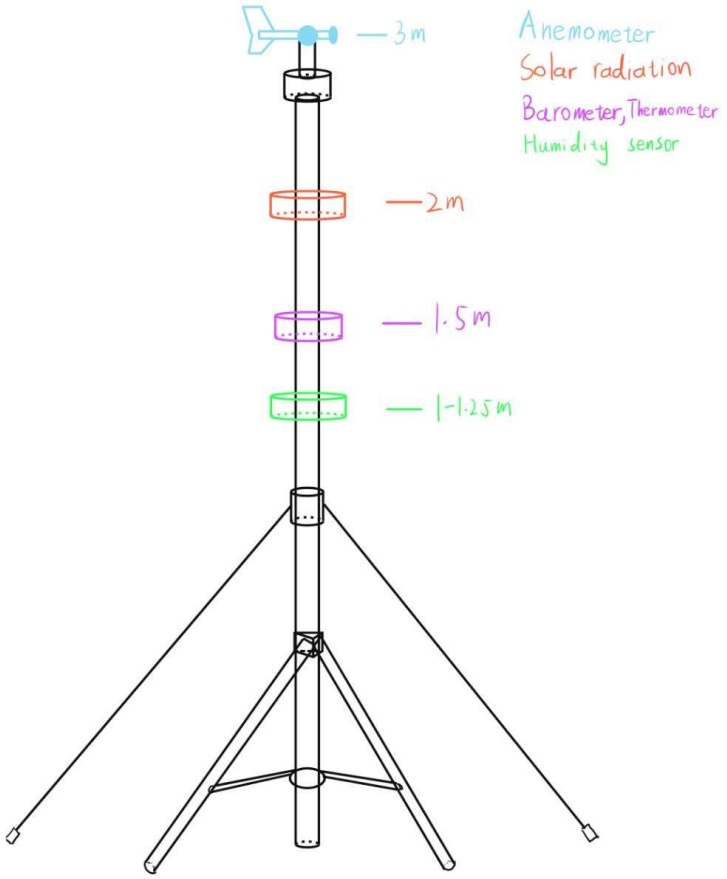


Figure 5: Design 1

The structural concept uses a modular tower structure with a height of about 3 meters, which is reinforced by tripod supports and cables. The sensors are installed in layers to facilitate independent arrangement and reduce mutual interference. The anemometer is installed at the top (3 meters) to ensure unobstructed airflow. The solar radiation meter is installed on the south side at a height of 2 meters to

ensure sufficient sunlight. The barometer and thermometer are installed on the opposite side at a height of 1.5 meters. The humidity sensor is installed at the lowest level (1–1.25 meters) and is placed in a housing with ventilation and radiation shielding to avoid direct sunlight and heat interference. The data cables of all sensors are routed from the inside of the main pole, which is both wind-proof and rain-proof, and also beautiful and neat. The modular structure design facilitates replacement, calibration and transportation. Its main disadvantage is that the anemometer height does not reach 10 meters, which does not meet the industry standard for wind measurement, which may affect the accuracy of the data.

Design 2 (Shutong):

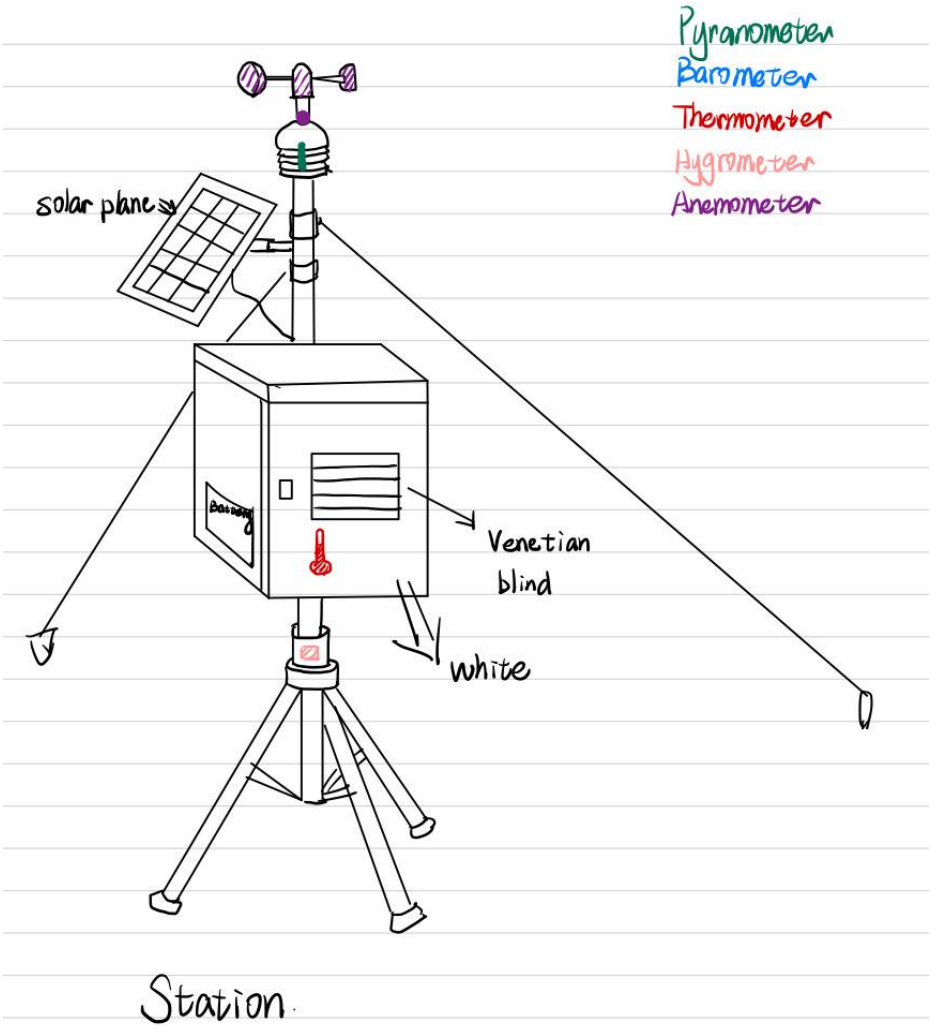


Figure 6: Design 2

This structural concept depicts a compact, integrated weather station built on a stable tripod platform with vertical mounting poles to maximize vertical space for sensor placement. The design consolidates all necessary sensors—including anemometers, wind vanes, pyranometers, and solar panels—onto a central mast. Temperature, humidity, and air pressure sensors are housed within a radiation-shielded Stevenson screen to reduce solar heating interference. The enclosure is also painted white and equipped with shutters to prevent overheating. Advantages of this setup include a small

footprint, ease of deployment, and centralized power/data routing. However, a major disadvantage is that the anemometer height does not meet the industry standard of 10 meters for wind measurement, which may affect data accuracy.

Design 3 (Ian):

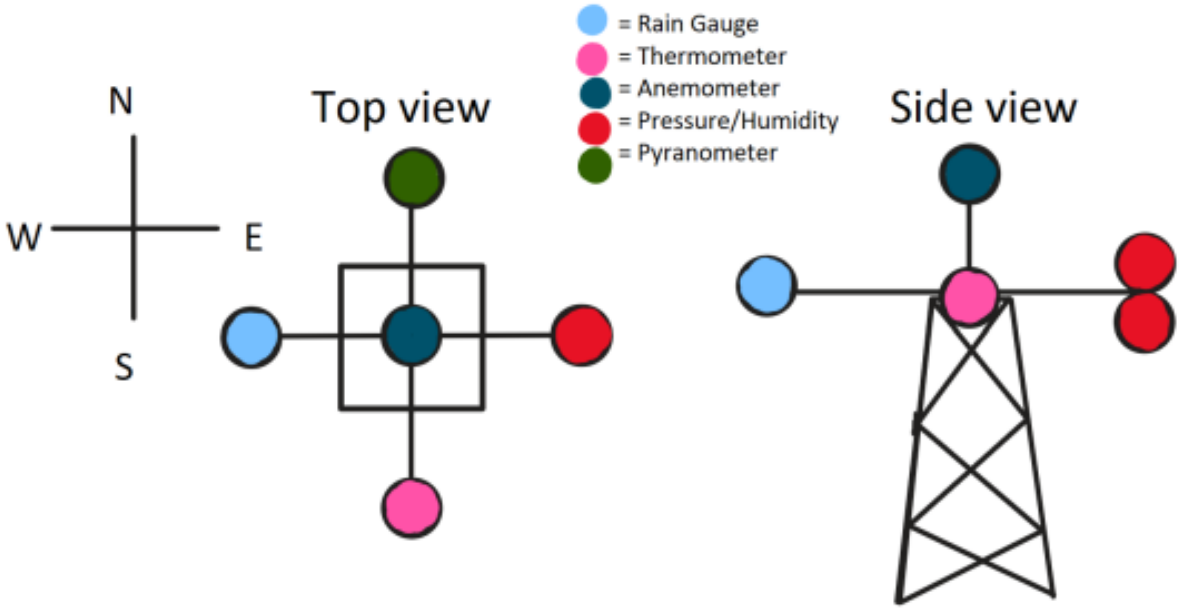


Figure 7: Design 3

This concept is similar to the DATUM in that it utilizes a mounted system atop the existing platform located outside the lab. The major difference lies in the mounting system used for the sensors, with this concept using a crossbar with a sensor located at the end of each rod. The pressure and humidity sensors will be mounted atop each other as they have similar exposure requirements. The major difference is the anemometer placement, with this concept having it placed on a vertical rod directly above the platform. The pros of this concept are similar to the DATUM, with ease of installation being added to the pros. The negatives of this concept include a lack of stability and durability with a single point of contact for each sensor mount and the height of the anemometer not reaching the industry standard of ten meters.

4.2.2 Pseudocode Concepts

DATUM (Ian):

The pseudocode I created was a simple outline describing the processes that would occur within the Raspberry Pi for each sensor. This is visible within Appendix A. The python-based code read the sensor data as it came in and completed the calibration calculations for each sensor accordingly. The idea was for each of these data points to be stored within the Raspberry Pi within respective sections for each sensor. I proposed that this data be uploaded to the website every 5 minutes to provide clear and up-to-date information. I also suggested that the Raspberry Pi should have external storage plugged into it so that we may have a local storage backup of all the data collected for about a year.

This design is simple and would be clear to read as an outside programmer. It would also ensure that we have backup storage for the data in the event that the website has an error occur. Unfortunately, this design would also be more expensive as it would require an external hard drive to be purchased. This

design would require a lot of storage space, as 5 minutes is a much faster upload rate than the Flagstaff Pulliam Airport Weather Station's data upload rate is. This would also entail a higher power consumption than other designs may require.

Design 1 (Chenxi):

I propose a Python-based script: it reads sensor data every 10 minutes and uploads the data to the local database and server synchronously. The data of each sensor is uploaded independently, rather than uploaded together, so that when a sensor fails, it will not affect the data of other sensors. At the same time, after a day has passed, the data of the whole day will be retained in the local database for an additional day to deal with data loss. Another option is to dynamically change the data recording frequency according to environmental changes: when several data are almost the same or the difference is not big, slow down the recording frequency; when the environment changes greatly, speed up the recording frequency.

Design 2 (Shutong):

This design takes a structured approach to processing and visualizing ambient weather data using Python-based tools. Data is collected every 30 seconds from local CSV files or remote cloud sources. Each data input includes a timestamp, temperature, humidity, and air pressure readings. These values are immediately cleaned using libraries to handle missing or invalid entries and ensure consistent datetime formats.

Every 30 seconds, the system adds a new record to a short-term buffer that stores up to 20 calibration readings. Every minute, it calculates a rolling average from these readings to reflect current conditions. This live value is rendered as a live chart using libraries such as or and can optionally be uploaded to a simple dashboard built with tools such as Dash or Streamlit.

In addition to real-time updates, the system also supports hierarchical averaging of long-term summaries. Every 10 minutes, the system calculates an average from the past 10 1-minute values and stores it in a buffer. Once the six 10-minute blocks of data are collected, they are averaged again to produce a final hourly summary that can be stored for historical analysis or downloaded as a PDF report.

Visualizations include:

- A line chart shows temperature over time.
- A bar chart to compare barometric pressure values at different intervals.
- A multi-line chart showing both temperature and humidity.
- Each chart is labeled, and color styled for clarity and automatically updates to reflect the latest readings.

The system is designed to support scheduled updates and an optional dashboard interface. Users can select sensor types or time windows to view specific data sets, and the entire process can be configured to repeat automatically using CRON jobs or time-based triggers in Python.

Design 3 (Rowan):

This design will utilize simple hourly averages calculated from six ten-minute averages for database upload. The sensors will be read every 30 seconds, holding up to 20 readings which are then averaged and sent to the website as a live reading. Every time a new reading is taken, the 20^{th} reading will be deleted and replaced, and every minute a new live reading will replace the one shown on the website.

This will provide real-time, accurate readings for anyone who wants to see the current weather values when viewing our website. After every 20 readings, a separate average will be taken and saved to a buffer which will collect them until one hour has passed. Then an hourly average is uploaded to the database for storage. This will prevent having too much data stored on the Raspberry Pi at one time, while still providing real-time and long-term data to the website. A basic overview of this process can be seen below, while the actual code structure can be seen in Appendix A.

- Every 30 sec:

Add calibrated raw data to a buffer of 20 readings.

- Every 1 min:

Compute rolling average from the last 20 readings.
Upload it to the website as the current "real-time" value.
Store it in the 10-minute buffer.

- Every 10 min:

When 10 1-minute values are collected, average them simply (equal weight).
Add this to the hourly buffer.

- Every 60 min:

When 6 ten-minute blocks are filled, average them to form the hourly summary.
Send to the database.

4.3 Selection Criteria

4.3.1 Structural Criteria

The selection criteria for the structural design were pulled directly from our engineering requirements and customer needs if they were applicable to this system. While we could not do calculations for most of them without a physical part, we discussed as a group to best determine how well each design accomplished them and how heavily they should be weighted in our decision matrix. These criteria included weather durability, safety compliance, low maintenance, easy installation, multiple wind speed readings, and adherence to industry standards.

4.3.2 Pseudocode Criteria

Similar to the structural criteria, these were pulled from our engineering requirements and customer needs if they were related to this system. While our actual code is not quantifiable for many of these criteria, we understand how well they would accomplish each of these goals and were able to determine their completion of each criterion and weigh each criterion for the decision matrix.

4.4 Concept Selection

4.4.1 Structural Concept Selection

Each design was placed into the Pugh chart, with the two top ranked designs moving on to the decision matrix.

Pugh Chart

Criteria	DATUM (Rowan)	Design 1 (Chenxi)	Design 2 (Shutong)	Design 3 (Ian)
Weather Durability	0	-1	1	-1
Safety Compliance	0	1	1	0
Low Maintenance	0	0	-1	0
Easy Installation	0	0	-1	0
Multiple Windspeed Readings	0	0	0	0
Measured at Industry Standards	0	-1	-1	-1
	Totals	-1	-1	-2
	Rank	2	2	4

Figure 8: Pugh Chart

The first step of the concept selection utilized was the Pugh chart, with a DATUM chosen and each design ranked as being better or worse at accomplishing each criterion than said DATUM. A 1 represents being better, while a –1 represents being worse, and a 0 represents being the same. These are then totaled to give a ranking of designs. We had design one and two tied for second, so after team discussion we decided to have design 1 move on to the decision matrix.

		Design DATUM (Rowan)		Design 1 (Chenxi)	
Criteria	Weights	Unweighted Score	Weighted Score	Unweighted Score	Weighted Score
Weather Durability	0.2	75	15	75	15
Safety Compliance	0.1	90	9	100	10
Low Maintenance	0.2	90	18	70	14
Easy Installation	0.15	50	7.5	30	4.5
Multiple Windspeed Readings	0.1	100	10	100	10
Measured At Industry Standards	0.25	80	20	60	15
Sum	1	485	79.5	435	68.5

Figure 9: Decision Matrix

In this decision matrix we assigned weights to each criterion, totaling to 1. We then gave scores to each design on how well they accomplished the criterion out of 100. These scores were multiplied by the weights of each criterion to give each design a score out of 100. The winning design was the DATUM design with a score of 79.5.

4.4.2 Pseudocode Concept Selection

Using the same process as the structural system, each pseudocode design was put into a Pugh chart to determine the two best designs which were then utilized in the decision matrix.

Pugh Chart

Criteria	DATUM (Ian)	Design 1 (Chenxi)	Design 2 (Shutong)	Design 3 (Rowan)
Long Term Data Storage	0	0	-1	1
Increased Data Accuracy	0	1	0	0
Remote Data Access	0	0	1	1
User Friendly	0	-1	0	0
Low Power Requirement	0	1	-1	0
Data Transmission	0	0	1	0
	Totals	1	0	2
	Rank	2	3	1

Figure 10: Pugh Chart 2

After comparing each pseudocode design against the DATUM, design 1 and design 3 were ranked the highest and advanced to the decision matrix.

		Design 1 (Chenxi)		Design 3 (Rowan)	
Criteria	Weights	Unweighted Score	Weighted Score	Unweighted Score	Weighted Score
Long Term Data Storage	0.25	75	18.75	85	21.25
Increased Data Accuracy	0.25	90	22.5	70	17.5
Remote Data Access	0.25	80	20	90	22.5
User Friendly	0.1	60	6	75	7.5
Low Power Requirement	0.1	80	8	80	8
Data Transmission	0.05	90	4.5	90	4.5
Sum	1	475	79.75	490	81.25

Figure 11: Decision Matrix 2

After weights were assigned to each criterion, scores were given to each design on their accomplishment of the criterion on a scale of zero to 100. These scores were then multiplied by the weights of each criterion to give an overall score to each design out of 100. The winning design was design 3 with a score of 81.25.

4.4.3 Final Structural Design

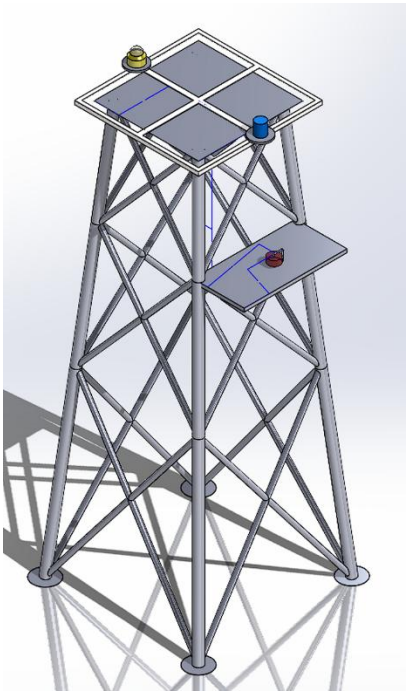


Figure 12: Overall View of the Weather Station Figure 13: Top Crossbar

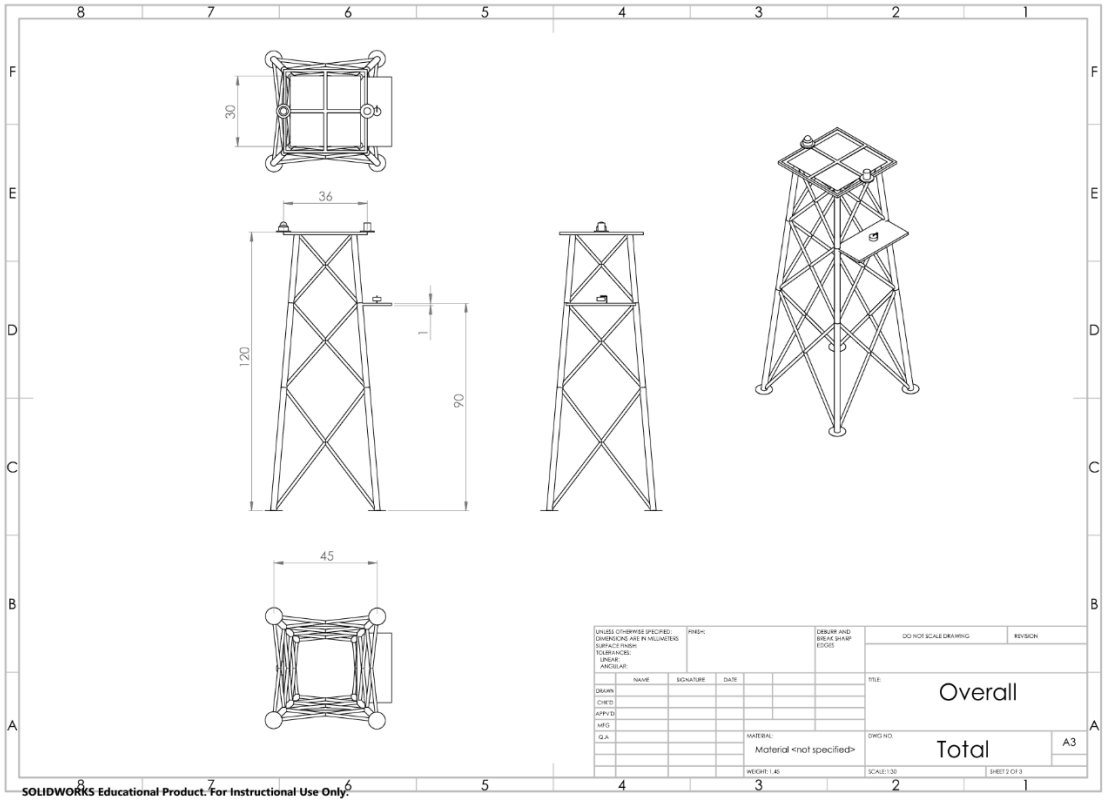


Figure 14: Overall Engineering Drawing with Dimensions

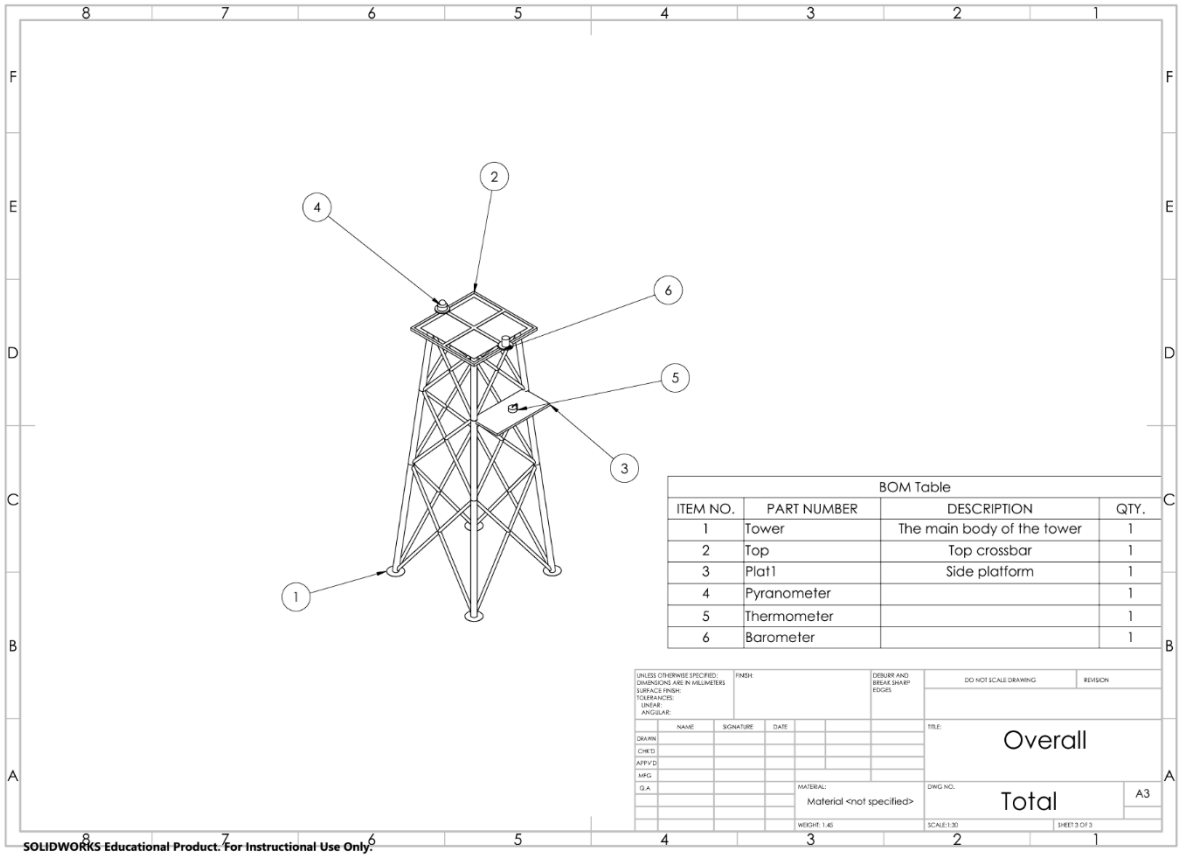


Figure 15: BOM View Drawing

Figure 12 shows the final structural design of the weather station tower. The top crossbar was initially designed to be diagonally positioned (see Figure 4 for details), but due to limitations in aluminum strip length, it was changed to a parallel installation, ultimately serving as a high-mounted sensor mounting platform. Originally, four sensors were planned for the crossbar, but ultimately only the Pyranometer and Barometer were installed in the center around the crossbar, maintaining a good distance to avoid interference. The lower side platform was used to mount the Thermometer to accommodate the height and sensor size. Figure 13 is a close-up view of the top crossbar. Figure 14 shows the complete engineering drawings, including front, side, and top views, with overall dimensions in inches. The total tower height is 120 inches, the crossbar width is 36 inches, the platform height is 90 inches, and the leg spacing is 45 inches. Finally, Figure 15 shows the bill of materials, which includes the tower body, the top crossbar, the side platforms, and the three sensors. Because the team had to change some parts and their installation locations during the manufacturing process, the design differs slightly from the initially selected plan.

5 Design Validation and Initial Prototyping

5.1 Failure Modes and Effects Analysis

Our team's FMEA analysis primarily focused on the wiring connections between sensors and the Raspberry Pi, as that is where the majority of our system failures are likely to arise. The subsystem with the highest risk of a wire disconnection failure is the breadboard, meaning we will need to secure the wire

connections to ensure this does not occur. All wire connections are soldered for the final product. Any wire disconnection will result in a loss of some form of data, meaning it can be easily identified if it were to occur. Another potential issue is snow buildup on the sensors during the winter months, especially the pyranometer since it cannot be housed in any way without interfering with the data. The crossbar should prevent major snow buildup from occurring around the sensors. The condensed FMEA table can be seen below in Table 1.

Part # and Functions	Potential Failure Mode	Potential Effect(s) of Failure	Potential Causes and Mechanisms of Failure	RPN	Recommended Action
Breadboard	Disconnection	Loss of connection	Outside Interference (Human)	35	Secure Connections
Sensors Wired Connection	Disconnection	Loss of connection	Outside Interference (Weather)	14	Secure Connections
PI Wire Connection	Disconnection, Electrical overload	Loss of connection, fried pins	Outside Interference (Human)	16	Secure Connections, no direct connections
Analog to Digital Converter	Disconnection	Loss of analog sensor data	Outside interference, product failure	14	Secure Connections
BME 280 Sensor	Disconnection	Loss of humidity data	Outside interference, product failure	12	Secure Connections, Housing
40C Anemometer	Disconnection	Loss of Windspeed data	Outside Interference, product failure	12	Secure Connections
T60C Temp. Sensor	Disconnection, snow buildup	Loss of temperature data	Outside Interference, Weather	18	Secure Connections, Housing
BP60C Barometer	Disconnection, snow buildup	Loss of pressure data	Outside Interference, Weather	18	Secure Connections, Housing
MS-60 Pyranometer	Disconnection, snow buildup	Loss of solar irradiance data	Outside Interference, Weather	36	Secure connections, Mounting
PI Data Conversion	Coding issue	Invalid data	Bug in coding, unrecognized input data	20	Extensive Code Testing
PI Data Upload	Loss of Internet	Missing data	Internet down, LAN connection failure	16	Database Check, Wired Connection
Truss Supports	Fatigue Failure	Broken Sensors	Poor mounting or durability	9	Secure mounting, strong material

Table 1: FMEA Table for RE Lab Weather Station

The only major risk trade-offs in our design revolve around using a breadboard for the electrical mapping to the Raspberry Pi and any kind of housing we make around the sensors. The use of a breadboard comes with the risk of potential connection loss, as they are not the most reliable in terms of secure wiring. We hope that soldering the wires in place will prevent this, as it is too important of a subsystem to connect all the sensors to the Pi for us to leave it out of the design. We must be careful with sensor housing to not interfere with the sensor reading and continue to adhere it industry standards. Each sensor has its own requirements in this department, and we must analyze the needs of each sensor as such.

5.2 Initial Prototyping

5.2.1 Physical Prototype 1

The first physical prototype aimed to answer the question: How will sensors connect to the Raspberry Pi? While we tried to answer this question with a functional model, we were unable to get a sensor to fully connect to the Raspberry Pi. While researching for this prototype, we learned that we would require an analog to digital converter to adjust the signal from our analog sensors into a digital signal that the Raspberry Pi could read. We were unable to secure this device before the deadline, so our first physical prototype was unable to answer the question we sought after. However, this did not mean we failed entirely. We were able to learn that we need analog to digital converters for our sensors to work in our system. We chose to go with MCP3008 converters as those are largely the most acceptable.

5.2.2 Virtual Prototype 1

While our physical prototype failed, our virtual prototype was a success. We wanted to answer the question: How will the data be interpreted internally and what can that data be displayed as? So, we created code for the Raspberry Pi that would take example samples (as we were unable to connect a sensor) and then interpret that data. The code created was fairly simple and could be used as an outline for future sensor coding. We found that with simple equations, we were able to translate the input signals,

which were pings based on rotation, and convert them into Hz and then m/s and mph. These results were then placed into a graph depicting the different signals received once they had been fully evaluated. With the success of this virtual prototype, we were able to prove that the Raspberry Pi could handle the calculations and will let us interpret the data from our sensors. Figures 16 and 17 show the results of this prototype.

```
== NRG #40 Anemometer Manual Test ==
1. Enter pulse count and sample time
2. Or directly enter frequency (Hz)
Select mode (1 or 2): 2
Enter frequency (Hz): 4
Estimated wind speed: 3.41 m/s
```

Figure 16: Direct Code Output

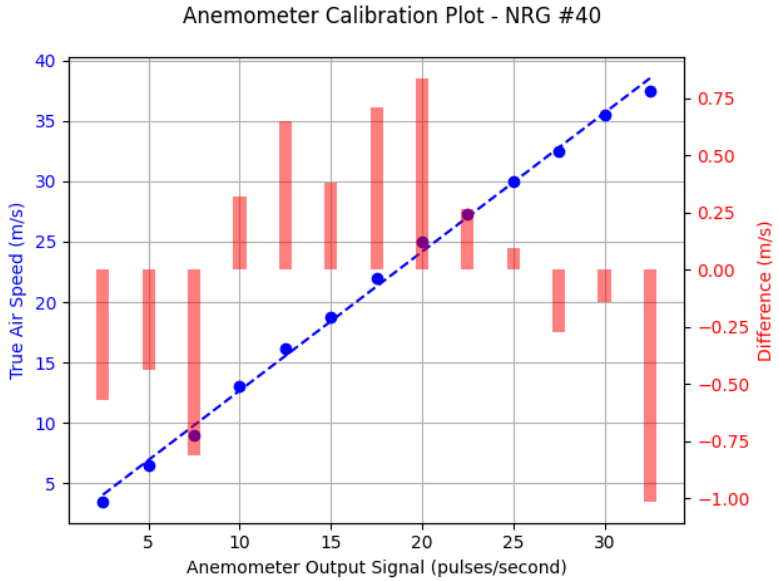


Figure 17: Graph Generated from Sample

5.2.3 Prototype 2

This prototype was fully successful in both the physical and virtual portions. For this prototype we wanted to answer two questions: How will sensors connect and interface with the Raspberry Pi, and how will we interpret live signals from three sensors into readable real-time data? With the use of a BME280 sensor, a digital sensor that reads temperature, humidity, and pressure, we were able to answer both of these questions. After researching how the connections worked [9] and learning how to directly interact with the sensor [38], we got results. With a soldered connection between the wires and the BME280, and each wire sent to the correct GPIO pin, we were able to receive live data. Figure 18 below shows what this live data was displayed as and Figure 19 shows the BME280 sensor with its connections. This prototype allowed us to have 3 fully functional sensors displaying live data to our Raspberry Pi. From here, we know that our Raspberry Pi works properly and that our code is successfully running without issue. This prototype outlines the rest of our project, as it contains everything we need to fully create our

weather station. With our sensors successfully integrated and our Raspberry Pi successfully interpreting live data, the only steps we have left are to establish this same connection with the other sensors.

Figure 18: Real-Time Data from Prototype 2

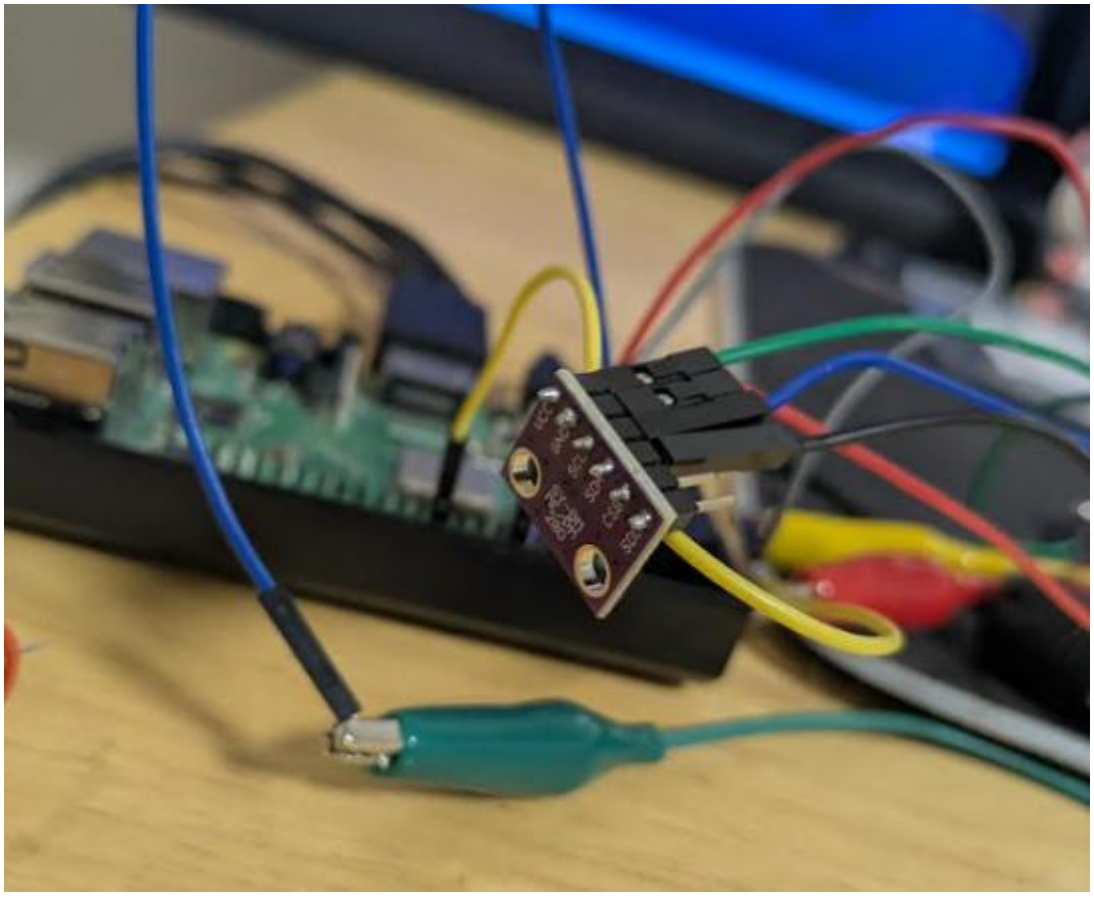


Figure 19: BME280 Sensor for Prototype 2

5.3 Other Engineering Calculations

Several more calculations relating to the design have been performed since the concept selection phase. One example is the calibration math for each sensor we own and is to be used in the final design. These sensors are brand new and have specific calibration certificates associated with their serial numbers. These certificates provide the exact calibration constants and relationships which relate the sensors output to the data they measure. While we will need to further alter these outputs for the Raspberry Pi to interpret them, these constants are still a vital piece of the code that will be used to process this data. An example of the information found on these certificates can be seen below in Figure 20.

Reference Pressure (hPa)	Sensor Output (V)	Slope (hPa/V)	Offset (hPa)	Linearity R^2
552.130000	0.236063	243.8566	494.6673	1.000000
756.910000	1.074680			
1048.660000	2.272093			

Figure 20: Calibration Factors for BP60C Barometric Pressure Sensor S/N: 9396000036

Table 2 below shows the summary table outlining all the engineering calculations that have been completed to date. This table shows who performed each calculation as well as its focus, result, validation, and which engineering requirement it applies to.

Table 2: Summary Table of Mathematical Modelling

Calculations				
Team Member	Focus	Result	ER	Validation
Ian Torp	Power	.0174 kwh/day	<0.2 kwh	NiuBol website
Rowan McCullough	Thermometer	$t = -\frac{A}{2B} + \frac{A}{2B}\sqrt{1 - \frac{4B}{A^2}\left(1 - \frac{R}{R_0}\right)}$	Calibration	IEC 60751
Chenxi Dong	Humidity	$RH = \frac{216-180}{250-180} \times 100 = \frac{36}{70} \times 100 = 51.4\%$	Calibration	Manufacturer
Shutong Wang	Heating	ΔT=P * R_θJA, Q_solar =G*A*α	Safety	Same product, measurement
Ian Torp	Storage	At least 9 years	4 years	Checked against similar products
Rowan McCullough	Anemometer Uncertainty	Within 3 %	Accuracy	Using random data
Chenxi Dong	Humidity Uncertainty	±2.51% RH	Accuracy	Simulation using model
Shutong Wang	Wind direction	225, 100.02, 34.56	Accuracy	Simulation using model
Ian Torp	Prototype Function	N/A (coding)	Fucntionality	Prototype
Rowan McCullough	Sensor Calibration Functions	Linear Relationships specific to S/N	Calibration	Certificates of Calibration
Chenxi Dong	Sensor life calculation	$\frac{60}{30} \times 60 \times 24 \times 365 = 1051200\ samples/year$	Maintenance	Manufacturer
Shutong Wang	Thermal Durability Analysis	Qsolar =α·G·A, Q_loss =hA(T−T a)+εσA(T^ 4−T a^4)	safety	Same product, measurement

6 Project Management

6.1 Schedule

The following figure, Figure 21 shows a simplified version of the Gantt chart that the team used for the first semester. In it, each major deadline is depicted as well as who oversaw each portion. While this chart is simplistic, it outlines what the team did as a whole throughout the semester. It is important to note that the general responsibilities of each member are as follows: Ian Torp handled the coding overall as well as

the Bill of Materials, Rowan McCullough worked on calibration math and was the original financial manager before that changed hands to Ian Torp, Shutong Wang was in charge of all electrical components and connections, and Chenxi Dong was the CAD manufacturer.

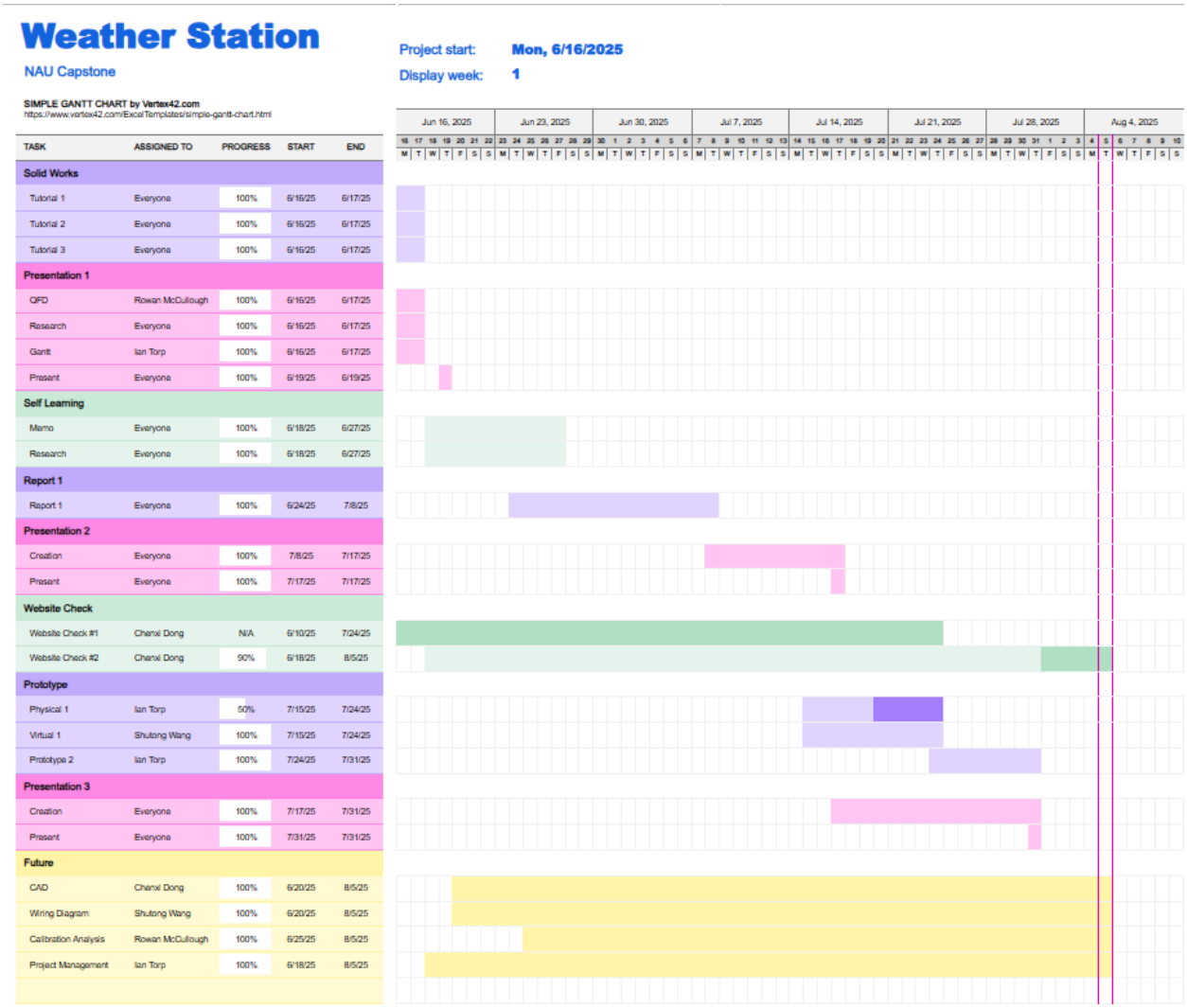


Figure 21: Gantt Chart Summer Semester

As shown above, the team was able to complete most of the tasks without issue. The two incomplete tasks are the first website check, which was cancelled by the professor, and the first physical prototype. That prototype was unable to be completed as discussed previously. The first portion of the chart shows the individual SolidWorks task each member was assigned to finish, which we all did on time. Our first big group assignment was our first presentation. Here we had to present the initial research as well as our project as a whole to the class. Following this was our first report. Everyone worked equally on that report and our contributions are clearly labeled within it. Then came the second presentation, in which we were getting ready to make a prototype of our design. Those prototypes are outlined in a later section of this report. Following these were presentation 3, in which we also presented our second prototype. This was the final presentation of the semester, where we discussed our completed design. The following figure, Figure 33 shows the Gantt chart used for the second semester of work.

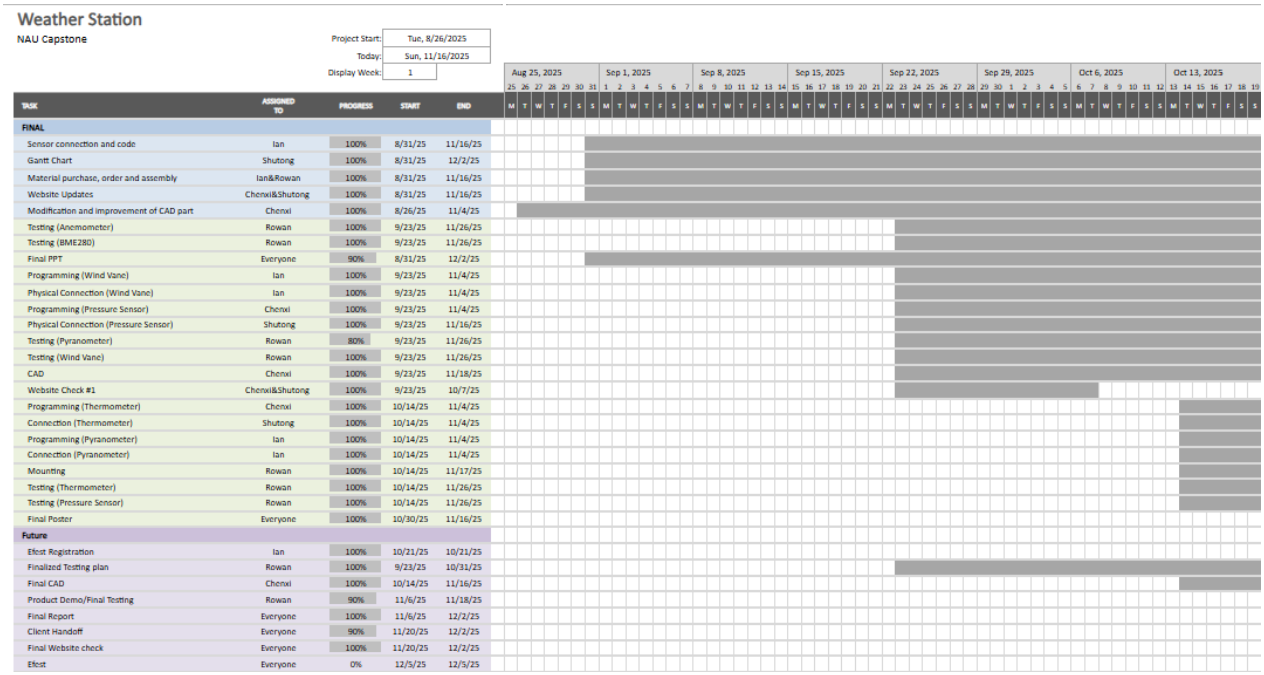


Figure 22: Gantt Chart for the Current Semester

As shown above, the Gantt chart outlines the updated timeline for the weather station project during the fall semester. Each item has been completed to the best of the teams ability within this time frame. The only remaining tasks are the client handoff and the Efest presentations. One may note that the pyranometer testing is not yet complete. This is due to hardware hiccups that are being worked on as of the writing of this report. The current progress has been approved by our client as a finalized product, but the team aims to finish this last portion before handing the project off.

Each line depicts who worked on what sections and their relative timeline for completion. As seen with the status column, almost every task has reached 100% completion. The client and the team are happy with the progress made within the allotted time for the project.

6.2 Budget

The team's budget is built around the original estimate of $3100 that the client provided to the team. This $3100 covers the cost of the sensors as well as the Raspberry Pi. On top of this initial investment, some of the primary purchases made by the team include the MCP3008 analog to digital converter, the crossbar aluminum, an electrical box to house the PI, and the BME280 sensors. The team was expected to gather $300 from fundraising efforts. While the team failed to collect sponsors, they each committed $75 to reach that $300 goal. The client, Professor Pete, has also pledged an additional $500 for any unforeseen costs that arise down the line. This gives us a total budget of around $3900, which we should not reach, as outlined in our current Bill of Materials.

6.3 Bill of Materials

The following Bill of Materials (BOM) showcases the cost of our weather station in total. It covers all expenses. While the Raspberry Pi, wires, and sensors were all given to us by the Client, it is important to note that they totaled somewhere in the ballpark of $3100. Once we started prototyping, we discovered that we needed to purchase MCP3008 analog to digital converters and BME280 sensors to achieve our goals. In total, the team only spent $36.36 the first semester and a total of $387 by the end of the project. Our Bill of Materials includes the material, quantity, cost, part status, primary vender, manufacturer, lead time, link to the product, and whether we would make or buy each part that is in the BOM.

Material	Quantity	Cost	Part Status	Make/Buy	Primary vender	Manufacturer	Lead Time	Link to Product
Onn MicroSD card 32gig	1	6.96	Arrived	Buy	Walmart	Onn	Same Day	https://www.wa
MCP3008 Analog to Digital Converter	2	15.3	Arrived	Buy	Amazon	Bridgold	Two Day	https://www.am
BME280 Sensor	2	14.1	Arrived	Buy	Amazon	Qoroos	Next Day	https://www.am
Model MS60 Pyranometer	1		In house	Preowned	NRG	EKO	N/A	https://eko-instr
T60C Temperature NRG Sensor	1		In house	Preowned	NRG	NRG	N/A	https://www.nrg
Wind Vane NRG #200P	1	3100	In house	Preowned	NRG	NRG	N/A	https://www.car
NRG #40C Anemometer	1		In house	Preowned	NRG	NRG	N/A	https://www.nrg
NRG BP60 Barometric Pressure Sensor	1		In house	Preowned	NRG	NRG	N/A	https://www.nrg
Raspberry Pi 3 model B+	1	35	In house	Preowned	Mouser	Raspberry Pi	N/A	https://www.mo
Wire/cabling	1	22c/ft	In house	Preowned	Home Depot	Syston	N/A	https://www.ho
3/4 emc tubing	18	N/A	In house	Preowned	N/A	N/A	N/A	N/A
36/1/(1/16) Angle Gage Aluminum	4	34.32	Arrived	Buy	Home Depot	Everbilt	Same Day	https://www.ho
1.5/36/(1/8) Flat Bar Aluminum	2	27.28	Arrived	Buy	Home Depot	Everbilt	Same Day	https://www.ho
2' Boom Mount	2	N/A	In house	Preowned	N/A	N/A	N/A	N/A
Voltage Amplifier (AD620)	1	13.23	Arrived	Buy	Amazon	Midzooparts	6-10 Day	https://www.am
Mounting Hardware	4	14.19	Arrived	Buy	Home Depot	Everbilt	Same Day	https://www.ho
EKO Converter Cable	1	104.2	Arrived	Replace	EKO	EKO	4-6 Day	https://eko-instr
Monitor	1	37.42	Arrived	Buy	NAU Surplus	Dell	Same Day	N/A
Backup Raspberry PI 4B	1	85.42	Arrived	Buy	ADA Fruit	Sony	4-6 Day	https://www.ada
Steel Fish Tape	1	29.97	Arrived	Buy	Home Depot	Klein Tools	Same Day	https://www.ho
UV Resistant Zip Ties	40	12.58	Arrived	Buy	Home Depot	CE	Same Day	https://www.ho
Hose Clamps	6	14.79	Arrived	Buy	Home Depot	SS	Same Day	https://www.ho
8x8x4 Electrical Enclosure Box	1	40.02	Arrived	Buy	Home Depot	Southwire	Same Day	Southwire 8 in. V
	Total Cost:	3487.42	100% ordered					
			100% Arrived					

Figure 23: Finalized Bill of Materials

6.4 Manufacturing Plan

Due to the nature of the project, there was very little manufacturing involved with our final design. The primary aspects of the project which could be considered manufactured, such as the construction of the crossbar, are shown in the following table.

Table 3: Manufacturing Plan

Manufacturing Items	Status	Completion Date
Crossbar Aluminum	Assembled	7-Oct
Crossbar Hardware	Assembled	12-Oct
Sensor Mounting	Assembled	3-Nov
Long-Term Outdoor Wiring	Assembled	3-Nov
EKO Converter Cable	Assembled	4-Nov
Boom Mounts	Assembled	3-Nov
Indoor Electrical Housing Box	Purchased	N/A

7 Final Hardware

7.1 Sensors on Crossbar

Figure 24: Pyranometer Mounted on Crossbar

Figure 24 depicts the EKO MS-60 pyranometer mounted on top of the crossbar for our weather station. This sensor detects solar irradiance and sends voltage signals to our Raspberry Pi. This sensor is attached to an amplifier as it sends microvolt signals which the Raspberry Pi cannot read on its own. With continued testing, we hope to improve the accuracy of the readings being sent through this sensor.

Figure 25 below depicts the barometric pressure sensor and temperature sensor which are also mounted atop this tower. Our testing has concluded that these two sensors are working nominally and outputting readable, scientific data. The website displays both sensors' outputs for the last ten minutes and the last 24 hours. The barometric pressure sensor has been adjusted to sea level to output more comprehensible readings.

Figure 25: Mounted Barometric Pressure (top) and Temperature Sensor (bottom)

7.2 Sensors Mounted on Large Tower

Figure 26: Anemometer and Wind Vane Mounted at 30ft

Figure 27: Anemometer Mounted at 90ft

Figures 26 and 27 above depict the entirety of our wind data collection system. At the lower height of roughly 30 ft, a wind vane and anemometer were mounted and actively collecting data and uploading it to the website. These two sensors were mounted by our team after testing and soldering their connections. The anemometer depicted in figure 27 was mounted before our team was assigned this project, so we tapped into its output as per our client's request. The two varying heights of wind data collection allow us to have a more robust data set.

7.3 Raspberry Pi Housing

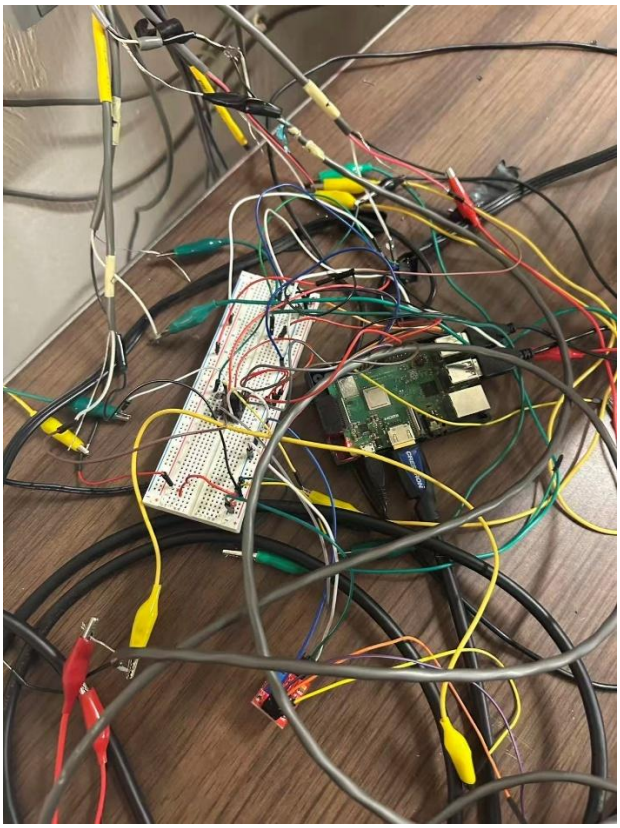


Figure 28: All cables for the Raspberry Pi

Figure 29: Raspberry Pi Housing

Finally, all the wires, breadboard, and Raspberry Pi will be placed inside this box (Figure 29). Along with the Raspberry Pi, the BME280 will be inside this housing recording humidity data. This sensor is depicted in Figure 19 and above in Figure 28.

This will be completed as the finalized pyranometer testing is implemented within the next few days. Below in Figure 30 is the wiring diagram outlining where each wire and sensor is placed. Almost every sensor is located on the left with additional points of interest being the AD623 microvolt amplifier (leftmost box), the MCP3008 Analog to Digital Converter (second box to the left), the Raspberry Pi 3+ (second box on the right), and the BME280 humidity sensor (rightmost box).

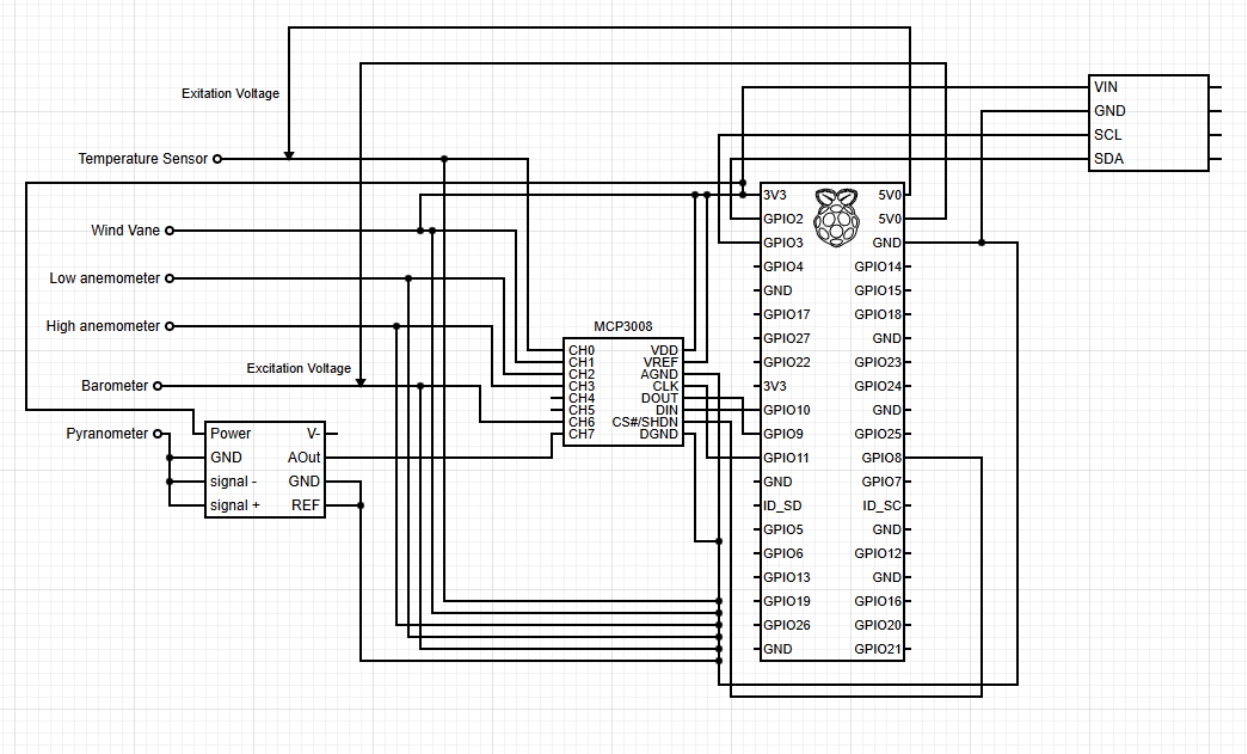


Figure 30: Wiring Diagram for all Connections within the System

8 Testing

8.1 Testing Plan

A total of seven experiments were conducted during the testing phase. These seven experiments are listed in the testing summary table below. Each experiment aims to confirm the project's accomplishment of specific engineering and customer requirements. Resources needed to complete these tests are also listed in the table. No new purchases were required during testing.

Test:	Requirements Satisfied:	Equipment Needed:	Other Resources:
EX1 - Anemometer Calibration Test	CR1, ER2, ER3, ER5, ER6	Fan, Tripod, Handheld Anemometer, Tachometer	Manufacturer's Calibration Certificate
EX2 - Barometer Data Comparison	CR1, ER2, ER4, ER5, ER6	N/A	Pulliam Airport Weather Database
EX3 - Pyranometer Test	CR1, ER2, ER4, ER5, ER6	Black Box	Pulliam Airport Weather Database, Sunny Day
EX4 - Temperature Sensor Calibration Test	CR1, ER2, ER4, ER5, ER6	Platinum RTD Sensor	Pulliam Airport Weather Database
EX5 - Wind Vane Calibration Test	CR1, ER2, ER4, ER5, ER6	Compass, Ruler, Paper	N/A
EX6 - Boom Mount Stress Test	CR5, CR8, CR10, ER3, ER4	Weight, Strap	N/A
EX7 - Weather Database Test	CR2, CR3, CR7, CR11, ER1, ER6	Raspberry Pi	NAU Wi-Fi

Figure 31: Testing Summary

8.2 Testing Details

The specific plan and expected outcomes for each test are outlined in this subsection.

EX1 - Anemometer Calibration Test

The goal of this test is to ensure the new anemometer is properly calibrated and produces accurate data. Essentially we will be ensuring the calibration certificate from the manufacturer, including the calibration equation, still upholds. In order to perform this test we will need a fan and tachometer, which can be borrowed from Thermo Fluids Lab 111. The rotational speed of the anemometer will be adjusted using the fan settings and measured using the tachometer. This rpm measurement will be plugged into the calibration equation to ensure the voltage output reading results in the same wind velocity measurement. A handheld anemometer will be equally positioned to our sensor and the wind speed measurements compared.

Steps:

1) Position fan at a distance of 1ft from anemometer
2) Record Wind Speed reading output from Raspberry Pi voltage conversion
3) Record rotational speed in rpm from tachometer
4) Convert RPM into wind speed based on geometry of anemometer
5) Compare measured and calculated wind speed
6) Measure windspeed from handheld anemometer at 1ft distance from fan
7) Adjust Calibration equation if needed until wind speeds are equivalent

Results: Outputs from converted Raspberry Pi voltage signal reading (read as Hz) using calibration equation 1 should be within 3% of converted RPM reading using equation 2.

$$v\ [\text{m/s}] = 0.75776 \cdot f\ [\text{Hz}] + 0.39322 \quad (10)$$

$$v\ [\text{m/s}] = 0.75776 \cdot (\text{RPM})/60 + 0.39322 \quad (11)$$

EX2 - Barometer Data Comparison

The goal of this test is to ensure the readings from the barometric pressure sensor are within a reasonable range. This range will be defined by the Flagstaff Pulliam Airport barometric pressure data since this reading should not vary much from our testing site to the airport. Since we do not have the means to create a controlled pressure environment to test the sensor, we will have to settle for ensuring the readings match the airport data. The variable measured will be the barometric sensor reading using the manufacturer stated calibration equation.

Steps:

1) Take reading from the barometer output using the manufacturer stated calibration equation.
2) Compare to Flagstaff Pulliam Airport's most recent barometric pressure reading
3) If our data is not within 3% of airport data, adjust the calibration equation and begin again from step 1.

Results: Output from converted Raspberry Pi voltage signal should be within 3% of most recent Flagstaff Pulliam Airport barometric pressure reading.

EX3 - Pyranometer Test

The goal of this test is to ensure the readings from the pyranometer are within a reasonable range. This range will be defined by the Flagstaff Pulliam Airport solar irradiance data since this reading should not vary much from our testing site to the airport. Since we do not have the means to create a controlled solar irradiant environment to test the sensor, we will have to settle for ensuring the readings match the airport data. The only control test we can perform is a black box test in which no solar irradiance should be measured. The variable measured will be the solar irradiance reading using the manufacturer stated calibration equation.

Steps:

1) Take reading from the pyranometer output using the manufacturer stated calibration equation.
2) Compare to Flagstaff Pulliam Airport's most recent barometric pressure reading.
3) If our data is not within 3% of airport data, adjust the calibration equation and begin again from step 1.
4) Cover the sensor with black box

5) Ensure zero solar irradiance is measured.

Results: Output from converted Raspberry Pi voltage signal should be within 3% of most recent Flagstaff Pulliam Airport solar irradiance reading. Zero solar irradiance should be measured from black box test.

EX4 - Temperature Sensor Calibration Test

The goal of this test is to ensure the temperature readings from the temperature sensor are within a reasonable range and the calibration equation is correct. The first check will be a comparison of data to Flagstaff Pulliam Airport temperature data. The ambient temperature at the RE Lab may be slightly different than that of the airport due to shade from trees and cloud cover. An RTD will also be used to check our site-specific temperature. The converted temperature reading from the raw voltage output should be within 5% of their measured air temperature with the RTD and within 5% of the airport reading on a day with consistent conditions (sunny, low wind).

Steps:

1) Record converted temperature sensor reading from manufacturer stated calibration equation with sensor placed inside SRP controlled temperature environment.

2) Let RTD reach steady state and compare reading to sensor output
3) Adjust calibration equation and repeat from step 1 until temperature reading is within 3% of controlled box temperature.
4) Compare temperature sensor reading in standard mounting position (ambient, 6ft off ground).
5) Compare to Flagstaff Pulliam Airport's most recent temperature data

Results: Output from converted voltage signal should be within 3% of controlled box air temperature. Temperature data should be within 5% of airport data on a day with consistent weather conditions (sunny, low wind).

EX5 - Wind Vane Calibration Test

The goal of this test is to ensure that the calibrated degree output from the wind vane voltage signal is accurate. The wind direction will not consistently match that of the Pulliam Airport data due to obstructions such as trees and buildings near the RE Lab which will disrupt the wind flow. Thankfully, there exists a simple way to test the calibration of the wind vane output. Using a compass and a piece of paper, a 360 degree circle will be drawn on the paper with cardinal directions marked and degree increments of 30 degrees marked. The wind vane will be adjusted to each of these incremental marks and the output will be recorded to ensure the degrees of rotation are the same.

Steps:

1) Sketch circle on paper and mark cardinal directions and increments of 30 degrees around the circle.
2) Align wind vane zero point (north) with north mark on paper. Adjust wind vane to align with

each cardinal direction and 30 degree increment and log output at each mark.

3) Ensure wind vane output matches 30 degree increments on paper at each point.
4) If output does not match the degree mark on paper, adjust the calibration equation and start over from step 2.
5) Ensure degree reading resets to zero after passing 359.9 degree point

Results: The degree output from the converted voltage signal through the Raspberry Pi should match the degree mark at which it was taken within 1 degree. These alignments will not be perfect as they will be done by eye, but the outputs should be extremely accurate. The output must reset to zero once the 359.9 degree point is crossed.

EX6 - Boom Mount Stress Test

The goal of this test is to ensure the mounting strategy for attaching the boom mounts to the tower located at the RE Lab. These booms are designed for the mounting of the anemometer and wind vanes, so their attachment to the boom is not at risk of failure. The stress test will ensure the hose clamps used to mount the boom to the tower are capable of keeping the boom in place under heavy wind conditions. This will be tested by attaching weights on a rope to the end of the boom and swinging them to replicate heavy wind gusting.

Steps:

1) Attach boom to tower 3 hose clamps at a height of ~5ft
2) Thoroughly tighten hose clams until the boom is as secure as possible
3) Tie rope with weights to the end of the boom rod.
4) Swing and move weights in semi-random patterns to simulate heavy wind gusting
5) Ensure the boom remains secure in place and hose clams do not loosen.
6) If any major movement or failure of the hose clamps occurs, add additional mounting clamps or adjust mounting position then repeat steps 1-5.
7) Repeat weight swinging trial to ensure the first trial did not lead to a weakening of the boom secure mounting.
8) Repeat steps 1-7 with second boom arm

Results: The boom arm is permitted to have slight movement during weight swinging but must remain equally secured as it was before the test. No loosening of the hose clamps are permitted. No bending of the boom arm is permitted. The second weight trial must maintain a similar result as the first to ensure fatigue does not lead to failure.

EX7 - Weather Database Test

The goal of this test is to ensure the weather database is working correctly. This includes timely upload of most recent data, proper display of past data, proper display updates when new data comes in, proper

labeling of data and no gaps or cut-outs in data collection. No tools or materials will be needed to perform this test.

Steps:

1) The Raspberry Pi data collection is activated, and data from all seven sensors is first saved to a local database. Then, a separate program extracts the data from the database and exports a .json file to the NAU website folder for display. Simultaneously, the database .db file is uploaded to the NAU website folder daily as a backup.
2) Leave the system running, checking data daily.
3) After one week, check for missing data points and compare data points to Flagstaff Pulliam Airport weekly data.
4) If data is not uploaded consistently and/or is outside a reasonable range to airport data, troubleshoot cause of error.

Results: Data from all 7 sensors should be uploaded consistently at planned intervals. No gaps in data collection should be present in the database. No major deviations from Flagstaff Airport data should be observed.

8.3 Testing Results

The results from experiment one can be seen in the tabulated form below. The results of the test were considered conclusive despite the clear error from the tachometer readings. This error was associated with the tachometer itself due to the variable distance from the sensor face causing measurement issues. Only one reasonable data set from the tachometer was able to be collected. The testing performed using the handheld anemometer proved to be extremely closely matched to our sensor output and thus the test was passed.

Table 4: Anemometer Calibration Verification Results

Measurement:	Average Reading:	%Difference:
Anemometer	3.2 m/s - 3.704 Hz	N/A
Handheld Anemometer	3.2 m/s	<1%
Tachometer	3.009 Hz	20.69%

Experiment two aimed to verify the accuracy of our barometric pressure sensor by means of data comparison to the Flagstaff Pulliam Airport hourly readings. A graph of our sea-level adjusted pressure readings averaged over each hour compared to simultaneous readings from the airport can be seen below. These readings were taken over a continuous twelve-hour period with an average difference of 1.35% from Airport readings. This low percentage is well within our 3-5% accuracy requirement and thus the test was passed.

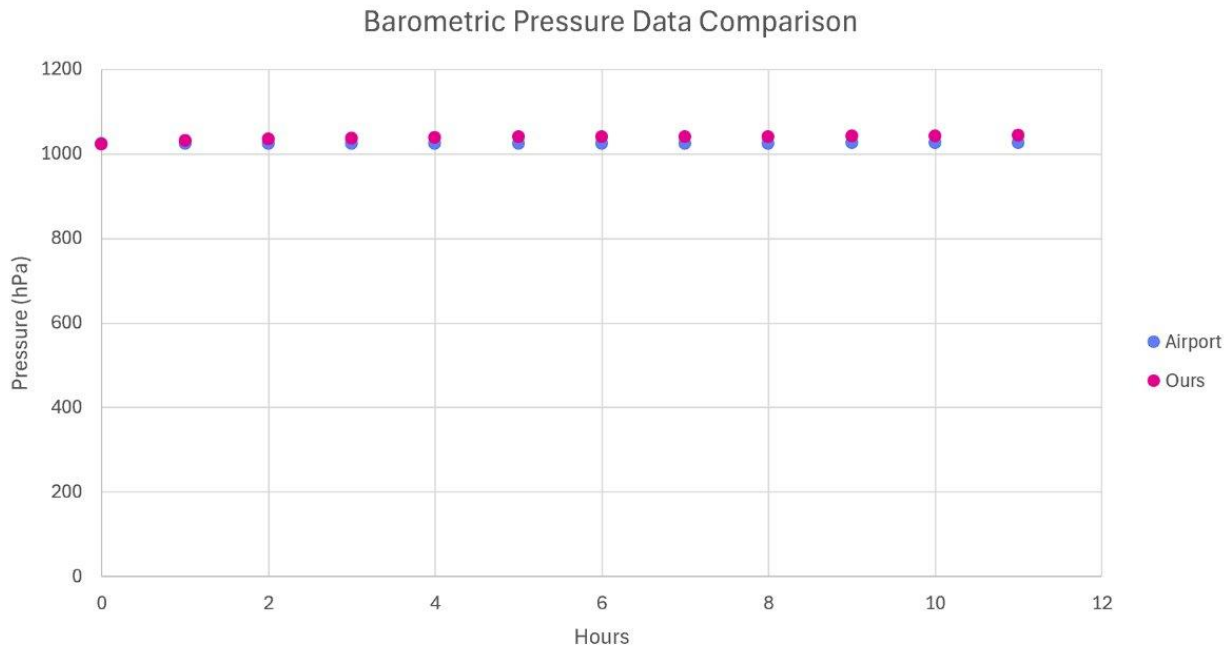


Figure 31: Barometric Pressure Data Comparison

The third experiment conducted was a check of solar irradiance readings gathered from our pyranometer. A major roadblock was hit during this test as the pyranometer was discovered to not be interfacing with the Pi correctly. This is believed to be due to the amplifier not working correctly and was further investigated through extensive troubleshooting. This test failed due to the inability of our system to collect solar irradiance data and therefore there was no comparison to be made. This is still being investigated, and progress has been made in fixing the amplifier. While the data is currently innacurate, the team hopes to resolve this issue soon.

The fourth experiment conducted was the testing of our temperature sensor against Pulliam airport weather data as well as a platinum RTD sensor. The temperature outputs of our sensor were averaged hourly and compared to simultaneous hourly temperatures collected from the airport database. A plot of these averages over a 24-hour period can be seen below. The average difference between the airport data and our sensor was 3.42% over the entire data collection period and a 4.7% difference from the RTD. This is within our 5% accuracy requirement and thus the test was passed.

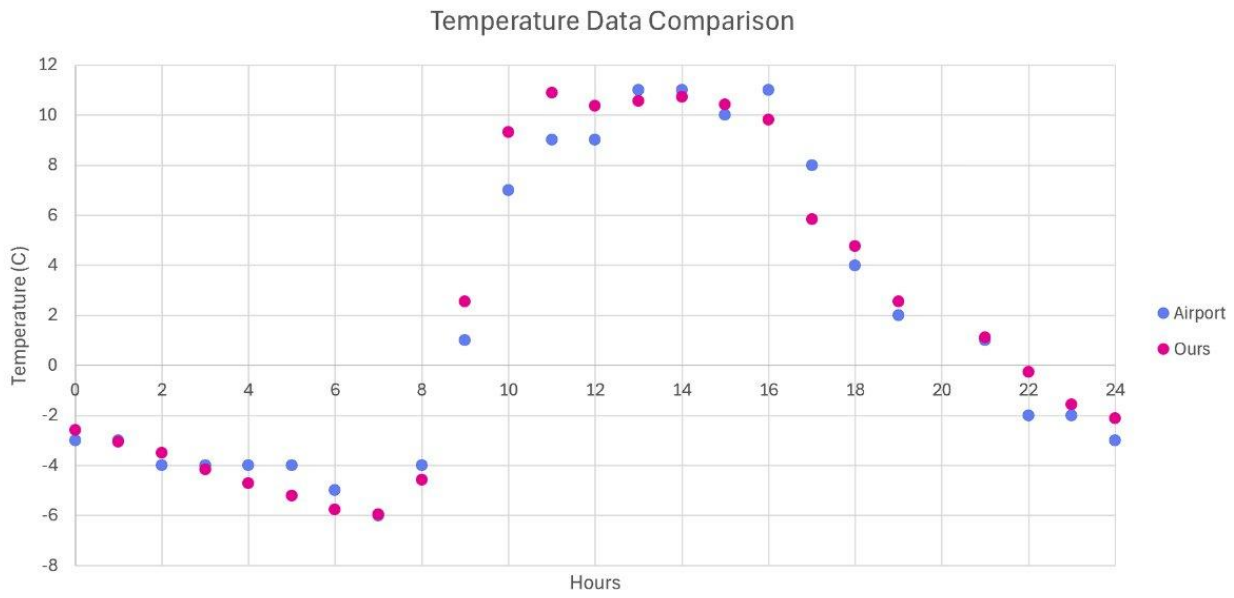


Figure 32: Temperature Sensor vs Airport Data Comparison

The fifth experiment conducted was a calibration test of our wind vane. This test was very simple and involved aligning the wind vane with each cardinal direction and verifying the output. This was done using a piece of paper and eyeballing the alignment of the vane. The test showed the wind vane to accurately respond to each alignment and to reset back to zero degrees after one full rotation, meaning the test was passed.

The sixth experiment conducted involved a simple stress test of the anemometer and wind vane booms. The focus was not on the strength of the booms, but rather the strength of the mounting strategy used to attach them to the tower. Each boom is connected using three hose clamps. A bucket with an extimeted weight of 15-18 lbs was attached to the end of each boom and swung to simulate intense wind gusting. No vertical oscillations and no loosening of the clamps were observed. The booms remained level after multiple trials and thus the test was considered passed.

The seventh and final experiment involved the website and database. The test illustrated an issue with the data graphs displayed on the website where certain data points could not be viewed. To combat this issue, an “export CSV” option was added in which all of the data from the graph would be exported into a spreadsheet with timestamps. An additional issue was found in which any interruption to the Raspberry Pi’s internet connection would cause a full stop to our system. A function was added to the code which checks for an internet connection and automatically reconnects if it is not found.

9 Looking Forward

9.1 Future Work

The team is currently working on finalizing the testing for the pyranometer, as the tests have so far been failures. The testing should be completed within the next few days, however if the team is unable to complete this, the expected solution has been discussed with the client. Using more sensitive and expensive tools which the team did not have access to, the amplifier may be calibrated by sending controlled microvolt signals through the AD623 until they reach the desired output. This would be a

14000-microvolt signal outputting 3.3 volts on the Raspberry Pi. With this, another pyranometer, and another amplifier, the calibration could be fully confirmed. Otherwise, future work is in the hands of future classes led by our client.

The point of this weather station was to provide weather data to Carson Pete for use within his various engineering courses. Students will be able to access our website to download different batches of real-time weather data to complete calculations for various assignments. Our weather station can also be replicated with the backup Raspberry Pi and adjusted to allow for more or different sensors as they need replacement over time. There are still two empty slots for more sensors on the MCP3008, so our client could add more sensors to the existing system if he so pleases as well.

10 Conclusions

This document is the final report on the design and development of the Renewable Energy Lab Weather Station for Northern Arizona University. The weather station has been requested to provide weather data measurements of air temperature, humidity, barometric pressure, solar irradiance, and wind speed. This weather station is expected to collect accurate scientific data and have its data accessible from anywhere with an internet connection.

Professor Carson Pete has specifically requested that the data be transmitted and visible via an internet connection. He also requested that the station run off a renewable energy supply, which is a simple request as this station is located in the Renewable Energy Lab. It is also expected to be durable enough to withstand all outdoor weather conditions. Each sensor is expected to be accurate and low maintenance so that the station may be left alone while still collecting accurate data. The website the data is uploaded to is user friendly and has the capability to store the data for at least a year. We have also managed to stay within our budget, as the project was relatively low cost.

In order to adhere to these requirements, we went through a design selection process. While our initial selection was good, a few adjustments have been made over the course of manufacturing and testing the weather station. Our final design has two main aspects. The structural aspect of the design involves the provided metal truss that is in the RE Lab as well as one of the towers stationed there. On top of the six-foot truss, a square-shaped crossbar was secured which has two mounted sensors on it: the barometric pressure sensor and the pyranometer. The thermometer is mounted on the same tower, but on a lower shelf to adhere to the measurement standards. To meet the standards for the anemometer, it was mounted with a boom arm 30ft high off one of the existing towers. The next aspect of this design decision is the coding. The final pseudocode outlines our sensors reading an average of data over two minutes to then be sent to the website. This provides real-time information that is accessible from anywhere with an internet connection. An hourly average is uploaded to the database to be stored indefinitely. This should prevent having too much data stored on the Raspberry Pi.

Our first prototype failed in answering the question that we had sought to answer; however, it was this that provided us with the insight that we needed extra parts. With this revelation, and an additional sensor, we were able to create a fully functional prototype one week later. This prototype took in three types of weather data: humidity, temperature, and pressure. These three types of data were run through code which output them onto graphs so that they were updated in real-time while being legible.

Through testing, our team was able to confirm 6 out of 7 sensors to be working as expected and outputting proper data. While this is not perfect, efforts are underway to finalize the last sensor before the

semester is over. Despite this, our client has approved of our progress and deemed the project a general success.

The final Bill of Materials and the Gantt chart both display our finalized project which was completed this semester. With our client's approval, the team has successfully created a weather station for the Renewable Energy Lab at Northern Arizona University.

11 REFERENCES

[1] Setra Systems, Inc, “Barometric Pressure Sensors | Setra Systems,” Setra.com, 2025. https://www.setra.com/product/pressure/barometric (accessed Jun. 18, 2025).

[2] N. US Department of Commerce, “Standards and Policy,” www.weather.gov. https://www.weather.gov/coop/standards

[3] World Meteorological Organization, Guide to meteorological instruments and methods of observation., Volume 1. Geneva, Switzerland: World Meteorological Organization, 2008.

[4] ISO-CAL, “What is a Pyranometer? 10 Important Points to Consider.,” isocalnorthamerica.com, Jan. 16, 2023. https://isocalnorthamerica.com/what-is-a-pyranometer/

[5] C. Gittins, “Considering the energy consumption of a Raspberry Pi,” IOT Insider, Sep. 23, 2024. https://www.iotinsider.com/news/considering-the-energy-consumption-of-a-raspberry-pi/

[6] NiuBol, “How much power does a weather station use？,” Niubol.com, 2024. https://www.niubol.com/Product-knowledge/How-much-power-does-a-weather-station-use.html

[7] US, “National Weather Service,” Weather.gov, 2025. https://forecast.weather.gov/MapClick.php?lat=35.19814000000002&lon=-111.65112499999998 (accessed Jun. 18, 2025).

[8] J. Portilla, “Python Bootcamps: Learn Python Programming and Code Training,” Udemy, 2019. https://nau.udemy.com/course/complete-python-bootcamp (accessed Jun. 22, 2025).

[9] Raspberry Pi, “Build Your Own Weather Station,” Raspberrypi.org, 2017. https://projects.raspberrypi.org/en/projects/build-your-own-weather-station/0 (accessed Jun. 27, 2025).

[10] Raspberry Pi, “Raspberry Pi Documentation - Getting Started,” www.raspberrypi.com. https://www.raspberrypi.com/documentation/computers/getting-started.html (accessed Jun. 27, 2025).

[11] Raspberry Pi, “Raspberry Pi OS,” Raspberry Pi, 2025. https://www.raspberrypi.com/software/ (accessed Jun. 27, 2025).

[12] “Tempest Weather Station,” Tempest, 2024. https://shop.tempest.earth/products/tempest msclkid=1236cca887f915b3bbd3a4f8024b2f08&utm_source=bing&utm_medium=cpc&utm_campaign=Bing-Search-B2C-NB-Broad US&utm_term=home%20weather%20station&utm_content=Weather%20Station (accessed Jun. 18, 2025).

[13] A. Overton, “A Guide to the Siting, Exposure and Calibration of Automatic Weather Stations for Synoptic and Climatological Observations,” 2009. Accessed: May 01, 2024. [Online]. Available: https://www.rmets.org/sites/default/files/2019-02/aws-guide.pdf

[14] Wisconsin DNR, “Calibration & barometric pressure | | Wisconsin DNR,” dnr.wisconsin.gov. https://dnr.wisconsin.gov/topic/labCert/BODCalibration2.html

[15] R. Coquilla, J. Obermeier, and B. White, “American Wind Energy Association,” 2007. Accessed: May 16, 2023. [Online]. Available: https://research.engineering.ucdavis.edu/wind/wp-content/uploads/sites/17/2014/03/AWEA-2007-Final-Paper.pdf

[16] “Login - CAS – Central Authentication Service,” Udemy.com, 2025. Available: https://nau.udemy.com/course/statistics-intro/learn/lecture/35671972#overview. [Accessed: Jun. 25, 2025]

[17] EngineerItProgram, “Experimental Uncertainty,” YouTube, Mar. 18, 2013. Available: https://www.youtube.com/watch?v=xJBta_HWTRc. [Accessed: Jun. 26, 2025]

[18] RDS, Ed., “Certificate for Calibration of Cup Anemometer,” SOH Wind Engineering LLC, 141 Leroy Rd - Williston, VT 05495 USA, Jun. 2021. Accessed: Jul. 26, 2025. [Online]. Available: https://www.nrgsystems.com/support/product-support/technical-product-sheets/technical-product-sheet-40
[19] “Certificate of Calibration,” NRG Systems, 110 Riggs Rd - Hinesburg, VT 05461 USA, Jun. 2021. Accessed: Jul. 26, 2025. [Online]. Available: https://www.nrgsystems.com/support/product-support/instruction-sheets/nrg-bp60-barometric-pressure-sensor-instructions
[20] J. Calandra, “Certificate of Calibration,” ESSCO Calibration Laboratory, 27 Industrial Ave Unit #9 - Chelmsford, MA 01824 - 3618 USA, Jul. 2021. Accessed: Jul. 26, 2025. [Online]. Available: https://www.nrgsystems.com/products/met-sensors/detail/nrg-t60-temperature-sensor
[21] J. Brooks, “Certificate of Calibration for LI-COR Sensor,” LI-COR, 4647 Superior Street - Lincoln, NE 68504 USA, Jun. 2020. Accessed: Jul. 28, 2025. [Online]. Available: https://www.licor.com/products/light/pyranometer
[22] E. Instruments, “Pyranometer EKO MS-60/MS-60S - Instruction Manual Ver. 8,” EKO INSTRUMENTS CO., LTD, 111 North Market Street, Suite 300 San Jose, CA 95113 USA, Oct. 2023. Accessed: Aug. 04, 2025. [Online]. Available: https://eko-instruments.com/us/product/ms-60-pyranometer/
[23] National Instruments, “NI DAQ Sensor Calibration Guide,” National Instruments, 2021. Accessed: May 10, 2024. [Online]. Available: https://www.ni.com/en-us/innovations/sensor-calibration.html
[24] Adafruit Industries, “How to Connect Weather Sensors to Raspberry Pi,” Adafruit Learning System, 2022. Accessed: Apr. 25, 2024. [Online]. Available: https://learn.adafruit.com/pi-weather-station/overview
[25] M. A. Islam and R. Haque, “IoT-Based Real-Time Weather Monitoring System Using Raspberry Pi,” in Proc. Int. Conf. on IoT and Applications, 2020, pp. 44–49.
[26] S. Pandey et al., “Analysis of Weather Monitoring Systems Using Cloud Integration,” Journal of Sensor Networks and Data Communication, vol. 10, no. 2, pp. 100–105, 2021.
[27] Ambient Weather, “Installation Guide for Ambient Weather WS-5000,” Ambient Weather, 2023. Accessed: May 15, 2024. [Online]. Available: https://ambientweather.com/ws5000manual
[28] Texas Instruments, “Thermal Design Considerations for Enclosures,” Application Report SPRA953, 2009. Accessed: July 14, 2025. [Online]. Available: https://www.ti.com/lit/an/spra953/spra953.pdf
[29] IPC, “Standard for Determining Current-Carrying Capacity in Printed Board Design (IPC-2152),” IPC, 2009. Accessed: July 14, 2025. [Online]. Available: https://www.ipc.org/TOC/IPCT-2152.pdf
[30] “Printed humidity sensors,” Encyclopedia.pub. https://encyclopedia.pub/entry/7493.
[31] “Review of Printed Humidity Sensors,” MDPI Sensors. https://www.mdpi.com/2079-4991/13/6/1110
[32] “Capacitive vs Resistive Humidity Sensors,” Encyclopedia.pub. – https://encyclopedia.pub/entry/7493
[33] BOMS Review – Comparison of sensing layers – https://www.ias.ac.in/article/fulltext/boms/045/0238
[34] “Humidity and Dew Point,” National Weather Service. https://www.weather.gov/media/epz/wxcalc/dewpoint_rh.htm
[35] Ma, H. et al., “Graphene-Based Humidity Sensors,” arXiv. https://arxiv.org/abs/2410.02255
[36] NOAA Sensor Siting Standards – https://www.weather.gov/coop/standards
[37] WMO Guide to Meteorological Instruments – https://library.wmo.int/index.php?lvl=notice_display&id=12407
[38] Shilleh, “Beginner Tutorial: How to Connect Raspberry Pi and BME280 for Pressure, Temperature, and Humidity,” YouTube, Nov. 20, 2023. https://www.youtube.com/watch?v=T7L7WMHbhY0 (accessed Jul. 30, 2025).

[39] DwyerOmega, “Modular Weather Monitoring and Data Storage Stations,” Dwyeromega.com, 2015. https://www.dwyeromega.com/en-us/modular-weather-monitoring-and-data-storage-stations/p/WMS-25-Series?srsltid=AfmBOopea9LGqjBDOoZJngG3fUaw1xYyItpGe5olf4VLMBsv2Ow8Cz9z# (accessed Jul. 15, 2025).
[40] P. Keheley, “How Many Pages In A Gigabyte? A Litigator’s Guide,” www.digitalwarroom.com, 2020. https://www.digitalwarroom.com/blog/how-many-pages-in-a-gigabyte (accessed Jul. 15, 2025).
[41] Bosch Sensortec. (n.d.). BME280 Datasheet. Retrieved from https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/
[42] NRG Systems. (n.d.). #40C Anemometer Product Manual. Retrieved from https://www.nrgsystems.com/
[43] NRG Systems. (n.d.). BP60C Barometer, T60C Thermometer, LI-COR LI200R & EKO MS-60 Pyranometers Specifications. Retrieved from manufacturer data sheets.
[44] Adafruit. (n.d.). BME280 Python Library. Retrieved from https://github.com/adafruit/Adafruit_Python_BME280
[45] Python Software Foundation. (n.d.). Python Requests Library Documentation. Retrieved from https://docs.python-requests.org/
[46] SQLite. (n.d.). SQLite Documentation. Retrieved from https://sqlite.org/

12 APPENDICES

12.1 Appendix A: Pseudocode

Design 1(Chenxi)-
Main Ideas:
Independent upload for each sensor.
Local backup for 24 hours.
Adaptive sampling rate based on environmental stability.
Initialization：
Load all sensor drivers.
Connect to SQLite database and server.
Define thresholds for environmental stability.
Every 10 minutes：

```
for sensor in sensors:
    data = read(sensor)
    timestamp = get_time()
    if sensor == "humidity":
        if abs(data - last_value) > 10 or data > 100:
            flag = "outlier"
        else:
            flag = "valid"
        save_local(sensor, data, timestamp, flag)
    else:
        save_local(sensor, data, timestamp)
        upload(sensor, data, timestamp)
```

Data is collected every 10 minutes.
Humidity data adds abnormal value judgment.
All data is first saved to the local database and then uploaded to the remote server.
Daily Maintenance：

```
delete_local_data(older_than=24_hours)
generate_daily_report()
```

Clean up local caches older than 24 hours every day and generate a sensor operation report for the day.
Adaptive Sampling Rate Logic：

```
if variance(sensor_data) < threshold:
    set_interval(longer)  # Environment is stable → reduce sampling frequency
else:
        set_interval(shorter)  # Drastic changes in the environment → Increase sampling frequency
```

Determine the sampling interval by judging the degree of data change.
Environmental stability → energy saving
Environmental fluctuations → improved resolution

Design 2(Shutong)-
Script: weather_visualization.py

Load data from CSV file
file_path = "weather_data.csv" data = read_csv(file_path)
Clean the data
Convert 'timestamp' to datetime format Remove rows with missing values
Filter temperature: Keep values between -50 and 60°C
Filter pressure: Keep values between 800 and 1100 hPa
Filter last 24 hours of data
now = current_datetime() start_time = now - 24 hours recent_data = data where timestamp >= start_time
Plot temperature chart
Create figure (10x4) Plot timestamp vs. temperature Label: "Temperature (°C)" Title: "Temperature Over Last 24 Hours" Rotate x-ticks by 45 degrees Add grid and legend Save as "temperature_last24h.png"
Plot pressure chart
Create figure (10x4) Plot timestamp vs. pressure Label: "Pressure (hPa)" Title: "Pressure Over Last 24 Hours" Rotate x-ticks by 45 degrees Add grid and legend Save as "pressure_last24h.png"
Output confirmation message
Print: "Charts saved: temperature_last24h.png and pressure_last24h.png"

Design 3 (Rowan)-

```
# Buffers
rolling_buffer = []       # Up to 20 raw readings (~10 min at 30s intervals)
ten_minute_buffer = []    # 1 average per minute, keep 10
hourly_buffer = []        # 1 average per 10 minutes, keep 6
LOOP every 30 seconds:
   data = read_and_calibrate_all_sensors()
   # Maintain rolling buffer
   IF length of rolling_buffer == 20:
      Remove oldest
   Append data to rolling_buffer
   # Every 1 minute: upload rolling average for real-time display
   IF time_since_last_minute == 60 sec:
      rolling_avg = average_buffer(rolling_buffer)
      upload_to_website(rolling_avg)
      Append rolling_avg to ten_minute_buffer
      # Maintain 10-minute buffer
      IF length of ten_minute_buffer == 10:
         ten_min_avg = average_buffer(ten_minute_buffer)
         Append ten_min_avg to hourly_buffer
         Clear ten_minute_buffer
         # Every 60 minutes (6x10min)
         IF length of hourly_buffer == 6:
 hourly_avg = average_buffer(hourly_buffer)
            store_hourly_avg_to_database(hourly_avg)
            Clear hourly_buffer
```

DATUM (Ian):

IMPORTANT NOTE CODE IS HIGHLY DEPENDANT ON WHAT SENSOR IS BEING USED
Anemometer: will produce signals as it rotates. These signals will likely occur twice per rotation. With this, wind speed will be calculated as:
Circumference = 2 * pi * radius
Rotations = signals/2
Speed = (rotations * circumference) / time
#this will then convert to kmph or mph through simple conversion math.
For rainfall: the signal will be produced when one bucket fills enough to tip over. Knowing the tipping point of the bucket is essential for calculating the overall rainfall.
bucket = .227mm #this is example size not real
count = 0
def bucket_tipped():
global count
Count = count+1
Print (count*bucket)
Rainfall= bucket_tipped/time
#time should measure from start of rain till end
#Reset function here
Temperature:
The thermometer can output many things depending on what it is. Assuming it outputs current it will be in mA
Reading = data input #in mA
Temp = # reading altered via reference table or specific formula.
#if using reference table, indexing and slicing very important. Additionally, the table will be uploaded to the Raspberry Pi.
Humidity
Similar to temperature, it will produce voltage or amps.
The code will take those outputs and convert them either using a reference table or a specific formula, likely provided in the instructions for the device.
Humidity = (input) + #modification based on input. Either formula or reference table
Barometric Pressure
Again, similar to temperature and entirely dependent on what sensor we have.
Pressure = (input) #modified with formula or table
Solar irradiance
#Will likely need a voltage reader, then convert that voltage to W/m^2 through calibration/ specific formulaIrradiance= (V #modified by formula) in (W/m^2)
#All of this data will be collected internally into separate tables specific to each sensor. Then, every 5 min (or sooner for certain devices) it will upload this to the website where it will then be sorted properly and displayed in a user-friendly manner.

12.2 Appendix B: Sensor Code

Anemometer

```
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
import time
from time import sleep
import sqlite3, pathlib, datetime as dt

#Paths/config
DB_PATH  = pathlib.Path("/home/pi/weather_db/weather.db")

running = True

#MCP3008
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
cs  = digitalio.DigitalInOut(board.D5)
mcp = MCP.MCP3008(spi, cs)

chan = AnalogIn(mcp, MCP.P2)

zero_threshold = 0.1          # V
MEASUREMENT_DURATION_S = 300.0  # 5 min window like the main anemometer
IDLE_GAP_S = 0.0              # optional pause between runs

#calibration
def freq_to_ms(freq_hz: float) -> float:
   return 0.75776 * freq_hz + 0.39322

#DB
SCHEMA_SQL = """
PRAGMA journal_mode=WAL;
CREATE TABLE IF NOT EXISTS readings (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 ts_utc TEXT NOT NULL,
```

```
 temperature_c REAL,
 humidity_pct REAL,
 pressure_hpa REAL,
 wind_speed_ms REAL,
 wind_direction_deg REAL,
 rainfall_mm REAL,
 solar_wm2 REAL
);
CREATE INDEX IF NOT EXISTS idx_readings_ts ON readings(ts_utc);
"""

def ensure_db():
  DB_PATH.parent.mkdir(parents=True, exist_ok=True)
  with sqlite3.connect(DB_PATH) as con:
    con.executescript(SCHEMA_SQL)
    try:
      con.execute("ALTER TABLE readings ADD COLUMN low_wind_speed_ms REAL;")
    except sqlite3.OperationalError:
      pass  # column already exists

def insert_low_wind(ts_utc: str, low_wind_ms: float):
  with sqlite3.connect(DB_PATH, timeout=3.0) as con:
    con.execute("PRAGMA busy_timeout=3000;")
    con.execute("""INSERT INTO readings(
      ts_utc, temperature_c, humidity_pct, pressure_hpa,
      wind_speed_ms, wind_direction_deg, rainfall_mm, solar_wm2, low_wind_speed_ms
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)""",
    (ts_utc, None, None, None, None, None, None, None, float(low_wind_ms)))
    con.commit()


def calculate_frequency_zero_crossing(duration=MEASUREMENT_DURATION_S):
  start_time = time.time()
  zero_crossings = 0
  prev_value = chan.voltage

  while time.time() - start_time < duration:
    current_value = chan.voltage
```

```
      sleep(0.1)
      if ((prev_value > zero_threshold and current_value <= zero_threshold) or
        (prev_value <= zero_threshold and current_value > zero_threshold)):
        zero_crossings += 1
      prev_value = current_value

    if zero_crossings > 0:
      return (zero_crossings / 2.0) / duration
    return 0.0

#loop
ensure_db()
while running:
    print(f"[low-anemometer] Measuring frequency for {int(MEASUREMENT_DURATION_S)}s
on CH2 ...")
    try:
      freq = calculate_frequency_zero_crossing()
      print(f"[low-anemometer] freq={freq:.3f} Hz")

      if freq > 0:
        v_ms  = max(0.0, freq_to_ms(freq))
        v_mph = v_ms * 2.237
        print(f"[low-anemometer] {v_ms:.2f} m/s  ({v_mph:.2f} mph)")
      else:
        v_ms = 0.0
        print("[low-anemometer] 0.00 m/s (0.00 mph)")

      ts = dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"
      insert_low_wind(ts, v_ms)

      if IDLE_GAP_S > 0:
        time.sleep(IDLE_GAP_S)

    except KeyboardInterrupt:
      running = False
      print("\n[low-anemometer] stopped.")
    except Exception as e:
      print("[low-anemometer] error:", e)
```

```
    time.sleep(5)
```

Wind Vane

```
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
from time import sleep
import sqlite3, pathlib, datetime as dt

#Paths
DB_PATH  = pathlib.Path("/home/pi/weather_db/weather.db")

#spi bus
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
#create chip select
cs  = digitalio.DigitalInOut(board.D5)
#mcp object
mcp = MCP.MCP3008(spi, cs)
#analog input on ch1
chan = AnalogIn(mcp, MCP.P1)

max_voltage = 3.3
running = True

#DB
SCHEMA_SQL = """
PRAGMA journal_mode=WAL;
CREATE TABLE IF NOT EXISTS readings (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 ts_utc TEXT NOT NULL,
 temperature_c REAL,
 humidity_pct REAL,
 pressure_hpa REAL,
 wind_speed_ms REAL,
 wind_direction_deg REAL,
 rainfall_mm REAL,
```

```
 solar_wm2 REAL
);
CREATE INDEX IF NOT EXISTS idx_readings_ts ON readings(ts_utc);
"""

def ensure_db():
  DB_PATH.parent.mkdir(parents=True, exist_ok=True)
  with sqlite3.connect(DB_PATH) as con:
    con.executescript(SCHEMA_SQL)

def insert_direction(ts_utc: str, wind_direction_deg: float):
  with sqlite3.connect(DB_PATH, timeout=3.0) as con:
    con.execute("PRAGMA busy_timeout=3000;")
    con.execute("""INSERT INTO readings(
      ts_utc, temperature_c, humidity_pct, pressure_hpa,
      wind_speed_ms, wind_direction_deg, rainfall_mm, solar_wm2
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?)""",
    (ts_utc, None, None, None, None, wind_direction_deg, None, None))
    con.commit()

#loop
ensure_db()

while running:
  try:
    voltage = chan.voltage
    direction = (voltage / max_voltage) * 360.0
    print(f"Voltage: {voltage:.4f}V  Wind Direction: {direction:.2f}°")

    ts = dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"
    insert_direction(ts, round(direction, 2))

    sleep(10)  # check every second (adjust as needed)
  except KeyboardInterrupt:
    running = False
    print("\n[windvane] stopped.")
  except Exception as e:
    print("[windvane] error:", e)
```

```
    sleep(1)
```

Barometer

```
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
import time
from time import sleep
import sqlite3, pathlib, datetime as dt

# Paths/config
DB_PATH  = pathlib.Path("/home/pi/weather_db/weather.db")

running = True

# SPI bus
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
# create chip select
cs  = digitalio.DigitalInOut(board.D5)
# mcp object
mcp = MCP.MCP3008(spi, cs)

# analog input on ch6
chan = AnalogIn(mcp, MCP.P6)

# BP60 calibration (station pressure at sensor height)
SLOPE_HPA_PER_V = 243.8566
OFFSET_HPA     = 494.6673
MAX_VOLTAGE    = 5.0

# Site altitude for sea-level correction (meters)
ALTITUDE_M = 2106.0

# Sampling controls
SAMPLE_INTERVAL_S = 10.0       # how often to write a pressure point (seconds)
AVERAGE_SAMPLES   = 10         # number of ADC reads to average each cycle
```

```
AVERAGE_DELAY_S   = 0.02      # delay between ADC reads when averaging (s)

# DB schema (kept as-is)
SCHEMA_SQL = """
PRAGMA journal_mode=WAL;
CREATE TABLE IF NOT EXISTS readings (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 ts_utc TEXT NOT NULL,
 temperature_c REAL,
 humidity_pct REAL,
 pressure_hpa REAL,
 wind_speed_ms REAL,
 wind_direction_deg REAL,
 rainfall_mm REAL,
 solar_wm2 REAL
);
CREATE INDEX IF NOT EXISTS idx_readings_ts ON readings(ts_utc);
"""


def ensure_db():
   DB_PATH.parent.mkdir(parents=True, exist_ok=True)
   with sqlite3.connect(DB_PATH) as con:
      con.executescript(SCHEMA_SQL)


def insert_pressure(ts_utc: str, pressure_hpa: float):
   """Insert sea-level pressure into the readings table."""
   with sqlite3.connect(DB_PATH, timeout=3.0) as con:
      con.execute("PRAGMA busy_timeout=3000;")
      con.execute(
         """INSERT INTO readings(
            ts_utc, temperature_c, humidity_pct, pressure_hpa,
            wind_speed_ms, wind_direction_deg, rainfall_mm, solar_wm2
          )
          VALUES (?, ?, ?, ?, ?, ?, ?, ?)""",
        (ts_utc, None, None, pressure_hpa, None, None, None, None),
      )
```

```
    con.commit()


def get_latest_temperature_c() -> float | None:
  """
  Fetch the most recent non-NULL temperature_c written by T60C.
  Returns None if no temperature is available.
  """
  try:
    with sqlite3.connect(DB_PATH, timeout=3.0) as con:
      con.execute("PRAGMA busy_timeout=3000;")
      cur = con.execute(
        """
        SELECT temperature_c
        FROM readings
        WHERE temperature_c IS NOT NULL
        ORDER BY ts_utc DESC
        LIMIT 1
        """
      )
      row = cur.fetchone()
    if row and row[0] is not None:
      return float(row[0])
  except Exception as e:
    print("[barometer] warning: cannot read latest temperature:", e)
  return None


def average_voltage(n=AVERAGE_SAMPLES, delay=AVERAGE_DELAY_S):
  total = 0.0
  for _ in range(max(1, n)):
    total += chan.voltage
    if delay > 0:
      sleep(delay)
  return total / max(1, n)


def voltage_to_station_hpa(vout: float) -> float:
```

```
    """
    Convert sensor output voltage to station pressure (hPa)
    at the sensor height, using the BP60 linear calibration.
    """
    v = max(0.0, min(MAX_VOLTAGE, vout))
    return SLOPE_HPA_PER_V * v + OFFSET_HPA


def station_to_sea_level_hpa(p_station_hpa: float,
                             temp_c: float,
                             altitude_m: float = ALTITUDE_M) -> float:
    """
    Convert station pressure (hPa) at altitude_m (m) and ambient
    temperature temp_c (°C) to sea-level pressure (hPa) using a
    standard-atmosphere barometric formula.
    """
    # Standard atmosphere constants
    L = 0.0065   # K/m
    EXP = 5.255  # g*M/(R*L) approx

    # Protect against bad inputs
    T_kelvin = max(180.0, temp_c + 273.15)   # avoid crazy low temps
    h = max(0.0, float(altitude_m))

    factor = 1.0 - (L * h) / T_kelvin
    if factor <= 0.0:
        # If the factor is non-physical, just return station pressure
        return p_station_hpa

    return p_station_hpa / (factor ** EXP)


# loop
ensure_db()
print(
    f"[barometer] channel={MCP.P6}, interval={SAMPLE_INTERVAL_S}s, "
    f"avg={AVERAGE_SAMPLES}, altitude={ALTITUDE_M} m"
)
```

```
while running:
  try:
    # 1) ADC averaging -> voltage
    v = average_voltage()

    # 2) Voltage -> station pressure (sensor height)
    p_station_hpa = voltage_to_station_hpa(v)

    # 3) Get latest temperature from DB (T60C)
    temp_c = get_latest_temperature_c()
    if temp_c is None:
      # Fallback temperature if T60C has not written yet
      temp_c = 10.0

    # 4) Station pressure + temperature -> sea-level pressure
    p_sea_hpa = station_to_sea_level_hpa(p_station_hpa, temp_c, ALTITUDE_M)

    # 5) Timestamp & logging
    ts = dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"

    print(
      f"Vout={v:.5f} V  "
      f"Station={p_station_hpa:.3f} hPa  "
      f"Sea-level={p_sea_hpa:.3f} hPa  "
      f"(T60C={temp_c:.1f} °C)"
    )

    # 6) Store sea-level pressure in DB
    insert_pressure(ts, round(p_sea_hpa, 3))

    sleep(SAMPLE_INTERVAL_S)

  except KeyboardInterrupt:
    running = False
    print("\n[barometer] stopped.")
  except Exception as e:
    print("[barometer] error:", e)
```

```
    sleep(1)
```

Humidity

```
import smbus2
import bme280
import time
import sqlite3, pathlib, datetime as dt
import sys

#Paths/config
DB_PATH  = pathlib.Path("/home/pi/weather_db/weather.db")

I2C_BUS_ID    = 1
BME280_ADDRESS = 0x76
SAMPLE_INTERVAL_S = float(sys.argv[1]) if len(sys.argv) > 1 else 5.0

#DB
SCHEMA_SQL = """
PRAGMA journal_mode=WAL;
CREATE TABLE IF NOT EXISTS readings (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 ts_utc TEXT NOT NULL,
 temperature_c REAL,
 humidity_pct REAL,
 pressure_hpa REAL,
 wind_speed_ms REAL,
 wind_direction_deg REAL,
 rainfall_mm REAL,
 solar_wm2 REAL
);
CREATE INDEX IF NOT EXISTS idx_readings_ts ON readings(ts_utc);
"""

def ensure_db():
   DB_PATH.parent.mkdir(parents=True, exist_ok=True)
   with sqlite3.connect(DB_PATH) as con:
      con.executescript(SCHEMA_SQL)
```

```
def insert_humidity(ts_utc: str, humidity_pct: float):
  """Insert only humidity; keep other columns NULL."""
  with sqlite3.connect(DB_PATH, timeout=3.0) as con:
    con.execute("PRAGMA busy_timeout=3000;")
    con.execute("""INSERT INTO readings(
      ts_utc, temperature_c, humidity_pct, pressure_hpa,
      wind_speed_ms, wind_direction_deg, rainfall_mm, solar_wm2
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?)""",
    (ts_utc, None, humidity_pct, None, None, None, None, None))
    con.commit()

def utc_ts():
  return dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"

def main():
  ensure_db()

  bus = smbus2.SMBus(I2C_BUS_ID)
  calibration_params = bme280.load_calibration_params(bus, BME280_ADDRESS)

  print(f"[bme280] addr=0x{BME280_ADDRESS:02X} interval={SAMPLE_INTERVAL_S}s")
  running = True
  while running:
    try:
      data = bme280.sample(bus, BME280_ADDRESS, calibration_params)
      humidity = float(data.humidity)  # %
      ts = utc_ts()

      print(f"humidity={humidity:.2f}%")
      insert_humidity(ts, round(humidity, 2))

      time.sleep(SAMPLE_INTERVAL_S)
    except KeyboardInterrupt:
      running = False
      print("\n[bme280] stopped.")
    except Exception as e:
      print("[bme280] error:", e)
      time.sleep(5)
```

```
if __name__ == "__main__":
    main()
```

Thermometer

```
    import busio
    import digitalio
    import board
    import adafruit_mcp3xxx.mcp3008 as MCP
    from adafruit_mcp3xxx.analog_in import AnalogIn
    import time
    from time import sleep
    import sqlite3, pathlib, datetime as dt
    import sys

    #Paths/config
    DB_PATH  = pathlib.Path("/home/pi/weather_db/weather.db")

    SAMPLE_INTERVAL_S = float(sys.argv[1]) if len(sys.argv) > 1 else 5.0

    running = True

    #MCP3008
    spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
    cs  = digitalio.DigitalInOut(board.D5)
    mcp = MCP.MCP3008(spi, cs)

    #CH0 for T60C
    chan = AnalogIn(mcp, MCP.P0)

    #T60C calibration
    SCALE_C_PER_V = 44.6894
    OFFSET_C     = -40.7170
    MAX_VOLTAGE   = 5.0

    # optional averaging to reduce noise
    AVG_SAMPLES   = 8
    AVG_DELAY_S   = 0.01
```

```
#DB
SCHEMA_SQL = """
PRAGMA journal_mode=WAL;
CREATE TABLE IF NOT EXISTS readings (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 ts_utc TEXT NOT NULL,
 temperature_c REAL,
 humidity_pct REAL,
 pressure_hpa REAL,
 wind_speed_ms REAL,
 wind_direction_deg REAL,
 rainfall_mm REAL,
 solar_wm2 REAL
);
CREATE INDEX IF NOT EXISTS idx_readings_ts ON readings(ts_utc);
"""

def ensure_db():
   DB_PATH.parent.mkdir(parents=True, exist_ok=True)
   with sqlite3.connect(DB_PATH) as con:
      con.executescript(SCHEMA_SQL)

def insert_temperature(ts_utc: str, temp_c: float):
   with sqlite3.connect(DB_PATH, timeout=3.0) as con:
      con.execute("PRAGMA busy_timeout=3000;")
      con.execute("""INSERT INTO readings(
         ts_utc, temperature_c, humidity_pct, pressure_hpa,
         wind_speed_ms, wind_direction_deg, rainfall_mm, solar_wm2
      ) VALUES (?, ?, ?, ?, ?, ?, ?, ?)""",
      (ts_utc, temp_c, None, None, None, None, None, None))
      con.commit()

def average_voltage(n=AVG_SAMPLES, delay=AVG_DELAY_S):
   s = 0.0
   for _ in range(max(1, n)):
      s += chan.voltage
      if delay > 0:
```

```
        sleep(delay)
    return s / max(1, n)

def voltage_to_c(vout: float) -> float:
    v = max(0.0, min(MAX_VOLTAGE, vout))
    return SCALE_C_PER_V * v + OFFSET_C

def utc_ts():
    return dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"

#Main loop
ensure_db()
print(f"[T60C] CH0 active, interval={SAMPLE_INTERVAL_S}s, avg={AVG_SAMPLES}")

while running:
    try:
        v = average_voltage()
        temp_c = voltage_to_c(v)
        ts = utc_ts()
        print(f"[{ts}] Vout={v:.5f} V  Temp={temp_c:.2f} °C")

        insert_temperature(ts, round(temp_c, 2))

        sleep(SAMPLE_INTERVAL_S)

    except KeyboardInterrupt:
        running = False
        print("\n[T60C] stopped.")
    except Exception as e:
        print("[T60C] error:", e)
        sleep(5)
```

Pyranometer

```
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
```

```
import time
from time import sleep
import sqlite3, json, pathlib, datetime as dt

#Paths/config
DB_PATH  = pathlib.Path("/home/pi/weather_db/weather.db")       # keep DB local (not on SMB)
OUT_DIR  = pathlib.Path("/mnt/schoolsite/ME/2025/Sum25toF25_WeatherStation")  # website folder
OUT_DIR.mkdir(parents=True, exist_ok=True)

running = True  #makes our systems run indef
#spi bus
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
#create chip select
cs  = digitalio.DigitalInOut(board.D5)
#mcp object
mcp = MCP.MCP3008(spi, cs)

#analog input on ch7
chan = AnalogIn(mcp, MCP.P7)

Sensitivity = .00001186

#DB
SCHEMA_SQL = """
PRAGMA journal_mode=WAL;
CREATE TABLE IF NOT EXISTS readings (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 ts_utc TEXT NOT NULL,
 temperature_c REAL,
 humidity_pct REAL,
 pressure_hpa REAL,
 wind_speed_ms REAL,
 wind_direction_deg REAL,
 rainfall_mm REAL,
 solar_wm2 REAL
);
```

```
CREATE INDEX IF NOT EXISTS idx_readings_ts ON readings(ts_utc);
"""

def ensure_db():
  DB_PATH.parent.mkdir(parents=True, exist_ok=True)
  with sqlite3.connect(DB_PATH) as con:
    con.executescript(SCHEMA_SQL)

def insert_solar(ts_utc: str, solar_wm2: float):
  with sqlite3.connect(DB_PATH, timeout=3.0) as con:
    con.execute("PRAGMA busy_timeout=3000;")
    con.execute("""INSERT INTO readings(
      ts_utc, temperature_c, humidity_pct, pressure_hpa,
      wind_speed_ms, wind_direction_deg, rainfall_mm, solar_wm2
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?)""",
    (ts_utc, None, None, None, None, None, None, solar_wm2))
    con.commit()


def utc_ts():
  return dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"

#Main loop
ensure_db()
print("[pyranometer] CH7 active, Sensitivity=%.2f µV/(W/m^2)" % Sensitivity)

while running:
  try:
    sleep(20)
    microV =(chan.voltage /57.09)
    SolarIrr = microV / Sensitivity      # W/m^2
    ts = utc_ts()

    print(f"[{ts}] Solar Irradiance: {SolarIrr:.2f} W/m^2")

    insert_solar(ts, round(SolarIrr, 2))
    export_jsons()
```

```
except KeyboardInterrupt:
    running = False
    print("\n[pyranometer] stopped.")
except Exception as e:
    print("[pyranometer] error:", e)
    sleep(5)
```