A.I.R.E. - Autonomous Intelligent Roof Enclosure

UCF Senior Design 2 - Fall 2019

Group 8
Sarah Riseden - Electrical Engineering
Lauren Miller - Computer Engineering
Christopher Smith - Electrical Engineering
Ronald Acevedo - Electrical Engineering

Sponsor
OpenAire - Retractable Roofs and Skylights

Table of Contents

List of Figures

List of Tables

1. Executive Summary

As the idea of a smart item, also known as the Internet of Things (IoT) technology grows in popularity, it has begun to appear in everything. The most demand being found in the smart home. In order to take the automation of a building even further, the next step would be an automated roof. In order to create a comfortable experience at any location, the roof would be retractable and glass, bringing the outside in. This system would be appropriate over facilities that can only operate comfortably in specific weather conditions but would like to continue functioning even when Mother Nature is not cooperating. In the current market, roof systems like this exist, as created by our sponsor OpenAire. However, this collaboration will allow for further research into the system, by testing any updates and modifications before creating something full scale. This can aide in the company's growth, in order to go above and beyond any other competitors on the market.

In an attempt to solve the issue of weather restrictions, we propose a fully automated retractable glass roof. The automation would come from various sensors to provide the most comfortable outdoor/indoor experience. These sensors would be used primarily for weather predictions to detect the following characteristics of weather: rain, wind speed, air pressure, temperature, humidity, and cloud coverage. The roof will act preemptively according to the outside weather conditions and is able to detect if unwanted weather is on its way, not just when it arrives. The sensors would be programmable to connect to an app, where when certain thresholds are met, the roof will operate automatically. Depending on the environment that the roof is placed over, and the climate it is built in, the roofs sensors could also be adjusted to work well in all conditions.

This system will also come with a locked app available on iPhone, that only allows owners of the roof access. This allows the user to control the system via app and set their own thresholds for operation of the roof and lighting depending on the setting the roof encompasses. With this app, the user can build their own packages of settings and create a schedule on the roofs operation to fit business hours, daylight hours, etc. Along with providing full control of the system, the app will include the statistics read in from the weather sensors, allowing the user to make educated decisions when creating their own roof settings. The app will mainly connect via Wi-Fi as well as having Bluetooth capabilities for backup. In case of emergencies such as a power outage, Wi-Fi outage, or sensor failure, the roof is connected to an emergency power source, and building will have a main switch inside which will override all app settings and sensor data and close the roof.

Overall, this system aims to create a comfortable indoor experience for any market, especially businesses that can only operate in certain weather conditions. Energy and money is saved by the various attributes of the roof including, tinting glass, solar panels, photometric lighting, and HVAC control. According to our sponsor, OpenAire, their customers saw up to 30% in savings for one year of energy costs. Our goal is to increase this percentage via the automation of the roof. Lastly, the system is easily customizable by the user via iOS app, which allows for full customization of the roof's operation dependent on the needs of the building owner.

2. Project Description

The A.I.R.E. autonomous roofing system appears as a retractable glass roof which operates on a wide variety of weather sensors. These sensors allow for the roof to be considered a smart systems, which can react to weather as well as predict incoming weather. Accompanied by an iOS app, this system allows for user to interface communication as well as customization by the user. The app is readable and allow the user to implement their own settings for the roof operation, provide real-time weather statistics as read by the sensors, as well as connect to a local weather API for further weather updates and communication with the implemented sensors.

2.1 Project Background

As the idea of a smart item, a.k.a. the Internet of Things technology grows in popularity, it has begun to appear in everything. The most demand being found in the smart home. In order to take the automation of a building even further, the next step would be an automated roof. In order to create a comfortable experience at any location, the roof would be retractable and glass, bringing the outside in. This system would be appropriate over facilities that can only operate comfortably in specific weather conditions, but would like to continue functioning even when Mother Nature isn't cooperating. These include water facilities, hotels, shopping centers, restaurants, cruise ships, and even residential homes. In the current market, roof systems like this exist, as created by our sponsor OpenAire. However, this collaboration will allow for further research into the system, by testing any updates and modifications before creating something full scale. This can aide in the company's growth, in order to go above and beyond any other competitors on the market.

In an attempt to solve the issue of weather restrictions, we propose a fully automated retractable glass roof. The automation would come from various sensors to provide the most comfortable outdoor/indoor experience. These sensors would be used primarily for weather predictions to detect the following characteristics of weather: rain, wind speed, air pressure, temperature, humidity, and cloud coverage. The roof will act preemptively according to the outside weather conditions, and is able to detect if unwanted weather is on its way, not just when it arrives. The sensors would be programmable to connect to an app, where when certain thresholds are met, the roof will operate automatically. Depending on the environment that the roof is placed over, and the climate it is built in, the roofs sensors could also be adjusted to work well in all conditions.

This system will also come with a locked app available on iPhone, that only allows owners of the roof access. This allows the user to control the system via app, and set their own thresholds for operation of the roof and lighting depending on the setting the roof encompasses. With this app, the user can build their own packages of settings and create a schedule on the roofs operation to fit business hours, daylight hours, etc. Along with providing full control of the system, the app will include the statistics read in from the weather sensors, allowing the user to make educated decisions when creating their own roof settings. The app will mainly connect to the sensor data and roof via Wi-Fi as well as having Bluetooth capabilities for backup. In case of emergencies such as a power outage, Wi-Fi outage, or sensor failure, the building will have a main kill switch inside which will override all app settings and sensor data and close the roof.

Another pressing issue in today's society is the matter of energy efficiency. Everyone wants to save money and energy whenever possible. This roof would assist in this by the addition of solar panels to power the sensors used for automation. Along with this solar energy, the indoor lighting would have photometric sensors and connect to switch to the roof, so that they are only on when it is darker outside and/or when the roof is closed. The HVAC systems would also have a kill switch connected to the roof, turning it off when the roof is open, so as not to waste excess energy. The roof would be fit with a tinting glass which helps for controlling the inside temperature, and the indoor HVAC would be able to adjust to a user specified temperature and humidity.

2.2 Objectives

Overall, this system aims to create a comfortable indoor experience for any market, especially businesses that can only operate in certain weather conditions. Energy and money is saved by the various attributes of the roof including, tinting glass, solar panels, photometric lighting, and HVAC control. According to our sponsor, OpenAire, their customers saw up to 30% in savings for one year of energy costs. Our goal is to increase this percentage via the automation of the roof. Lastly, the system is easily customizable by the user via iOS app, which allows for full customization of the roof's operation dependent on the needs of the building owner.

2.3 Engineering Requirements Specifications

The engineering requirement specifications are broken into the key aspects of the project: enclosure, hardware, and software. The specifications, and their subsidiaries, are laid out in their respective Tables 1 - 6.

Enclosure Specifications- The enclosure is the main structure that will support the roof and have the main unit housing the sensors and the PCB attached to it. Its specifications are compiled in *Table 1* below.

Table 1: Enclosure Requirements Specifications

Specification Number	Specification	Value	Unit
1	Dimensions	2x2x1	feet
2	Weight	< 30	lbs
3	Retractable Roof	1	N/A

System Requirement Specifications - *Table 2* lays out the specifications for the system. These specifications are those that involve the entire system, affect the entire system, and is used by the different components.

Table 2: System Requirements Specification

Specification Number	Specification
3	The system shall have a retractable roof
4	The system shall have a microcontroller capable of handling weather sensing data and signal processing
5	The system shall have a solar panel system not exceeding 8 cubic feet
6	The system shall have a lithium battery pack for back-up power, to be charged by the solar panel system
7	The system shall have a battery management system for handling charging cycles of the lithium ion battery pack
8	The system shall have an array of regulators providing +12V, +5V, and +3.3V power lines
9	The system shall have weather sensing technology capable of interacting with the microcontroller
10	The system shall have waterproof protection for all electronic components that require it

Hardware Specifications- The hardware specifications break down more of the specification mentioned in Table 3. The hardware is broken into two tables: power and sensors. The critical power hardware, including power sources and regulators. The sensor hardware requirements include all the necessary weather sensors for the project.

Table 3: Power Module Requirements Specifications

Specification Number	Submodule	Specification	Value	Units
11	Solar Panels	Power Delivery	1 – 5	Watts / Panel
12	Lithium Ion Battery	Package	18650	
13	Reverse Polarity	MOSFET Rds(on)	< 0.05	Ohms
14	12 V Regulator	Input Voltage Range	3.2 – 12.6	Volts
15	5 V Regulator	Input Voltage Range	≥12	Volts
16	3.3V Regulator	Input Voltage Range	≥5	Volts
17	BMS	Voltage Range	3.2 – 12.6	Volts
18	BMS	Fault Protection	Overcurrent Protection	
19	BMS	Fault Protection	Overvoltage Protection	

Sensor Specifications -The sensor specifications outline the main measurement of the given sensor, and the desired description of that measurement that will drive the selection of the component.

Table 4: Sensor Module Requirements Specifications

Specification Number	Module	Specification	Value	Unit
20	Anemometer	Resolution	< 10	meters/second
21	Barometric Pressure Sensor	Accuracy	+/- 5	millibar
22	Temperature Sensor	Accuracy	+/-1%	^{o}Celsius
23	Humidity Sensor	Accuracy	+/3%	Relative Humidity
24	Rain Sensor	Response Time	< 240	seconds
25	Light Sensor	Spectral Range	1500	nanometer
26	Bluetooth	Range	50	meters

Software Requirements Specifications- Software Specifications can be broken into two parts: microcontroller software specifications and the phone application software specifications. The microcontroller specifications include the software implementations for the board and the required storage for the software.

Table 5: Microcontroller Software Requirements Specifications

Specification Number	Specification	Value	Unit
27	Microcontroller is hard coded with an identification number	16	Bits
28	For specific sensors, their status is polled	60	seconds
29	The Weather API is polled for predictions	60	seconds
30	Microcontroller will signal a roof reaction when deemed necessary in an appropriate time frame from receiving measurements	30	seconds
31	Code will only require half of the microcontroller's available storage	128	kiloBytes
32	Weather API prediction data is stored on the microcontroller	32	kiloBytes

iOS Application Specifications - The phone application software specifications outline the expected procedure and protocols that the application will follow.

Table 6: Phone Application Software Requirements Specifications

Specification Number	Specification
33	Application shall provide exact parameters for Microcontroller to signal roof enclosure
34	Application shall support iOS 9.0 and above
35	Application shall support iPhone 6s Models and above
36	Application shall support Bluetooth 4.0 and above
37	Application shall decode sensor data in a human readable manner
38	Application shall show sensor data from last minute of reading
39	Application shall follow security protocol for storing Username Password information in the database.

2.4 House of Quality Analysis

The following shows our house of quality analysis when planning our project development. Out project aim is to be low cost, accurate, and maintain or improve upon our sponsors level of energy efficiency.

Legend*:*

- \+ Positive Polarity (Increase requirements)
- \- Negative Polarity (Decrease requirements)
- ↑↑ Strong positive correlation
- ↑ Positive correlation
- ↓ Negative correlation
- ↓↓ Strong negative correlation

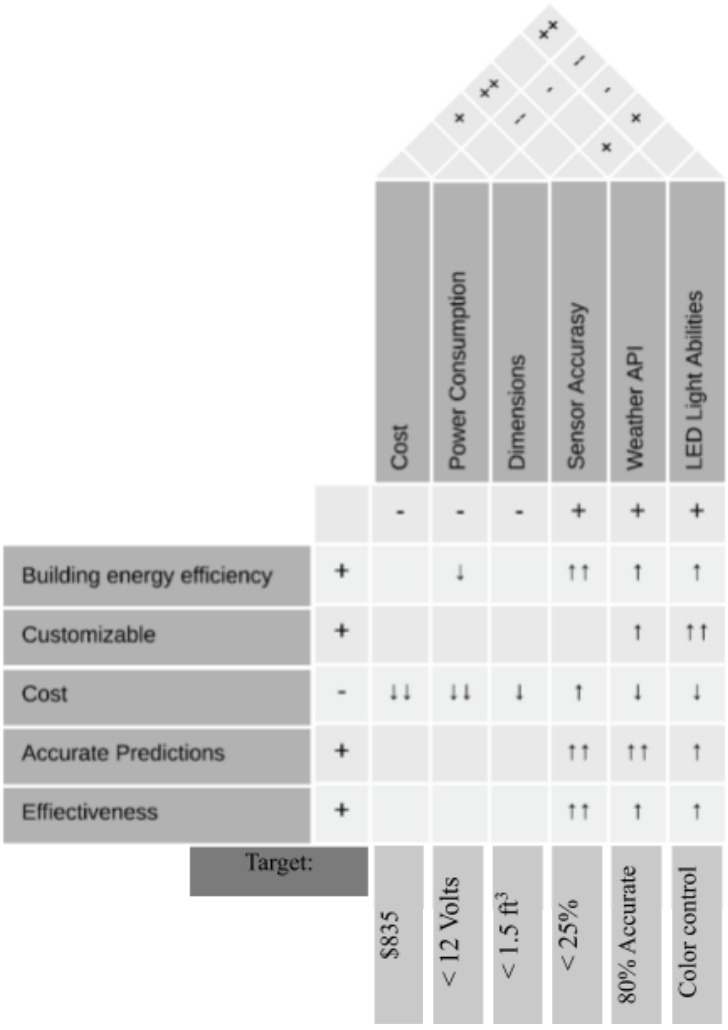


Figure 1: House of Quality Analysis

3. Initial Project Research

The high-level lay out for the hardware aspects for this project is depicted below. An iOS app is the main interface for users to interact and set certain parameters. The sensors and weather API is the driving force for the autonomous movement of the roof. The high-level logic for the sensors is depicted in the flow chart. The roles of the development and design for the different hardware aspects are color coded, with the key being located in the bottom right of the figure.

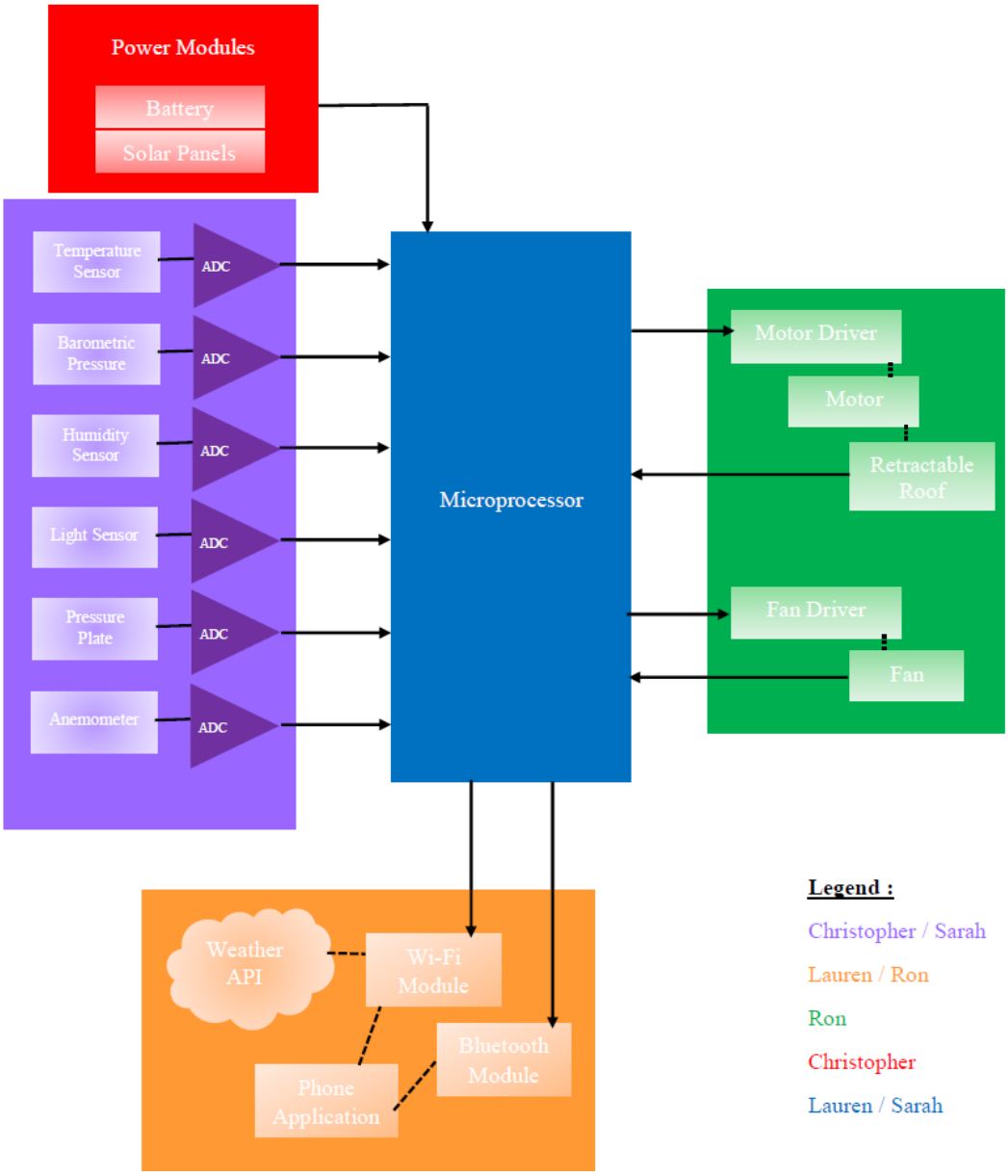


Figure 2: Hardware Flow Chart

3.1 Existing Projects and Products

Open Aire is a designer, manufacturer, and installer of custom glass retractable roof enclosures and operable skylights. As of now, their systems are only semi-automated, with proximity sensors to locate the position of the roof, anemometers to calculate the wind speed, and conductivity sensors detect rainfall. Overall, the company goal is to provide an enclosure for weather sensitive areas such as water parks, shopping areas, and other activities that are mainly outdoors. They also aim to provide businesses with a potential for a lifelong investment that will save them money by reducing energy costs and increasing year-round business.

Although our sponsor's product is one of the most trusted out of the few companies that provide this service, we would like to provide them with improvements to bring them levels above their competition. Our ideas for improvement include, updating their current sensors, or finding a better way to implement them, and adding new sensors for more accurate measurements of outdoor conditions. These new additions include sensors for temperature, wind, rain, humidity, barometric pressure, as well as the addition of solar panels for even more energy efficiency.

The addition of more sensors will allow the roof to become a smart system that can detect unpleasant weather before it happens instead of as it happens. We would also like to provide the owner of the roof with full interactive control of the roof system. Currently, Open Aire places most of the roof's operation in the hands of the few implemented sensors, and provides the owner with a control panel to override the roof's semi-automated operation. We propose an app/web server, which will connect to the roof system via Wi-Fi and Bluetooth. We will also include the data from the local weather API in an easy to read format, so the user can make educated decisions about which setting they want the roof to operate in. The weather API will also be compared to the roof's sensor information, and the user can receive alerts about inclement weather, roof operation, and roof failure. As a failsafe, we would like to keep a control switch somewhere inside the building in case the Wi-Fi or Bluetooth goes out.

Our stretch goals include the addition of lighting and HVAC controls, building our own light sensor, and power module shutdown. The lighting controller is connected to the light sensor, so that the indoor lighting will only turn on when the sunlight goes below a certain level and/or when the roof is closed. As to not waste energy and create a comfortable environment, the HVAC will only be on when the roof is closed, and it will include controls for temperature and humidity. Additionally, we would like to have the power module automatically shut down when there is a failure to any part of the system, thus requiring the user to manually reset the system via a switch on the control panel. We would also like to dedicate a control panel that controls different aspects and sensors of the roof, and another to control the roof system as a whole.

3.2 Relevant Technologies Comparison

Overall, this autonomous roofing system requires a wide range of sensing technologies and power supply technologies. The following sections analyze each type of sensor needed split into the many different types of sensors available. These in-depth comparisons allow us to wisely choose relevant technologies needed for this system to function properly.

3.2.1 Relevant Sensing Technology

For a fully automated system, a wide range of sensors is to be used. This section goes through the decision process of choosing each sensor used for this enclosure. Because this enclosure operates based on the weather, the five main characteristics must be measured. These include: wind speed, temperature, lighting, humidity, and rain. as well as using these, sensors must also be placed on the enclosure to measure and control the position of the sliding roof.

3.2.1.1 Proximity Sensing

The use of proximity sensors is helpful when controlling the operation of the roof. Unlike most of the other sensors we have equipped this system with, the proximity sensor will not be used any sort of weather sensing. It is used to determine the position of the roof as it moves, going from full open to full close. This design was originally used by our sponsor, so in order to stay consistent with them, we have decided to implement proximity sensors as well.

The current design by our sponsor, which we will also be using operates as described within this paragraph. Proximity sensors and targets are placed on opposite sides of the roof. Two targets is placed, one on the uphill side of the roof slider, the other on the downhill side of the roof slider The target is a steel or magnetic plate (in contrast to the aluminum structure used by our sponsor), and will sit on the slider, moving alongside the roof. The two proximity sensors that is implemented must stay stationary as the targets move alongside the roof. One proximity sensor is mounted to the motor at the bottom of the roof and the other is mounted to the top of the roof enclosure.

In proximity sensing there are five different types of sensors that could be used: inductive, capacitive, magnetic or hall effect, photoelectric, and ultrasonic. The main goal of proximity sensing is to be a non-contact sensor that can still determine when something is in range of it. Here, we would only like the proximity sensor to sense when our target is in range, and ignore any other object that could be within sensing distance. We would also like to consider cost, size, and how well this sensor fits into our design as a whole. The following paragraphs break down each type of sensor into their main characteristics.

Inductive Proximity Sensor - Inductive proximity sensors use either a PNP or NPN transistor along with an inductor to induce a current within the device, thus creating a signal to the output. When a metal comes into the sensing range of the sensor, the inductor creates an electric field, which creates a current, or magnetic field, within the target. The closer the target gets to the sensor, the weaker its electric field, causing a current to flow through the inductor to the output. An advantage to using a sensor like this, is that it only reacts to certain materials within its range. This specific material happens to be a ferrous material, or a material that contains iron. Luckily iron and iron alloys are easy to come by and don’t cost an arm and a leg, assisting us in our goal to keep this project as low cost as possible without sacrificing too much else.

Most inductive proximity sensors come in the shape of a cylinder, which makes them compact, perfect for our small-scale building. They can operate within a range of input voltages allowing us to choose one which fits our needs, and most have a maximum sensing range of 1.5-10mm. keeping

the size of our building, and target in mind, we can pick one which has a fast response time when the target is within the range that we see best fit.

Conductive Proximity Sensors - Capacitive proximity sensors operate on the notion of two capacitive plates placed some distance away from each other acting as an open capacitor [5]. This acts as an open because one of the plates of the capacitor is the proximity sensor itself and the other is the target we are trying to measure. When the target comes into sensing range of the sensor, the capacitance of the sensors changes, causing a change in its field amplitudes, thus causing an output change. What may seem like an advantage to most but is actually a disadvantage to this project, is the wide range of objects the capacitive sensor can detect. These sensors can detect various, liquids, powders, and metals, which could be cause for unwanted action as our structure its self is made of a metal.

Conductive sensors can sense wide ranges of materials, making them popular for water level applications, or in cases where an object must be detected through another object. They also have wide sensing range of up to 50mm depending on the sensor. This sensor is still applicable to us because they can be tuned to ignore certain objects, making it an option for our project.

Hall Effect Proximity Sensors - Hall effect, or magnetic, proximity sensors, operate on the principle of magnetism as the name suggests [1]. Utilizing the sensors magnetic field, an electric potential between the sensor and the magnetic target is created and grows as the target comes into sensing range. Once this potential reaches a threshold value, the current flows through the sensor to the output creating a control signal. An advantage of Hall effect sensors is that they can detect static magnetic fields, whereas inductive sensors can only detect changing magnetic fields. This allows for a wider range of objects to be detected.

A disadvantage of the hall effect sensor is that it is not always stable or accurate. In our application, we must ensure that the sensor only makes a decision based on the location of the target, so accuracy is key. These sensors require more compensation, regulation, and maintenance than would be ideal. Most hall effect sensors come in small sizes and can operate at various input voltages at around 4.5V-28V and an average response time of about 2 micro seconds. This makes it easy for us to find a sensor that could still be accurate if the correct regulating circuitry is applied around it.

Photoelectric Proximity Sensor - Photoelectric proximity sensors use optics, instead of a materials electromagnetic fields, to sense an object [5]. Mainly used as distance sensors the can sense an object up to 60m away. All photoelectric sensors have some sort of light source, whether it be an LED or a laser diode, a photodiode or phototransistor to collect the light, and an amplification circuit for the signal processing.

The light source emits a photon wave, which interacts with objects around it, and depending on the material that the light comes into contact with, a large or small amount of light is reflected back to the receiver [5]. Depending on the setup of the photoelectric sensor, the target could come into the range of the light, and reflect the light back to the receiver. The light could be on all the time at an angle that always reflects the light back to the receiver until the target cuts off the light reflection. Or, the target could diffuse the light until it reaches the perfect spot in which all light

waves converge toward the receiver. Either way, the target material or the reflector must be specific depending on the wavelength of the light emitted such that the light reflects perfectly back to the receiver. These specifiers of the materials used could prove difficult to our budget, or the overall difficulty of our project.

Ultrasonic Proximity Sensor - Last, an ultrasonic sensor could be employed. These sensors use the reflection of sound waves, in the same way the photoelectric uses the reflection of light waves. Here, however, the color and material of the object do not matter as long as the texture of the material isn't fully absorbing or extreme in anyway [5]. These sensors use one of the same three techniques as photoelectric sensors, except that they measure the propagation time of the sound wave as a way of detecting the closeness of an object. Most of these sensors can sense an object up to 2.5m away, which is far enough for our project. However, air quality could greatly alter the response of this sensor, especially since it is outdoors.

Proximity Sensing Technologies Conclusion - A final summary of the above sensors is described below in Table 7.

Table 7: Comparison of Proximity Sensing Technologies

Technology	Output Signal	Environmental
Inductive	Current	Safe
Conductive	Voltage	Safe
Hall Effect	Voltage	Safe
Photoelectric	Voltage	Safe
Ultrasonic	Current	Safe

For simplicity, we have decided to further analyze the inductive and hall effect sensors. These sensors only respond to certain target materials making it easier for us to find a material to use for our target. We also won't have to do much outside configuration of the sensor in order to force it to only sense certain materials. This leads to less error on the sensor's end and our end. These types of sensors often come in compact sizes, which will fit well into our already miniature design.

3.2.1.2 Anemometry

An anemometer is a device that measures wind speed. High winds are important to measure because they are usually indicative of bad weather, and can be dangerous for the activities going on inside the enclosure. There are two categories of anemometers, which can then be split further into four more types. All four sensors can be classified as either a constant-power anemometer or a constant-temperature anemometer. Both of which can be split into four subtypes: vane, thermal, thermal with velocity and temperature, and cup anemometers. All of these sensors is explained and analyzed, so we can make a better decision for this project.

Constant-power Versus Constant-temperature - Constant-power or constant-temperature anemometers receive either a constant power flow or constant flow of electrical heat, where the temperature is proportional to heat with no feedback, to power the device [6]. The most popular classification of anemometers is the constant-power as it has a higher frequency response rate, lower electrical noise, a more stable zero-flow, and are overall more compatible with other sensors and processors. Constant-temperature anemometers has a slower response rate, produce more noise, have limited temperature compensation, and are not very stable at zero-flow, or when no wind is present.

Vane Anemometer - A vane wind sensor uses a rotary plate with the axis of rotation placed parallel to the direction of the wind. Because the axis of rotation must stay parallel to the direction of wind, this style of anemometer is not applicable to our project [6]. In an unobstructed out door area the wind direction is constantly changing, so vane anemometers are usually employed in areas where wind direction isn't as variable.

Thermal Anemometer - A thermal anemometer use a very fine wire (micrometers in length), or a small element heated to a temperature above the ambient. When air flows over the element, it produces a cooling effect on it, signaling the presence of wind. Because a material's resistance is related to temperature, the change in resistance as the element is cooled, is used to measure the wind's velocity. Thermal anemometers can come as constant-current, constant-voltage, and constant-temperature [6]. Although most constant-temperature devices are slow and inaccurate, a hot wire anemometer is actually one of the most popular due to its fast response time and fine spatial resolution.

Thermal Anemometer with Velocity Profiling - Thermal anemometers with velocity profiling operate the same as thermal anemometers, except with multi-point data-logging. Almost all anemometers, no matter the type, have data-logging capabilities. However, this specific type can log the temperature as well as the velocity to measure wind speed and produce statistics about wind trends due to temperature [6]. This makes these anemometers applicable in data analysis for heat sinks and wind tunneling.

Cup Anemometer - The cup anemometer is one of the more simple designs, but still widely used today. Normally designed with three to four cups attached to horizontal arms connected to a vertical rod, this anemometer uses PWM and angular rotation to measure wind speed. The wind catches in the cups causing them to rotate, and the number of rotations within a certain time period is then used to calculate the wind speed using an average value over the specified time period. Each rotation sends a pulse signal to the microcontroller where the distance between each pulse is the instantaneous wind speed [6]. This speed is used to find the frequency at which the cups rotate, which can then be converted to a readable wind speed.

Anemometry Technologies Conclusion - The following table 8 concludes all findings about each type of anemometer.

Table 8: Comparison of Anemometry Technologies

Type	Classification	Output	Environmental
Vane	Constant-power	PWM	Safe
Thermal	Constant-temperature	Voltage, Current, or PWM	Safe
Thermal with Velocity	Constant-temperature	Voltage, Current, or PWM	Safe
Cup	Constant-power	PWM	Safe

From these comparisons, we have decided to look further into the cup and hot-wire styles of anemometers. The cup anemometer is already employed by our sponsor and requires no calibration or extra setup. However, we would like to see if this design could improve at all using a hot-wire thermal anemometer.

3.2.1.3 Barometric Pressure Sensing

One of the main determining factors of inclement weather is pressure. Air rises in a low-pressure area and falls in a high-pressure area. In a low pressure area the rising air cools and this is likely to condense water vapor and form clouds, and consequently rain. The opposite is true in a high-pressure area, which is why high pressure tends to give cloudless skies. More specifically for this reason, it became quickly evident that a pressure sensor was necessary for the purpose of the project. This section will discuss the different barometric pressure sensors that were considered and give insight as to why the sensor chosen was the best fit for the AIRE system.

There is a large variety of pressure sensors out in the market, however, the sensors can largely be categorized according to the type of pressure measurement they make, the sensing principle employed, the output signal and the media they're measuring. These were all factors that were considered and evaluated when choosing a sensor.

Right off the bat, it was known that this pressure sensor was specifically needed for the outdoors and its weather, so that removed any sort of air, gas, water, liquid, pneumatic and hydraulic, or any other pressure sensors. In other words, out of the three different types of pressures can be measured: gauge, absolute, and differential, an absolute pressure sensor was needed, as these types of sensors are best suited for measuring atmospheric pressure. There are 5 different ways (Resistive, capacitive, piezoelectric, optical and MEMS) in which the mechanical displacement taking place inside a sensor is turned into an electrical output. This is important to understand, as this factor influences accuracy, reliability, measurement range, and compatibility with the target environment. Absolute pressure sensors usually use the latest microelectromechanical system (MEMS) technology.

Going back to the fact that a barometric sensor was the one suitable for the project, the team had to go with the industry norm, which usually means that the sensor is a small pressure transducer. Even after understanding how the sensor outputs information, there was more to be considered, and that was the measurement approach of the transducer, either resistive or capacitive. As the team was choosing the different suitable options the main variables considered were precision, sensitivity, maximum limits, energy consumption, and size.

3.2.1.4 Temperature Sensing

Temperature is the measure of heat within an object or substance. For this project, a weather resistant component is needed as degradation in the sensor's accuracy is not ideal. A change in temperature is caused by the change in density of the molecules in the air, and by how much moisture each molecule is retaining. Because temperature is affected by the properties of small molecules, an accurate sensor is required for this design.

In temperature sensing there are four different types of the main component of the circuit. These are: thermocouples, resistance temperature detectors (RTDs), thermistors, and semiconductor based integrated circuit (IC). Each type is examined in the following sections, as well as their accuracy and other relevant characteristics.

Thermocouples - Thermocouples measure temperature difference by the Seebeck Effect [7]. They are constructed of two dissimilar wires joined together, and the difference in temperature between the two wires produces a voltage. The resulting voltage can then be used to calculate the ambient temperature. This configuration is highly sensitive, ranging from ten to sixty-eight microvolts per degree Celsius.

Although high sensitivity is ideal for a more accurate reading of temperature, this can create problems in building the circuit around the thermocouples [7]. Because their output voltage is so small (microvolts), a precise amplifier must be placed at the output, and the signal is highly susceptible to external noise from long wires, and cold junction from any copper that can be found on the circuitry.

Resistance Temperature Detectors - Resistance temperature diodes (RTDs) measure a material change in resistance to measure a change in temperature [7]. As any material goes through a temperature change, its resistance also changes, thus creating a different output voltage. The most accurate design is made of platinum, as it has an almost linear response to temperature change. This design also includes four wires, two of which separate force and sense, thus cancelling any wire resistance or noise that could deplete the sensors accuracy. They also have the largest temperature measurement range out of any sensing mechanism examined, which includes -200 to +850 degrees Celsius.

Although accurate, RTDs are extremely sensitive to external forces such as light, moisture, and other various characteristics of weather. They also have a slow response time, and are susceptible to self-heating. Because of this, they require specific maintenance regulated by the American Society for Testing and Materials (ASTM), American Scientific Apparatus Manufacturers Association (SAMA), International Electrotechnical Commission (IEC), and Japanese Standard

(JIS).Because of these constraints this type of temperature sensor wasn't deemed as a feasible option by the team. Luckily we had multiple choices when trying to find a different type of temperature sensor to use.

Thermistor - Thermistors operate similarly to RTDs, as they measure temperature change by measuring the materials change in resistance. Slightly less accurate, they are, on average, less expensive than the RTDs, as well as made from either polymer or ceramic. The material of the thermistor makes it a better option for outdoor use, as these materials are less susceptible to corrosion or degradation due to weather.

The most widely used thermistor is a Negative Temperature Coefficient (NTC) build, in which the resistance decreases as the temperature increases [7]. From the American Society for Testing and Materials (ASTM) E879, NTC thermistors must operate at 100 ohms at 0 degrees Celsius, or 10k ohms at 25 degrees Celsius. Thermistors are fairly accurate, with a resistance tolerance of 1% and a measuring range of -55 to +150 degrees Celsius.

Integrated Circuit - Semiconductor based integrated circuit come in two different types: local and remote digital temperature sensing. Local temperature sensing measures the change in temperature of a diode within the circuit, via the physical properties of a transistor [7]. Remote digital temperature sensors measure the temperature of an external resistor. A local temperature sensor can be used to measure the temperature of the PCB or the ambient air around it, whereas the remote sensor only measures the ambient air, as it is placed away from the main PCB.

Local temperature sensing would not be applicable in this case as the heat emitted from the PCB would cause the ambient air surrounding the diode, therefore not give an accurate reading of the outside temperature. Remote sensor would be more applicable as the extra diode is placed away from the main circuit, thus producing a more accurate temperature reading. Either way, IC temperature circuits produce almost linear outputs with respect to temperature and operate within the range of -50 to +150 degrees Celsius.

Temperature Sensing Technology Conclusion - A final comparison of the technologies examined above as well as their RoHS compliance is summarized below in Table 9.

Table 9: Comparison of Temperature Sensing Technologies

Technology	Output Signal	Environmental Impact
Thermocouple	Voltage	Hazardous
RTDs	Voltage	Safe
Thermistor	Voltage	Safe
IC	Digital or Analog (Voltage/Current)	Safe

For simplicity and for the environment's sake, we have decided to implement a thermistor into our project. Thermistors are easy to implement, even though they require a simple regulating circuit. They are also the best option for measuring outside air, as they aren't affected much by extreme weather, and can be placed away from the rest of the circuitry, such that its measurement isn't skewed by the heat emitted from the PCB.

3.2.1.5 Humidity Sensing

Humidity is the measure of moisture within the air at certain temperatures. There are three different types of humidity sensors, or hygrometers: capacitive, resistive, and thermal. Determining relative humidity is dependent on the kind of sensor and the type of dielectric that is used in them. For this project, accurate measurements are important for effective roof functionality. Precision takes a back seat when it comes to the project's measurements as drastic weather events will cause large enough changes in the humidity. The different kinds of sensors and their accuracy and other characteristics are examined in the following sections.

Capacitive Humidity Sensor - Capacitive humidity sensors measure electrical capacitance when moisture makes contact with the sensor. Capacitive humidity sensors are very precise, as the moisture changes, the output voltage from the electrical capacitance changes [8]. Capacitive humidity sensors can operate in humidity in the range of 0% to 100%, while some have more accurate readings within that range. Most capacitive humidity sensors range from about 150 pF to 200 pF for capacitance.

Resistive Humidity Sensor - Resistive humidity sensors are similar to capacitive sensors, in how they function. When moisture is absorbed by the hygroscopic material within the sensor, the change in resistance is measured. Resistive humidity sensors have a smaller range in measuring humidity, only between 5% to 90% relative humidity can be measured. In lower humidity ranges, resistive humidity sensors tend to perform poorly. Lower temperatures affect the performance the sensor as well.

Humidity Sensing Technologies Conclusion - The following table provides a brief summary of the types of humidity sensor relating to their output and environmental effect.

Table 10: Comparison of Humidity Sensing Technologies

Technology	Output	Environmental
Capacitive	Digital/Analog	Safe
Resistive	Voltage	Safe

In order to provide our system with the most accurate design, we have decided to further analyze both types of humidity sensors. Both sensors have their strengths and weaknesses, so a choice for this sensor would be better made if multiple different sensors were looked into. This in-depth discussion is seen later on in section 3.3 Parts Selections.

3.2.1.6 Rain Sensing

For this project, a major component is a rain sensor in order to keep all activities within the enclosure ongoing, even during bad weather. Bad weather is usually indicative of high winds, cloudiness, and rainfall. To detect any level of rainfall, a precise sensor is needed. Rain sensors come in four main types, water collecting, conductive, optical, and through the use of a hygroscopic disk. Each of these measures rainfall in a unique way, and have their own properties which makes them useful for various different applications. The following sections compare each type of sensor with its main application and ends with a summarized conclusion of our decision.

Water Collecting - Water collecting sensors are usually some sort of cup or basin that acts as rain gauge. They collect the rain, and once the basin collects a preprogrammed amount of water, the weight trips a switch which sends a signal to the control system it is connected to [9]. This sensor is simple, and undeniably easy to implement; however its simplicity can cause problems from the accuracy stand point. Because its design is basically a cup, the weight of debris that might have gotten in can cause the switch to trip prematurely.

Another inconsistency with these basins is that if they are too large, a gust of wind could blow out anything in the cup causing the switch to not trip at all. For this project, we need a rain sensor that will send a signal at even the slightest detection of rain, a.k.a. a couple drops of rain. such small amount of rain will not trip this type of sensor in time before everything in the enclosure becomes soaked with rain.

Hygroscopic Disk - Hygroscopic disk sensors, made to replace the outdated water basin, utilizes the expansion of materials when water is absorbed, to trip the signal switch [9]. These are made of a synthetic material, most similar to cork, that expands when wet. As this material collects more and more water, it expands until it hits a switch at a certain, preprogrammed, size point. The system will not return back to normal until the material has fully dried and its size has gone back to its base size. Similar to the water basin sensing technique, this type of sensor is not near accurate enough for this project. Its response time is too slow, and it needs a too large amount of water in order for it to signal any rain detection to the system. Therefore, other types of sensors must be analyzed.

Conductive - Conductive rain sensors use capacitors along with the change in conductivity of a material as it encounters different solutions to detect rainfall [9]. These are normally designed as copper traces laid out on a flat PCB. Utilizing the effect of fringe fields, the electric fields emitted from the capacitive plates, not only flow between the plates, but also to its surroundings. The conductance on the plates when surrounded by the ambient air is noted as normal, and the conductance of the material when in contact with water is also noted and preprogrammed into the system. When the conductance of the material changes due to an interaction with a new solution other than air, the output voltage signal sent to the system changes. When this signal matches those of the ones programmed to mean water, the system reacts as programmed.

This sensor is much more useful to us as it is far more accurate than the water basin and hygroscopic disk approaches. Usually with a base capacitance along the order of 2-20pF with a change in capacitance normally around 0.5pF, even the smallest change in capacitance can cause

a large change to the output. Ideally, these sensors work best when the base capacitance equals the change in capacitance, however a slight difference between the two values will not cause a large loss of accuracy.

Optical - Optical rain sensors operate based on the properties of optical electromagnetic fields. Utilizing the principles of boundary conditions to sense light, these are some of the most popular rain sensors on the market as of now [9][2]. A pane of glass in placed over an infrared light, processor, and light receptor. When no moisture has built up onto the glass, almost all of the light emitted by the sensor is reflected back to the light receptor. As more moisture is built up, the light emitted by the infrared light is scattered by the water, and less light is reflected back to the receptor. At this point, the processor is programmed to different settings to determine the intensity of the rainfall. For this project, even the slightest detection of rainfall is enough to send a signal to our control system, thus closing the roof.

Rain Sensing Technologies Conclusion - A final summary of each type of rain sensor with respect to output and the environmental effects is seen below in Table 11.

Table 11: Comparison of Rain Sensing Technologies

Technology	Output Signal	Environmental Impact
Water Basin	Switch	Safe
Hygroscopic Disk	Switch	Safe
Conductive	Voltage	Safe
Optical	Voltage	Safe

In order to keep this system as accurate as possible, the obvious choices are between the conductive and optical sensors. Although the water basin and hygroscopic disk sensors are used widely in irrigation and sprinkler systems, they require a much larger amount of rain in order for a system shutoff to be worth it. Here, we need a more accurate sensing mechanism for our project purposes, which can be achieved by either conductive of optical sensing techniques.

3.2.1.7 Light Sensing

A light sensor, though not absolutely necessary for weather prediction, is still helpful in our goal in creating a comfortable environment inside the enclosure. For this we have two options: buying and implementing a premade light sensor or building our own. Both options provide different benefits to this project.

When buying premade sensors, we know we receive accurate results with very little error from the sensor, and there are also many resources available for easy implementation in the system. When building our own light sensor, we gain a deep understanding of how the sensor is operating and how to implement it, as well as full customization of the sensor in terms of threshold measurements, shape, and size.

In the following sections, we is comparing the different technologies utilized to measure the intensity of light flooding into our enclosure. The two most popular technologies used are the photoresistor and photodiode [3].

Photoresistors - Photoresistors are resistors whose resistance is sensitive to light. When dark, the photoresistor have very high impedance (MΩs), while in light, the photoresistor will have very low resistance. The most common material that makes up a light-dependent resistor is cadmium sulfide or lead supplied.

Since it acts as a passive device, the circuitry required to measure the light it receives is a very simple non-inverting amplifier. This means that less circuitry is required to integrate the photoresistor into our project. On the other hand, the materials often used to manufacture these photoresistors are highly toxic both to humans and the environment. Having a photoresistor made of lead of cadmium sulfide would be in stark contrast to our goals of creating an environmentally friendly, economical autonomous roof system.

Photodiodes - Another technology used to measure light is the semiconductor device of a photodiode. Photodiodes are often constructed using similar material to any other semiconductor device, typically silicon.

The main difference between the photodiode and photoresistor is the fact that the illumination measured by a photodiode is in terms of current produced by the photodiode as opposed to resistance measured by the photoresistor [4]. This will require a transimpedance amplifier in order to measure the light received in terms of voltage input to our analog-to-digital converters.

While the photodiode might require more circuitry to integrate, it is much more environmentally-conscious than a photoresistor.

Phototransistors - The last technology we is comparing is the phototransistor. This component has a similar relationship to a photodiode as does a regular transistor has to a regular diode. Similarly, it has the same advantages and disadvantages to the photodiode technology.

As opposed to a photodiode, the phototransistor can handle its own gain in order to amplify the signal it produces. This would mean that it will simply need to be placed in its own amplifier stage instead of using a transimpedance amplifier. The difference in integrability to the photodiode is negligible.

The use of a semiconductor in this project would be useful as we already know how to build an amplifier around diodes to achieve a desired output. Some of the other light sensing technologies require new circuitry which isn't as familiar to the team. Here, the external component count isn't as limiting as the complexity and familiarity of the circuit its self.

Light Sensing Technologies Conclusion - A summary of our comparison is established in *Table 12* below. This summary also includes any environmental impact these sensors may have.

Table 12: Comparison of Light Sensing Technologies

Technology	Output Signal	Environmental Impact
Manual	Manual Button	Safe
Photoresistor	Resistance	Hazardous
Photodiode	Current	Safe
Phototransistor	Voltage	Safe

In conclusion, to keep our project both environmentally friendly and simple, we opted for the semiconductor component to use for our light sensing technology of choice. Out of the two remaining options, we used the photodiode since we are more familiar with transimpedance amplifying topology than the other designs that would be required to integrate a phototransistor into our project.

3.2.2 Relevant Power Delivery Technology

To power the whole unit: sensors, motors, and communication modules included, solar panels and a lithium battery with be utilized together. In order to meet the hardware's power consumption and also maintain optimized energy efficiency, the hardware technologies for the power modules is analyzed and compared to find the best technology to use for the project.

3.2.2.1 Solar Panels

The solar panel options that we is exploring is polycrystalline (photovoltaic) or monocrystalline. The difference between the two lies in their production. A monocrystalline panel is constructed from a single ingot of silicon which has been pulled slowly from a vat of molten silicon. A polycrystalline panel is constructed from a vat of molten silicon, that is allowed to cool down, where different crystalline structures are formed inside the silicon ingot.
The two different processes result in solar panels with different efficiencies, price ranges, and reliability. By comparing several products that have abundant availability on Digikey.com, we will decide on which solar panel to use to power the autonomous roof system.

Due to the similarities between the two technologies, both types is considered for use in our project. Further research on which hardware to use is conducted and summarized in section 3.3.2.1 with specific parts.

3.2.2.2 Batteries

In the event of cloud cover preventing the solar panels from generating sufficient energy to be used by the autonomous roof system, the system will have a back-up battery pack capable of providing

power to the system until solar power is restored. According to Battery University, the most commonly used batteries in commercial use are lead-acid batteries and lithium-ion batteries.

Compared to the other options on the market, the lead-acid battery is very inexpensive. It has been in use for many decades, meaning that the technology used to control this type of battery is very mature and has a heavily-developed support network to provide application notes for dealing with the lead-acid battery. On the other hand, the lead-acid battery has very poor energy density and can only be used for occasional full-discharge cycles.

Compared to lead-acid batteries, lithium-ion batteries have much larger energy density and many available charge/discharges cycles before needing maintenance. In addition to the improved energy density, lithium-ion technology lasts much longer in storage while being capable of discharging more often. While it boasts an impressive list of advantages, the lithium-ion battery requires a battery management system to protect the circuit it is powering as well as itself. Also, the technology is relatively new and, while it does have extensive application notes for dealing with technology, aging of the lithium cells can have a detrimental effect on their performance. Lithium ion batteries are also much more expensive than lead-acid batteries.

Battery Technology Conclusion - From this simple comparison, it is clear to see that a lithium-ion battery system is much better suited for our application.

3.2.2.3 Voltage Regulation

There exist many different types of voltage regulation technologies. In our initial research, we will research the different voltage regulation topologies to see which one is best for our use.

Shunt Zener Regulator - One of the simplest voltage regulators would be a Zener diode-based circuit. A Zener diode, reverse-biased, will maintain its constant voltage across itself while directing any excess current to its negatively-based terminal.

In this circuit diagram, V1 represents the solar panel. The diode D1 will prevent current from flowing into the solar panel, avoiding damage. The resistor R1 will limit the current being passed to the load, RL. The Zener diode will have a certain Zener voltage. In our case, the Zener diode will need to have a Zener voltage of 3.6 - 4.2 V, which will match the nominal voltage of our battery.

The main advantages of this circuit include its small budgetary footprint and simplicity. Consisting of only a few components, the Zener diode voltage regular will consist of a bill of materials that will not exceed the budget set aside for this module. In addition, the components can be configured simply for the lithium battery.

The main disadvantages of this circuit include its inflexibility, large power loss, and poor voltage regulation. While simple, the Zener diode voltage regulator will only have provided a reference voltage of 3.6 – 4.2 V, meaning that it cannot be connected to a different battery, or may run into issues when the lithium battery exceeds 4.2 V. In addition, the circuit loses much of its power through a current-limiting resistor. Lastly, if the load current changes drastically, then the Zener

voltage will also change, meaning that the output voltage will drift from its nominal value of 3.6 – 4.2 V.

Linear Voltage Regulator - Another option for a voltage regulator would be a linear voltage regulator. These voltage regulators are useful for regulating a voltage at low power levels.

In these types of configurations, a low-dropout linear voltage regulator provides powerful voltage regulation with lower power loss than a Zener shunt voltage regulator. With the LM317 linear voltage regulator, the regulator takes a more active role in regulating the output voltage to the battery. The main disadvantage to this is that the LM317 is relatively inflexible compared to a much more sophisticated battery management integrated circuit (BMIC).

However, much like the Zener regulator, the linear voltage regulator in this configuration has no feedback for properly controlling the output voltage to the battery. In addition to this, the linear voltage regulator will only charge the battery to the specified reference voltage. This regulator will not prevent the lithium battery from overcharging, which can lead to a degradation of the lithium cells and reduced lifetime of the battery systems.

Buck/Boost Voltage Regulator - The third option for a voltage regulator would be a switching voltage regulator. These voltage regulators are useful for regulating a voltage at higher power levels than a linear voltage regulator.

The main difference between the switching voltage regulator and the linear one is that the switching voltage regulator can convert voltage much more efficiently. This configuration wastes very little power and utilizes a switching mechanism that controls a transistor which drops voltage over its very low-power-wasting silicon.

The switching voltage regulator, however, produces a lot of broadband noise which can affect other components operating nearby. When utilizing our Wi-Fi or Bluetooth modules, the noise from this switching voltage regulator may affect the modules' abilities to transmit or receive the high frequency signals it interacts with.

Battery Management IC - The last option that we are considering is to construct our own battery management system that utilizes a specialized battery management integrated circuit (BMIC). This provides the most flexibility and allows us to charge the lithium battery without worrying too much about overvoltage and excessive thermal runaway.

The chief advantage of a dedicated BMIC is the active role the integrated circuit takes in accurately regulating the charging cycle of the lithium battery. The BMIC operates similarly to the switching voltage regulator, while having extra features that allows it to account for overcurrent and excessive thermal anomalies. In addition to the incredibly high efficiency, the exceptionally low Rds(on) resistance of the transistor controlling the input/output current means that very little power is dissipated in the transistor.

The main disadvantage of the BMIC is its complexity. The extra features encased in the BMIC make it more prone to damage compared to simpler systems like the linear and switching voltage

regulators. Despite this downside, the BMIC is the most cost-effective solution to implementing a battery management system while also taking up a comparable amount of printed circuit board space to the other considerations.

Voltage Regulation Conclusion - While weighing all relevant factors, *Table 13* was constructed that organizes everything in an easy-to-reference format to accurately consider all solutions.

Table 13: Voltage Regulation Technologies Comparison

Regulation Technology	Average Bill of Material Cost ($USD)	Active Management	Thermal Shutdown
Shunt Zener Regulator	$3.00	0V-Reference	No
Linear Voltage Regulator	$4.00	0V-Reference	No
Switching Voltage Regulator	$8.00	0V-Reference and Feedback	Yes
Battery Management IC	$5.00	0V-Reference, Feedback	Yes

From the table we constructed comparing the voltage regulation options, we decided that constructing our own battery management system utilizing the battery management integrated circuit (BMIC) would be the best candidate for its role in managing the charging cycles for the lithium ion battery pack.

With its active role in regulating the charge cycles for the battery and preventing over-voltage and overheating, the BMIC is the most powerful candidate for the battery management system while keeping costs and board size low. It will also be capable of handling the amount of power that the solar panel will provide (5 W).

For all other power regulation, buck/boost converters is used to efficiently maintain voltage and ensure that there exists power lines for +12 V, +5V, and +3.3V.

3.2.2.4 Reverse Polarity Protection

The lithium ion battery pack is a component of the power module that is expected to be replaced or tinkered with relatively often. Because of this, there is an increased risk of user error when re-installing the lithium ion battery pack into the power module.

To protect the output voltage regulators from an unexpected negative polarity that could be experienced when installing the lithium ion battery pack to the power system incorrectly, a reverse polarity protection mechanism is implemented. The most common method for doing this is to use

a semiconductor component such as a diode or p-channel MOSFET. These methods is compared to come up with the most efficient solution to this common problem.

Diode - The simplest way to provide for reverse polarity protection is to connect a diode that will only allow current to flow through a circuit if a positive voltage is applied where it is expected to. If a battery is placed into the system with its polarity reversed, then the diode will prevent current from flowing into the system, or into the battery, which protects the circuitry on the other side of the battery.

The main issue with this system is the power dissipated across the diode at high current loads. Using the equation $P=I \cdot V$, the power can be calculated assuming a diode forward voltage of 0.6 Volts and expected current load of 2 Amps. The power dissipated will then be a whole 1.2 Watts. This is a lot of power wasted in just the diode.

P-Channel MOSFET - The p-channel MOSFET topology is another option to provide our system's reverse polarity protection.

The MOSFET will turn on when the battery polarity, V_bat, is positive, and current will flow into the load circuit. When the battery polarity is negative, no current will flow. This topology has the added benefit of having an exceptionally low R_ds(on) , which will dissipate very little power when the battery is connected and supplying power.

Reverse Polarity Protection Technologies Conclusion - From the comparison conducted, it was easy to conclude that the p-channel MOSFET topology would be the most efficient solution to our reverse polarity protection circuit. The MOSFET allows for a very small amount of power to be wasted in the protection circuitry, which is the best for our expectation of a large amount of current to supply the plethora of sensors and motors.

3.3 Part Selections

Following the above comparisons of relevant technologies, an even more in-depth comparison of our chosen technologies are done in the following sections. Each chosen technology is compared amongst the wide range of products offered. The main considerations for this project when choosing parts include: cost, environmental impact, ease of implementation with extra components required, and accuracy. All parts is chosen with these in mind as well as any relevant standards and constraints that may limit our choices when purchasing and implementing these products.

3.3.1 Sensor Components

For a fully automated system, a wide range of sensors is to be used. This section goes through the decision process of choosing each sensor used for this enclosure. Because this enclosure operates based on weather, the five main characteristics must be measured. These include wind speed, temperature, lighting, humidity, and rain. as well as using these, sensors must also be placed on the enclosure to measure and control the position of the sliding roof.

3.3.1.1 Proximity Sensors

As concluded in section 3.2.1.1. we have decided to look further into the implementation of the inductive and Hall effect proximity sensors. To do so, we is looking at a few of each type of sensor. The inductive sensor, as recommended by our sponsor, is the Alselectro Metal Inductive Proximity Sensor, another inductive sensor GX-F12A by Panasonic, and the Hall effect sensor is the MP102103 Hall effect Sensor by ZF Electronics. For this we is looking at how it reacts to various materials, sensing range, response time, and power consumption.

Inductive Proximity Sensor - The Aleselectro Metal Inductive Proximity Sensor was recommended to us by an employee of our sponsor because it is the most similar to ones that are currently in use by OpenAire. This one is cylindrical in shape and only detects metal objects. Operating in a range of 6-24V DC it consumes a moderate amount of power, and senses an object up to 4mm away. This sensor best detects objects that are made up of a ferrous material, while nonferrous materials can decrease the sensing range up to 60%. The main selling point of this device is that is durable in outdoor environments against dirt and moisture. The main downside for us it that this sensor retails at about $260.00 USD with a grand total of $920.00 USD after fees, taxes, and shipping, making it just above our price range for a sensor.

The Panasonic GX-F12A is much less expensive sensor at $24.26 USD which is well within the amount we are willing to spend, however we must decide if it works well enough for us to implement. This sensor has a maximum sensing distance of 4 mm, stable sensing distance of 3.3 mm, and a variation of +/-8% at maximum. This sensor requires 12V-24V DC +/1 10% which is within the same range as the Alselectro sensor. This sensor is also durable as it went through multiple shock tests in production and is encased in plastic

Hall Effect Proximity Sensor - The MP102103 Hall effect sensor Operates in the same range as the two inductive sensors of 4.5V-24V DC [a]. It has a maximum sensing range of 4mm and is housed in a glass reinforced casing. By inspection, this sensor is similar to the inductive sensors, however its working motivation is driven by different physics. Also different from the inductive sensors, Hall-Effect sensor all seem to be available at lower price ranges. Compared to the inductive counter parts, the MP102103 has a cost of $5.41 USD. However it is known that magnetic proximity sensors need more regulation than the inductive ones, but the amount of available resources will assist us in making any changes needed to our circuit in order to produce an accurate measurement.

Below, in Table 14, is a summary of the above paragraphs followed by a final decision for our preferred proximity sensor, and why it was chosen.

For this sensor the clear winner is the MP102103. It has low power consumption, and has the lowest cost out of all sensors analyzed. Although Hall effect sensors need regulating, the multitude of resources available to us can assist us when picking a target that will generate the highest signal, making the sensor ignore any other material that might be in range.

Unfortunately, due to the COVID-19 pandemic our team ran into some problems accessing the labs available to us on campus. Because of this, we were unable to do the proper testing for this part and had to omit it from our final design. This worked out in the end as our final design, explained in section 8, didn't end up needing the proximity sensing mechanism.

Table 14: Proximity Sensor Options Characteristics

Type	Part Number	Supply Voltage	Measuring Range	Sensing Objects	RoHS	Cost
Inductive	Proximity Sensor Metal	6-24V DC	4mm	Ferrous Metal	Yes	$920.00
Inductive	GX-F12A	12-24V DC	4mm	Ferrous Metal	Unknown	$24.26
Hall Effect	MP102103	4.5-24V DC	4mm	Magnetic Material	Yes	$5.41

3.3.1.2 Anemometers

As concluded in section 3.2.1.2. we have decided to have a more in-depth discussion about the thermal and cup anemometers. This was concluded based on how applicable these sensors are to our project, and how fast they respond. We is taking cost and size into consideration as well when picking this part. Below is a comparison of two different anemometers: the Jacksking cup anemometer and the Adafruit 1733 cup anemometer.

Cup Anemometers - The Jacksking cup anemometer operates at a range of 12-24V DC and has an accuracy of +/-0.3m/s. It is made out of an aluminum alloy making it lightweight, and good fit for up to place on top of our enclosure. It requires a start wind speed of about 0.5m/s which is more than enough considering the American standard of measuring wind as miles per hour. It operates in a fair range of -40 to +50 degrees Celsius, and can still operate in 100% relative humidity [b]. This is appealing because it makes the sensor durable, and able to withstand any weather conditions it may encounter at the top of our structure. With a response time of less than 5 seconds we know it will give us an almost real time insight as to what the conditions are like outdoors, allowing us to provide the roof with the most current information as it happens.

The Adafruit 1733 is a bit smaller than the Jacksking anemometer, and only needs an input of 7-24V DC to operate accurately. Measuring up to 70m/s wind speed with a resolution of no more than 1m/s we know it is accurate, and can respond quite fast. At about $44.95 USD it costs more than the Jacksking anemometer by about $15.00. Despite the cost, we think this would be a great sensor to implement besides one fact. The problem we have encountered with this anemometer is that it contains led in its material makeup. This means it is not compliant with RoHS standards, which is a major concern for this project since most of our sensors is outside, and all of them is around people.

Thermal Anemometer - As much as we wanted to find a hotwire anemometer to test, we could not find one that wasn't already fully built with its own measurement screen and converter. Unfortunately, this sensor would be have to be built fully from scratch, which is not a part of our current goals for this project. As a stretch goal we would like to design and test a hot wire anemometer.

The following table 15 displays the main characteristics of the sensors discussed, followed by our final product conclusion.

Table 15: Anemometer Options Characteristics

Type	Part Number	Supply Voltage	RoHS	Cost
Cup	B07SN1V427	12-24V DC	Yes	$29.99
Cup	1733	7-24V DC	No	$44.95

Based on all of the information and products available to us, we have decided to implement the Jacksking anemometer in our project. Recommended by our sponsor, the cup anemometer is our best choice for our specific applications. Ultimately, this decision came down to price and RoHS compliance. Because the Adafruit 1733 is more expensive and not compliant with the RoHS constraints, we cannot use this component in our final design.

Again, our team ran into issues testing this device. We have since decided to omit this part from our design even though we considered this to be integral to the weather prediction aspect of our project. We were unable to get this product tested and wired correctly to our design as it came in too late before on campus labs shutdown.

3.3.1.3 Barometric Pressure Sensors

As the team narrowed down the choices, the decision came down to three sensors outlined in the following table. It appears as though these are the most common technologies used when interfacing sensors with microcontrollers, which is why they were all similar. The three sensors all fit our needs, so it was small differences that lead to our final decision. Each of the main variables were looked into and is discussed for each of the mentioned barometric sensors.

Table 16: Barometric Pressure Sensors Comparisons

Part Name	Output Signal	Extra Features
MPL115A2	I2C	Temperature Sensing
BMP280	I2C/SPI	Altitude Sensing
MS5607	I2C/SPI	Altitude Sensing

The MPL115A2 is an absolute pressure sensor with a digital I2C output targeting low cost applications. It employs a MEMS pressure sensor with a conditioning IC to provide accurate pressure measurements. The MPL225A2 has an accuracy of plus-minus 1 kPa when operating within the temperature range of -20ºC to 85ºC.

The limits of the sensor are 50 to 115 kPa when it comes to pressure, and operates under the wide temperature range of -40°C to +105°C. The VDD Power Supply Connection range is 2.375V to 5.5V, Low current consumption of 5 A during Active mode and 1 A during Shutdown (Sleep) mode. These specs all hint towards a device that can sustain demanding environmental conditions. When it comes to size, the sensor is a miniature 5 by 3 by 1.2 mm LGA package.

The Adafruit BMP280 I2C or SPI Barometric Pressure & Altitude Sensor is based on Bosch's proven Piezo-resistive pressure sensor technology featuring high EMC robustness, high accuracy and linearity and long-term stability. The relative accuracy (950 … 1050hPa @25°C) is equal to ±0.12 hPa, equiv. to ±1 m, while the absolute accuracy (950 ...1050 hPa, 0 ...+40 °C) is yp. ±1 hPa. The pressure range falls within 300 - 1100 hPa and temperature within -40 - +85 °C [c].

The voltage at which it is operated is within 1.71 and 3.6 V, with a low power consumption of 2.7 µA @1Hz. It is an extremely compact 8-pin metal-lid LGA package with a footprint of only 2.0 × 2.5 mm^2 and 0.95 mm package height, alongside the low power consumption allow the implementation in battery driven devices such as mobile phones, GPS modules or watches. This in turn leads to the impressions that demanding environmental conditions could be sustained by this sensor.

The last sensor considered was the Micro Altimeter Pressure Sensor MS5607 for Absolute Pressure, is a new generation of high resolution altimeter sensors from MEAS Switzerland with SPI and I2C bus interface. This new sensor module generation is based on leading MEMS technology and consists of a piezo-resistive sensor and a sensor interface IC.. The accuracy at 25°C, 750 mbar is plus/minus 1.5 mbar. When looking at the temperature range falls within -40 to 85 °C and a pressure range of 10 to 1200 mbar. The operating voltage of the device is -0.3 to 4.0 V, and a low power consumption of Low power, 1 µA (standby < 0.15 µA). The sensor has small dimensions of only 5.0 mm x 3.0 mm and a height of only 1.0 mm allow for integration in mobile devices.

As one can see it is a very small differences that separate the common technologies out in the market. All the sensors analyzed in this case were MEMS based and had SPI/I2C output communication with the microcontroller. All had similar power consumption, all fell within the necessary specifications when it came to temperature and pressure ranges and had similar prices. In the end, however, the team decided to go with the Adafruit BMP280 I2C or SPI Barometric Pressure & Altitude Sensor. This device check listed all the boxes when it came to Precision, Pressure sensitivity, Pressure and temperature limits, Energy consumption, Operation [7] [8] environment, and size.

The other sensors had similar attributed, however it appeared as though this sensor had better documentation than the rest, and deemed most flexible when it came to incorporating with our selected microcontroller. The output units were also much easier to work with in comparison to

the other devices. Unfortunately, this part was also omitted from the project due to the effects of the COVID-19 pandemic. During testing, this part broke, but we were unable to purchase a new one before the university placed restrictions against ordering from China.

3.3.1.4 Integrated Temperature and Humidity Sensors

Humidity is the measure of moisture within the air at certain temperatures. There are three different types of humidity sensors, or hygrometers: capacitive, resistive, and thermal. Determining relative humidity is dependent on the kind of sensor and the type of dielectric that is used in them. Temperature sensing is the measure of heat within an object or substance.

This can be done my measure the change in resistance of an object, and converting this resistance into a voltage, or it can be done via the implementation of diode whose properties are directly affected by temperature. For this project, accurate measurements are important for effective roof functionality. Precision takes a back seat when it comes to the project's measurements as drastic weather events will cause large enough changes in the humidity and temperature. The different kinds of sensors and their accuracy and other characteristics are examined in the following sections.

Capacitive Humidity Sensor -Capacitive humidity sensors measure electrical capacitance when moisture makes contact with the sensor. Capacitive humidity sensors are very precise, as the moisture changes, the output voltage from the electrical capacitance changes. Capacitive humidity sensors can operate in humidity in the range of 0% to 100%, while some have more accurate readings within that range. Most capacitive humidity sensors range from about 150 pF to 200 pF for capacitance.

The first option for a capacitive sensor is the ENS210 from amsDesign. It is an integrated humidity sensor and temperature sensor, allowing for accurate interpretations of the sensor's readings. The sensor has an I2C interface for communication and recalibration, although the chip already comes pre-calibrated. The ENS210 in particular is a good option for low power.

The chip operates in standby mode, with the only the I2C receiving on, and when a measurement command is sent, the chip will boost to active and begin a measurement. The boost to active takes approximately 1 millisecond. This is helpful for reducing power consumption and managing the amount of data received by the board. More electrical and data characteristics are outlined in Table 17 below.

The second option for a capacitive sensor is Sensirion's SHTW2 integrated humidity and temperature sensor. The sensor includes an A/D converter and uses an I2C communication interface [d]. The sensor operates mainly in idle mode to conserve power, waiting for a measurement command from the microcontroller.

The third option for a capacitive humidity sensor is Sensirion's SHT-85 integrated humidity and temperature sensor. This sensor is built to last with it's protective plating that keeps dirt and dust build-up off the sensing plates, allowing for a more accurate measurement. This sensor provides a digital output with I2C c capabilities. Further specifications are outlined in the table below.

Although each measurement command triggers both the humidity and temperature measurements, part of the command can be coded to distinguish which measurement is read first. Since this chip would be used mainly for humidity measurements, this allows for quick measurement readings. A measurement reading takes a maximum of 14.4 milliseconds. For more comparison of electrical and data characteristics, reference Table 17 below.

Resistive Humidity Sensor - Resistive humidity sensors are similar to capacitive sensors, in how they function. When moisture is absorbed by the hygroscopic material within the sensor, the change in resistance is measured. Resistive humidity sensors have a smaller range in measuring humidity, only between 5% to 90% relative humidity can be measured. In lower humidity ranges, resistive humidity sensors tend to perform poorly. Lower temperatures affect the performance the sensor as well.

The following table compares both capacitive and resistive humidity sensors. The characteristics that would affect the project the most are evaluated. Focusing on their power on voltage, response time, accuracy, operating range for relative humidity, and extended performance expectancy.

Telaire's HS30P is a low cost and low power option for a resistive humidity sensor. It only requires 1 Volt of power, and the resistive output changes as moisture makes contact with the sensor. Unfortunately, extra circuit design would be needed to measure that change of resistance, but that could be as simple as another capacitor whose voltage would be measured, and any change would signify a humidity change. HS30P does have a long response time due to the extra circuitry needed.

The other option for a resistive humidity sensor is the HCZH88 from MultiComp. Similar to HS30P, additional circuitry is needed to measure the difference is resistance as the humidity changes. The response time for the resistive changes is quicker than its competition, aiding in accurate and fast measurements, which this project requires. Other characteristics for this sensor can be found in the table 17 below, comparing it to all the other options for this sensor in the project.

Table 17: Humidity Sensor Option Characteristic Comparison

Type	Part Number	Supply Voltage	Accuracy	Humidity Range	Response Time	Extended Expectancy (per year)	RoHS
Capacitive	ENS210	3.6 V	+/-4%RH	0-100%	3-5 sec	+/-0.25% RH	Yes
Capacitive	SHTW2	1.8 V	+/-3%RH	0-100%	8 sec	+/- 0.25% RH	Yes
Capacitive	SHT-85	3.3V	+/-1.5% RH	0-100%	8 sec	<0.25% RH	Yes
Resistive	HS30P	1 V	+/- 5%RH	20-95%	120 sec	+/- 1% RH	Yes
Resistive	HCZH8B	1 V	+/- 3%RH	20 - 90%	< 10 sec	+/- 2% RH	Yes

To be accurate and effective in a timely manner of changes in the weather surroundings of the project's structure, the SHTW2 is was originally selected as the humidity sensor for this project. With a better range in measuring relative humidity, and an average response time of all the possible options, it is the best choice for meeting the goals of accuracy and speed. Having power consumption in mind, the SHTW2 also has a lower required power supply voltage than it's competing capacitive sensor. The SHTW2 can also act as a backup thermometer sensor if the main thermometer sensor is damaged or loses power.

After testing and more consideration it was found that the SHTW2 did not work with our project as expected. The needed operating voltage of 1.8V is too low for our power module to produce. Since the lowest voltage regulator we have is a 3.3V linear regulator, the SHT-85 became a perfect choice. With similar specifications and better accuracy, this part fit seamlessly into our final design.

Thermistor - The chosen component for temperature sensing is a thermistor. Measuring the change in resistance of the component's material, an output voltage can be measured, and from this a temperature can be calculated. Three different sensors, all NTC (Negative Temperature Coefficient) type, are compared in the following paragraphs and from this, a part is selected. The compared components are the SMD 0603, PR103J2, and B57863S, two of which include a two-wire connection, and the other is a surface mount connection.

The first option is the Vishay SMD 0603 NTC surface mount thermistor. It comes in a protected case, which allows for it to be used for outdoor temperature measurement. It requires at least 5.6V for operation and has a resistance tolerance of 1%. It is accurate and provides a stable, almost linear output versus temperature. However, the regulating circuit to be built around the sensor requires multiple extra parts such as an operational amplifier, thus needing multiple input sources. This sensor would also require the implementation of an additional PCB to be wired to the main circuit, so that it is away from any heat emitted by the PCB. Although this part is only $1.12 USD, the number of additional components required to produce a stable and measurable output, was not deemed worth it by the group.

The second option is the LittelFuse PR103J2 bead thermistor with two copper wires and an epoxy coating on the resistive material [e]. This thermistor is small with a fast-thermal response, long life, and high stability. It has 10k ohm resistance at 25 degrees Celsius with a resistive tolerance of 0.05% and a measuring range of -55 to +80 degrees Celsius. This part requires a simple voltage divider, aka, the implementation of another 10k ohm resistor. Then a simple series of calculations within the program will output the measured ambient air temperature. At a cost of $8.12 this component is one that was highly considered for implementation.

The final comparison done was with the EPCOS (TDK) B57863S NTC bead thermistor. This one operates similarly to the PR103J2 except it has a higher resistance tolerance at 1% and a lower cost of $2.93 USD. This thermistor was placed through a series of tests by the manufacturer to the effect of storage conditions, temperature cycling, endurance, and long-term stability, all with no visible damage to the thermistor. This thermistor also requires the same circuitry needed as the PR103J2, which is a simple voltage divider with minimal code manipulation.

Table 18 shows all comparisons of the investigated temperature sensors. Below is a final decision of which part was chosen and why.

Table 18: Potential Temperature Sensor Characteristics

Type	Part Number	Power Rating	Resistance Tolerance	Measuring Range	Response Time	RoHS	Cost
Surface Mount	SMD 0603	30mW	+/-1%	-40 - +150 °C	1-8 sec	Yes	$1.12
Bead	PR103J2	30mW	+/-0.05%	-55 - +85 °C	1-10sec	Yes	$8.12
Bead	B57863S	30mW	+/-1%	-55 - +150 °C	1-15 sec	Yes	$2.93

Using the above comparisons, we ultimately decided to go with the B57863S. This decision was due to its simplicity in implementation, and low cost compared to the other sensors. Although it has the slowest response time out of all three sensors, we have decided that this characteristic is not as important as temperature tends to change gradually throughout the day.

We have also decided that the resistance tolerance value is not a major deal because we want to measure drastic changes in temperature, and a couple degrees off of the actual temperature will not cause the system to fail. Because the price of the B57863S is 4 times less than the sensor that is 2 times more accurate, the compromise here was to go with the cheaper, and slightly less accurate part.

For a better and more accurate description of the outside weather, we have decided to go with both humidity and temperature sensor. Temperature and humidity go hand in hand as the dew point outside can have an affect on the feel of the temperature. This humidity effect can cause discomfort to the activities within the building and also be indicative of something worse to come. It is known that high humidity usually leads to precipitation, so having both sensors and allowing them to communicate will provide better prediction services by our system, making for a better product.

3.3.1.5 Rain Sensors

For this section, two types of rain sensing techniques is further analyzed in order to make a decision on which is implemented into the project. The constraints and specifications that is considered are: RoHS compliant, ease of implementation into circuit and software, cost, accuracy, and size. As concluded in section 3.2.1.6, we have decided to further analyze the conductive and optical sensors, as they were deemed the most accurate out of the four types of sensors compared.

Conductive Rain Sensors - The two capacitive rain sensors that is compared in the following sections are the Vaisala DRD11A rain detector and the Vaisala YL-83 rain detector. These follow the principle of fringe fields as they emit an electric field outward onto the coating that covers the copper plating, and creates a change in capacitance once in contact with different solutions. Both have been configured to accept contact with ambient air as their normal, and only the specific capacitance changes due to water will set off signal to the output of the circuit.

The DRD11A, manufactured by Vaisala, has a capacitive plat angled at 30 degrees as to not let water moisture accumulate on its surface. This allows for better accuracy when sensing rain not just a morning dew. It also comes with a snow detection setting, and heats itself just enough to melt off any snow from accumulating on the surface.

With a detection delay of less than 0.1ms and an off delay of less than 5 minutes, it ensures that the rain has fully subsided before signal that it is no longer raining. It operates at a voltage of 12V DC +/-10% and consumes around 0.3W of power. When a low signal is sent to the output, this is the sensors way of signaling that there is rainfall. At a price of $1,1152 USD this sensor, although extremely accurate and well fit for this project, is too expensive for us to use.

The YL-83, also built by Vaisala, is much more simplified and lower power version of the DRD11A [12]. This sensor comes with the capacitive plate, and the regulating integrated circuit, which produces a digital and analog output to indicate rain and measure the strength of the rain, that can be easily read and converted to an on or off signal by our software. This sensor operates at 5V DC and sends a high signal when rain is detected.

The response time being about 1sec it is much slower than the DRD11A, but we have accepted this as a compromise as it consumes less power and only puts us back $7.81 USD from Walmart. Because it comes with its own microchip that includes and LM393 regulator, we know it is much easier to implement and will produce a stable output.

Optical Rain Sensors - Most optical sensors that aren't configured specifically for cars, are configured for motion sensing. However, these motion sensors can still be used to detect rain with a little tweaking in the code. We have decided to look into the PIR-02 sensor by OESPP, and the GRS390 built by Equalizer. The same standards apply to these sensors as they did to the capacitive sensors. We is looking at price, ease of implementation, response time, and power consumption.

The PIR-02 is an optical motion sensor that, we believe, can be utilized for rain sensing if configured properly within the code. This sensor has a response time of 2.5 – 250 sec depending on the amount of resistance applied to the sensor.

Using an input of 5V, this sensor consumes low power, which is ideal, and it connects easily to the PCB by only three wires. However, we would have to do a very in depth test to see how this sensor responds to rain, and ensure that it only goes off for rain fall. Because it is primarily a motion sensor, a bird or debris flying through the sensors line of vision could potentially cause an unwanted response, which could prove to be difficult when implementing this sensor into our project.

The GRS390 is manufactured to work for various vehicles, but it is configured to be a rain sensor. This allows for minimal code manipulation, however the wiring to our main PCB could potentially prove to be difficult. Because this sensor is made for vehicles, it requires a much larger power supply of 24V AC. On the other hand, it responds fairly quickly to rain detection, making it a good option.

Another downfall of this sensor, is that there not many available resources on how this could implement into our system. The handbook and other resources for this component only show the implementation of this sensor into a vehicle. This makes the sensor much more difficult to implement because we will have nowhere to go if we cannot get it to operate correctly. At $55.96 this price point is a little too high to make any struggle we encounter worth it.

Table 19: Potential Rain Sensor Characteristics

Type	Part Number	Supply Voltage	Response Time	RoHS	Cost
Capacitive	DRD11A	12V DC	< 0.1ms	Yes	$1,152.00
Capacitive	YL-83	5V DC	1 sec	Yes	$7.81
Optical	PIR-02	5V DC	2.5-250 sec	Yes	$8.95
Optical	GRS390	24V AC	< 120 sec	Yes	$55.96

Looking at the above chart it is easy to see the clear winner. With the lowest cost, and even one of the shorter response time, the YL-83 is the obvious choice for this project. It is easy to implement, as it has a multitude of resources available, and it is already configured to sense the presence of rain without doing any signal manipulations in the code. The other sensors, although great for their specific applications, are either too expensive, too difficult to implement, or both, thus leading us to our final choice of the YL-83 rain sensor.

3.3.1.6 Light Sensors

As concluded in section 3.2, our choice of light sensing technology is the simple and environmentally friendly photodiode. The two photodiodes that we is using are visible light sensitive photodiodes, with operating frequencies of around 400 – 700 nm, and ultraviolet (UV) light sensitive photodiodes, with operating frequencies of around 100 – 400 nm.

The visible light sensor is used to measure the amount of light that is flooding into the enclosure that is topped by the roof. The ultraviolet light sensor is used to measure the amount of harmful-to-humans UV light that is flooding into the enclosure. The main goal of the UV light sensor is to avoid excessive exposure of the occupants in the enclosure to skin-damaging UV light.

Visible Light Sensors - The visible light sensors we is selecting is based on their ease of soldering and their level of efficiency. Based on these criteria, our options include the ODD-5W photodiode, ALS-PT19 photodiode, and OPT101 integrated photodiode with transimpedance amplifier.

The first option, the ODD-5W photodiode is a through-hole diode in a TO-5 package. With its beefy package, it is capable of handling high reverse-bias voltage of at most 60 V and accurately responds to changes in light with its changing current within 10 ns. The diode has a spectral range of 300 – 1000 nm, well within the visible light spectrum. The light current will vary from a minimum of 1 nA up to 100 µA. The main downside to this option is its hefty price of $15.89 per part.

The second option is the ALS-PT19, which is a surface-mount photodiode with a much smaller package. Since it is much smaller, it is only capable of handling a maximum supply voltage of 6 V. The diode has a smaller spectral range of 400 – 700 nm, which is still within the visible light spectrum that we are concerned about. The light current will vary from a minimum of 5 µA up to 520 µA. This option has a very reasonable price of just $0.43 per part.

The last option is the integrated photodiode and transimpedance amplifier that is labeled as OPT101. This one is an 8-pin IC with a wide supply voltage range of between 2.7 – 36 V, internal feedback, and internal transimpedance amplifier for easy integration into our sensor system [f]. The photodiode on the chip has a spectral range of 400 – 1000 nm, which is within the visible light spectrum.

The last option is the most impressive option, since we is able to integrate this IC into our design without having to worry about leakage current errors, noise, and gain peaking when designing our own transimpedance amplifier. In addition to these positives, the IC package means that our design is compact and costs just $8.80 per part. The comparison between all the visible light sensor parts is summarized in *Table 20* below.

Table 20: Comparison of Visible Light Sensing Parts

Part Name	Components	Price ($USD)	Spectral Range	Maximum Voltage
ODD-5W	Photodiode	$15.89	300 – 1000 nm	60 V
ALS-PT19	Photodiode	$0.43	400 – 700 nm	6 V
OPT101	Integrated Photodiode and Transimpedance Amplifier	$8.80	400 – 1000 nm	36 V

The clear winner in this case is the OPT101. With its compromise on both price point and size in comparison between all three competitors, the integrated photodiode and transimpedance amplifier simplifies the overall design of the sensor module and allows us to save board space and provide an accurate reading of illumination.

UV Light Sensors - The UV light sensors, much like the visible ones before, is selected based on their ease of soldering and their level of efficiency. Based on these criteria, our options include the SD040 analog UV photodiode and the VEML6070 UVA light sensor with I2C interface.

The first option is an analog-output photodiode with a spectral range of 350 - 1050 nm. In a 1206 SMT package, this photodiode is easy to solder while also having a very small board footprint. It is also an analog-output sensor, allowing easy integration into a transimpedance amplifier that can then be connected to the microcontroller’s on-board ADC ports. On the other hand, this photodiode’s spectral responsivity range is far too wide to be a dedicated UV light sensor.

There arises an issue that, although visible light and no UV is entering the enclosure, this sensor might provide a false reading that shows that there is too much UV in the enclosure. This can be circumvented by designing a low-pass filter that will filter out the lower frequencies of visible light photons.

The second option is a digital-output photodiode with an I2C interface. The VEML6070 comes with an SMT package that contains an integrated photodiode, low pass filter, and I2C interface that allows for digital communication between the chip and the microcontroller [g]. This allows us to free up ports on the ADC that can be used for extra sensors later on in the project design phase.

The comparison between all the UV light sensor parts is summarized in *Table 21* below.

Table 21: Comparison of UV Light Sensing Parts

Part Name	Extra Circuitry Required	Price ($USD)	Spectral Range	Maximum Voltage
SD040	Low Pass Filter, Transimpedance Amplifier	$1.91	350 – 1050 nm	50 V
VEML6070	I2C Support Circuitry	$2.77	280 – 400 nm	5.5 V

From the comparison, we are choosing to go with the VEML6070. While it has a much lower maximum supply voltage and a higher price, the integrated UV sensor with low pass filter, temperature sensor, and I2C interface makes it a much more integrable candidate for the UV light sensor. With the VEML6070, we have added I2C support circuitry to ensure it can adequately interact with the microcontroller.

3.3.2 Power Module Parts

To power the whole unit: sensors, motors, and communication modules included, solar panels and a lithium battery with be utilized together. In order to meet the hardware's power consumption and also maintain optimized energy efficiency, the hardware for the power modules is analyzed and compared to find the best equipment for the project.

Included in the power module is a solar panel subsystem, a battery management system, a lithium ion battery pack, a reverse polarity protection interface, and separate voltage regulators for the +12V, +5V, and +3.3V power lines.

3.3.2.1 Solar Panels

By comparing several products that have abundant availability on Digikey.com, we have decided on which solar panel to use to power the autonomous roof system.

The Kitronik Polycrystalline Solar Cell 1W, 5.0 V is a polycrystalline solar cell capable of supplying an open-circuit voltage of 5.0 V and a maximum current of 200 mA. The cell has dimensions of 110 mm x 110 mm x 3 mm. It has an efficiency of ~13%.

The IXYS Monocrystalline Solar Cell 1W, 6.9 V is a monocrystalline solar cell capable of supplying an open-circuit voltage of 6.91 V and a maximum current of 205 mA. The cell has dimensions of 90 mm x 65 mm x1.8 mm. It has an efficiency of ~25%.

The Speed Technology Monocrystalline Solar Cell 1W, 8.2 V is a monocrystalline solar cell capable of supplying an open-circuit voltage of 8.2 V and a maximum current of 170 mA. The cell has dimensions of 100 mm x 75 mm x 1.5 mm. It weighs just 33 g and has an efficiency of 15.5%.

To diverge from the low-power solar cells available on Digikey, the ECO-WORTHY solar panel is a polycrystalline solar panel module capable of supplying up to 12 V and 5 W of power [h]. This will allow the solar panel to provide enough energy to be used in case the lithium battery is not usable at some particular moment. This solar panel has an aluminum frame for durability and has dimensions of 255 mm x 194 mm x 17 mm. This solar panel is purchasable from Amazon.com, which is conducive to manufacturability by being readily purchasable.

Solar Panel Selection Conclusion - While weighing all relevant factors, *Table 22* was constructed that organizes everything in an easy-to-reference format to accurately consider all solutions.

Table 22: Solar Panel Comparisons

Solar Panel Company	Supply Voltage (V)	Max Supply Power (W)	Dimensions (mm^3)	Unit Price ($USD)
Kitronik	5.0	1	110 x 110 x 3	$22.45
IXYS	6.9	1	90 x 65 x 1.8	$21.77
Speed Technology	8.1	1	100 x 75 x 1.5	$12.74
ECO-WORTHY	12.0	5	255 x 194 x 17	$13.45

From the table we constructed comparing the solar panel options, it was easy to deduce that the ECO-WORTHY 12 V 5 W solar panel system would be the best candidate to integrate into our autonomous smart roof system. With an impressive maximum power output of 5 W and a high open circuit voltage of 12 V, the one solar panel module would remove the necessity of having a large array of solar cells in order to provide enough power to microcontrollers, sensors, and output motors.

Not only is the single solar panel capable of delivering an exceptional amount of power, it is also a lower-cost solution compared to the amount of money that would be required to create the array of lower-powered solutions.

3.3.2.2 Lithium Ion Batteries

The lithium ion batteries we is looking to add to our project is housed in an 18650 package in order to easily integrate the battery pack into an off-the-shelf battery enclosure. The parts that we will have selected is based primarily on nominal capacity and integration ability. In order to provide support, the batteries is held in plastic holders.

Batteries - The lithium ion batteries we is comparing are ones found on Digikey.com and Amazon.com. The batteries that we have chosen as candidates are the PRT-13189 from SparkFun Electronics and the 1781 from Adafruit Industries, LLC. The relevant data pulled from their datasheets are summarized in *Table 23* for comparison.

Table 23: Lithium Ion Battery Package Comparison

Li-ion Battery Part	Nominal Capacity	Unit Price ($USD)
PRT-13189	2.6 Ah	$6.50
Adafruit-1781	2.2 Ah	$9.95

From the comparison conducted, it is obvious to choose the PRT-13189 [i]. This lithium ion battery package comes an impressive nominal capacity of 200 mAh more than the competition while being priced at only $6.50 per unit. It also comes with a soldering tab so that we can avoid soldering directly to the battery, which can often damage the battery if careful considerations are not taken.

Plastic Holder - The plastic holder that is used to separate and contain the lithium ion batteries. This will help with securing the batteries close together and on the enclosure. The two options we have include the Ltvystore battery case holder from Amazon.com and the EV plastic battery spacer from Ebay.com.

The option we will go with is the EV plastic battery spacer from Ebay.com due to easy of ordering and shipping. With its origin of shipping from within the United States, and its use specifically for 18650 cell packages, the EV plastic battery spacer is the best candidate for the plastic battery holder.

3.3.2.3 Battery Management System

The battery management system is responsible for managing the battery charging and discharging cycle. Since the safety of both the entire autonomous roof system and the users handling this system are of utmost importance, the battery management system chosen will have to be very reliable and capable of operating for longer than the lifetime of the lithium ion battery.

The battery management system will need to be very accurate, with constant-voltage, constant-current charge cycles that maintain a 0.5 – 1 C charging rate to our lithium ion batteries to preserve their health. This means that, at a constant charge voltage of 4.1 – 4.2 V, the battery management IC will need to facilitate at most 1.3 A to the lithium battery. In addition, beneficial features to

have in the BMS IC would include temperature monitoring and emergency safety termination that will stop all charging until it is reset.

Of course, these features will also have to be balanced with cost and integration ability. With these conditions and constraints in mind, our options for battery management ICs include the MCP73842, LT3650-4.2, and the BQ2057. A pre-assembled battery management system is also considered from Amazon.com. The relevant information is assembled into *Table 24* for accurate comparison.

Table 24: Battery Management System Integrated Circuits (BMS IC) Comparison

BMS IC	Manufacturer	Unit Price ($USD)	Input Voltage Range	Fault Protection	External Components
MCP73842	Microchip Technology	$1.29	4.5 – 12 V	Overcurrent, Overvoltage, Overtemperature	6 passive, 2 active
LT3650-4.2	Linear Technology	$7.14	7.5 – 32 V	Overcurrent, Overvoltage, Overtemperature	5 passive, 2 active
BQ2057	Texas Instruments	$2.11	4.5 – 15 V	Overcurrent, Overvoltage, Overtemperature	8 passive, 2 active
HX-3S-001	DIY MORE	$3.00	12 V	Overcurrent, Overvoltage, Overtemperature	None

Since the differences between the options were relatively minor, it was concluded that the MCP73842 would be the better candidate to begin prototyping with. The fact that it is not pre-built but has a very low unit price of just $1.29 and needing just 8 extra components means that the cost to manufacture remain low [j].

In addition to its low price, it has the necessary fault protections to adequately maintain the lithium battery pack throughout its lifetime. It is easy to integrate into our battery management system with the help of the reference and application designs provided in the datasheet.

3.3.2.4 Voltage Regulators

The voltage regulators is responsible for taking the unregulated input voltage from the lithium ion batteries and holding them at 12, 5 and 3.3V power lines.

12V Regulator: Buck/Boost IC - The 12V regulators we is focusing on is buck/boost converters due to the varying input of the lithium ion battery pack. With a 3-series lithium ion battery pack,

the expected input voltage to the system will vary between 11.1 – 12.6 V. This will also allow flexibility in case we wish to change our set-up for the lithium ion battery pack.
In addition to this, the 12 V regulator, being the main interface between the lithium battery pack and the entire autonomous roof system, it will need to handle the most power flowing through the chip. Lastly, the regulator will need to have an efficiency of 80% or greater with a reasonably low number of external components needed to maintain such efficiency.

For this reason, Texas Instruments' WEBENCH® power designer tool is used. From this tool, and for our specific application, we are recommended the LM25118 Wide Voltage Range Buck-Boost Controller [k]. In addition to the designs recommended by the TI WEBENCH® tool, we chose to compare the LT1370 and MC34063 switching converter circuits from Digikey.com. The relevant information is pulled from the datasheets and assembled into *Table 25* for accurate comparison.

Table 25: 12V Regulator Comparison

Part Number	Manufacturer	Unit Price ($USD)	Input Voltage Range	Maximum Power Handling	Topology
LM25118	Texas Instruments	$6.88	3.0 – 42 V	2 W	Buck/Boost Controller
LT1370	Linear Technology	$14.89	2.7 – 30 V	2 W	Buck/Boost Converter
MC34063	STMicroelectronics	$0.58	3.0 – 40 V	0.625 W	Buck/Boost Controller

From the comparison, we are deciding to go with the LM25118 Buck-Boost Controller. Despite its simpler topology which requires more external components, the LM25118 presents the best performance buck-boost system on a chip which has an appropriate amount of power handling with a wide input voltage for maximum flexibility with our battery system.

12V Regulator: Buck/Boost MOSFET - The LM25118 requires additional transistors to drive the buck-boost controller's output. These transistors are known as high-side buck MOSFET and low-side boost MOSFET [j]. In the following sections, we is comparing different parts that can handle the amount of power that is expected in the buck/boost cycles of the buck/boost controller. For simplicity, we is utilizing the design in the typical application circuit published in the datasheet since its application similarly matches our own application.

From the WEBENCH® power designer tool, the CSD16340Q3 N-channel power MOSFET is recommended for the high-side buck MOSFET and the BSC046N02KS-G N-channel power MOSFET is recommended for the low-side boost MOSFET. The LM25118 lists the Si7148 N-channel power MOSFET in its typical application circuit published in its datasheet.

Ultimately, we chose to use the LM25118. However, due to the PCB coming in dysfunctional and the COVID-19 crisis, we were forced to use the LM2596 to regulate our battery voltage to +12V.

5V Regulators - The 5V regulator in our project is responsible for receiving the 12V power line produced by the 12V regulator and stepping it down to 5V. Due to its only responsibility being stepping down voltage, it is appropriate to use a step-down buck converter in order to achieve this. Similar to the other regulators, our main priority in choosing the 5V regulator is high efficiency with reasonable costs. Because of the very small variation in input voltage, the minimum number of external components should be needed in order to step down the voltage.

Similar to the other regulators, we will use both the TI WEBENCH® power designer tool and the Digikey.com search function in order to find the appropriate systems to integrate into our design. From the WEBENCH® design tool, it is recommended to use either the TPS563231 or the TPS561208 3A or 1A, respectively, step-down converter. From Digikey.com, we have chosen to field the LM2596 step-down regulator as a candidate. The relevant information is pulled from the datasheets and assembled into *Table 26* for accurate comparison

Table 26: 5V Regulator Comparisons

Part Number	Manufacturer	Unit Price ($USD)	Maximum Current Handling
TPS563231	Texas Instruments	$0.85	3 A
TPS561208	Texas Instruments	$0.85	1 A
LM2596	Texas Instruments	$5.72	3 A

While the TPS56X series voltage regulators provide adequate current handling capability with an incredible small price, the LM2596 would be a much more appropriate chip for our application. The TPS56X series voltage regulators are only available in the very tiny SOT-563 package, while the LM2596 provides variety in the packages available to use in our design [l]. For this reason, despite the much higher unit price for similar features, the LM2596 will serve as the 5V regulator in this project.

3.3V Regulators - The 3.3V regulator in our project is responsible for receiving the 5V power line produced by the 5V regulator and stepping it down to 3.3V. Due to its only responsibility being stepping down voltage, it is appropriate to use a step-down buck converter similar to the 5V regulator in order to achieve this. Similar to the other regulators, our main priority in choosing the 3.3V regulator is high efficiency with reasonable costs. Because of the very small variation in input voltage, the minimum number of external components should be needed in order to step down the voltage.

In addition to the low variation in voltage input and output, the 3.3V regulator is not expected to handle much power being dissipated across its package. For this reason, we is comparing devices that may use either the step-down regulator technologies or linear regulator technologies. Similar to the other regulators, we will use both the TI WEBENCH® power designer tool and the Digikey.com search function in order to find the appropriate systems to integrate into our design.

The parts we is comparing are the LT1763 and LM2596. The comparison is compiled in *Table 27*.

Table 27: Comparison of 3.3V Regulators

Part Name	Manufacturer	Regulator Type	Efficiency	Unit Price ($USD)
LT1763	Linear Technology	Linear Dropout	82.5%	$4.72
LM2596	Texas Instruments	Step-Down Switching	75.0%	$5.72

From the comparison above, it is clear that the LT1763 from Linear Technology provides superior performance for a much better price point [m]. Therefore, we chose to use the LT1763 Low-Noise LDO Regulator for our 3.3V regulation. Due to the COVID-19 epidemic, we chose to utilize the 3.3V regulator on the development board.

3.3.2.5 Reverse Polarity Protection P-MOSFETs

The reverse polarity protection P-channel MOSFETs is responsible for the reverse polarity protection circuitry that must protect the rest of the power module from an incorrect input of the lithium ion battery pack.

Texas Instruments' whitepapers on reverse current protection recommend the IRLML2502 or the Si2312 MOSFETs for power-path P-channel MOSFET circuitry. The comparison is compiled in *Table 28* below.

Table 28: Comparison of P-MOSFETs

Part Name	Manufacturer	Rds(on) @ Vgs = 20V	Unit Price ($USD)
IRLML2502	International Rectifier	0.045 Ω	$0.50
Si2312	Vishay	0.031 Ω	$0.42

From the comparison above, it is clear that the Si2312 provides a much better Rds(on) for our P-channel MOSFET acting as reverse polarity. With this low Rds(on) at a cheaper price, less power is dissipated across the MOSFET when the lithium ion battery pack is providing power to the autonomous roof system.

3.3.3 Microcontrollers

The microcontroller will take on multiple roles in this design. The microcontroller is required to, with the help of peripheral analog-to-digital converters (ADC), process the weather-related data from the sensors. From the data, the microcontroller will output a PWM signal that will drive an external motor and an external fan. The sensors that is used include barometric pressure, temperature, humidity, light, wind speed, and pressure plate, meaning there is a total of 6 sensors whose data needs to be processed. Due to the number of sensors that is interacting with the microcontroller, the microcontroller that is used needs to have enough GPIO ports.

In addition, this microcontroller will interact with an external mobile phone application. The microcontroller will send information regarding the sensor-gathered data and statuses of the motor and fan drivers to the mobile phone application. In order to enable this feature, a Bluetooth and Wi-Fi module will need to be interfaceable with the microcontroller.

Lastly, the microcontroller should have enough online application resources to best implement the design. This means that the microcontroller should be capable of being programmed using a programming language that is familiar with the software development team. Similarly, the microcontroller will need to be powerful enough while keeping costs reasonably low.

ATmega1280 - The first microcontroller under consideration is the Atmel ATmega1281. In summary, this microcontroller is a high-performance RISC-based microcontroller with 128 Kbytes of flash program memory, 54x GPIO lines, and a plethora of supported peripherals. This microcontroller can support UART, SPI, and I2C communication. With a CPU speed of 16 MHz, data processing is exceptionally fast, while keeping operating voltage at a low voltage range of 1.8 – 5.5 V [n].

With the UART, SPI, and I2C communication capabilities, this microcontroller can interface with digital-output sensor systems that we may need to purchase. The operating voltage range of 1.8 – 5.5 V allows us to keep the amount of voltage regulators that we will need to power all components of the system to a minimum.

The largest benefits of the ATmega1281 is its 8-channel ADC and large number of GPIO pins. This ADC can read data from 8 different external sensors with 10-bit resolution. The GPIO pins can enable many different features in case more can be added. The ATmega1281 can be programmed in the Atmel Studio 7 development environment. This allows code to be written in C/C++ or assembly, in which the development team is proficient.

ATmega328 - The second microcontroller under consideration is the Atmel ATmega328. This microcontroller is a lower-power version of the ATmega1281 with less features. Like its beefier cousin, the ATmega328 supports UART, SPI, and I2C digital communication, has a low operating voltage range, and can support many peripherals. With UART, SPI, and I2C communication capabilities, this microcontroller can interface with digital-output sensor systems that we may need to purchase. However, it has only 1 UART port, greatly reducing the number of components that require UART to communicate with a microcontroller. The operating voltage range of 1.8 – 5.5 V allows us to keep the amount of voltage regulators that we will need to power all components of the system to a minimum.

The main differences between the chips are that the ATmega328 has 23x GPIO pins, has a 6-channel, 10-bit resolution ADC, and only 32 Kbytes of flash program memory. The ATmega328 can be programmed in the Atmel Studio 7 development environment. This allows code to be written in C/C++ or assembly, in which the development team is proficient.

MSP430FR6989 - The third microcontroller under consideration is the Texas Instruments MSP430FR69x series. This microcontroller is an ultra-low-power microcontroller with 128

Kbytes of FRAM program memory, 63x GPIO pins, and has many opportunities for adding peripherals. This microcontroller can support I2C, SPI, and UART communication protocols. It operates at a supply voltage range of 1.8 to 3.6 V.

With the UART, SPI, and I2C communication capabilities, this microcontroller can interface with digital-output sensor systems that we may need to purchase. The operating voltage range of 1.8 – 3.6 V, however, severely limits the voltage range of our voltage regulator network. This may mean that we would need to have many different voltage levels to power the sensors, motors, and microcontroller.

Compared to the ATmega microcontrollers above, this microcontroller has a 12-channel 12-bit resolution ADC, allowing for more analog sensors to be added and with higher-precision data to be collected. This microcontroller can be programmed using TI's Code Composer Studio, allowing the program code to be written in C. In addition to this, the entire senior design group has experience in programming on the MSP430FR6989 from the Embedded Systems course at the University of Central Florida. This means that we are all capable of contributing to the coding of the microcontroller. The main benefits to this microcontroller are the expanded analog-to-digital converter capabilities and the entire senior design groups' proficiency in programming this microcontroller.

Microcontroller Selection Conclusion - While weighing all relevant factors, Table 29 was constructed that organizes everything in an easy-to-reference format to accurately consider all solutions.

Table 29: Microcontroller Comparisons

Microcontroller	Unit Price ($USD)	Program Memory	Operating Voltage Range	Digital Communication	ADC Capability
ATmega1280	$10.63	128 KB	1.8 – 5.5 V	• 4x UART • 1x I2C • 4x SPI	16x 10-bit
ATmega328	$1.42	32 KB	1.8 – 5.5 V	• 1x UART • 1x I2C • 2x SPI	8x 10-bit
MSP430FR6989	$8.40	128 KB	1.8 – 3.6 V	• 4x UART • 1x I2C • 2x SPI	12x 12-bit

From the table, we were able to quickly deduce that, while the ATmega1280 is much more expensive than the other options, it provides the best capabilities for our stated goal. The 128 KB of program memory allow sophisticated algorithms, with as much information as possible, can be programmed and executed smoothly.

The wide operating voltage range of 1.8 – 5.5 V allows us to keep the number of voltage regulator stages to a minimum, which will contribute in keeping both costs low and board size small.

As an external-sensor-oriented project such as ours, the 16-channel analog-to-digital converter that the ATmega1280 possesses, in addition to the plethora of digital communication support and abundant program memory, provides the flexibility to add as many features to our project as one might need.

Due to the COVID-19 pandemic, we were unable to finalize our microcontroller PCB design, and order it on time. Because most of the parts used for this project are from China, the requirements for this project changed and we were able to compromise by using development boards and breadboards.

To make up for the lack of microcontroller PCB, we decided to go with the BlunoMega 1280, which is a development board with an ATMega 1280 chip on it. All specifications are the same as the ones stated for the ATMega chip with the exception of a higher operating voltage range, which has proven to add some complications to the programming of this board. This board also has Bluetooth capabilities with a fair amount of documentation to assist in the setup of this board and its connections. However the use of the development board limits the number of Inter-Integrated communication (I2C) devices the team can use since the board is only equipped with two I2C ports.

3.3.4 Wireless Communication Modules

To be able to connect to both the web and phone application, the unit was proposed to be equipped with both Wi-fi and Bluetooth capabilities. Since the unit may be placed outside or on top of the structure, Bluetooth and potentially Wi-Fi would need to be implemented to ensure proper communication between the hardware and software of the system.

The microcontroller will also need to connect to the database to store sensor data, and to receive predictive data that will drive the trigger for the roof reactions. The Bluetooth will need to handle both client to client connections and client to access point connections, since Bluetooth is used to communicate between the board and the database along with the board and the user's phone. Having at least Bluetooth with preferred Wi-Fi communications will mitigate the loss of functionality of the roof, if one of the connections is lost due to power or unforeseen errors.

To try and mitigate both cost and power consumption, a module that has both Wi-fi and Bluetooth would be ideal for the project. One downside of having one module, is that if any damage, or power loss occurs, both communicative protocols is lost. But if power is to be lost to the Wi-fi and Bluetooth modules, it is expected that total power loss is to be expected. The benefit of lower consumption outweighs that risk as well.

Bluetooth Classes – Before deciding which Bluetooth module we wanted to implement, we had to find a Bluetooth class which worked with our project scope. There are three Bluetooth classes

all of which have different transmission powers and ranges. The summary and comparison of these classes can be seen below in *Table 30.*

Table 30: Bluetooth Class Comparison

Class	Transmisson Power	Intended Range
Class 3	1 mW	<10 meters
Class 2	2.5 mW	10 meters (33 ft)
Class 1	100 mW	100 meters (328 ft)

As seen, Class 1 is the most powerful class with the longest transmission range. Because of this, we have decided to only investigate Bluetooth Class 1 devices for this project, as a 328 ft range is more than enough for us. If this were implemented into a full-scale building, multiple Bluetooth modules could be placed around the building as well as the implementation of Wi-Fi.

CYBT-423028 - The CYBT - 423028 is a dual-mode Bluetooth and low-energy wireless module. The module's possible communication modules is compatible with all other hardware for this project. It is capable of UART, I2C, and SPI, and supports both ADC and PWM functions [o].

There is also a peripheral UART to allow for interfacing any extra peripheral hardware with GPIO. With the unit being exposed to different weather conditions, the module can handle temperature ranges from –30 °C to +85 °C. The module has 1024 KB flash memory and 512 KB SRAM to combat any latency in communication, lowering the risk of data loss due to loss of connection or power.

The module complies with Bluetooth's current Core Specification, version 5, and it operates on the ISM 2.4GHz The minimum voltage for power on of the chip is 1.76 Volts. The CYBT-423028 supports multiple hardware power modes, such as idle, sleep, and Timed-Wake. For transmitting data, the module requires 5.6 mA, and receiving data requires 5.9 mA. The module has a PCB antenna and has a dynamic communication range.

nRF24L01 - The nRF24L01 is probably the cheapest and most common radio transceiver. It transmits on the Industrial, Scientific, and Medical (ISM) band of 2.4 GHz. The nRF24L01 can be interfaced using Serial Peripheral Interface (SPI) to configure different frequency channels, data rate, and output power for the communications. There is a built-in power saving mode to reduce power consumption, but the module still only requires a minimum supply voltage of 1.9 Volts.

The nRF24L01 also comes equipped with Enhanced ShockBurt, which eliminates the need of any extra hardware or interfaces since the chip can handle its own protocol for receiving packet acknowledgements and resending packets. The chip functions between -40 °C to 85 °C. For transmitting data, the chip requires 11.3 mA, and receiving data requires 12.3 mA. The nRF24L01 has a transmission range of 800 meters from a PCB antenna, in open air, and an external antenna (IPEX) can be used to increase the range.

ESP32 - The ESP32 is another dual module with both Wi-Fi and Bluetooth. The Bluetooth category is capable of both Bluetooth low-energy and Bluetooth version 4.2. The Wi-Fi category

supports multiple 802.1 protocols. The module can be interfaced with SPI, UART, I2C, and I2S, which extends to communicating to peripherals via the same protocols. The module is equipped with 4 MB of Flash data storage but does not include an SRAM. The ESP32 module family can have either a PCB antenna, or an IPEX antenna for extended communication range. The chip functions between -40 °C to 85 °C. The minimum power supply voltage is 3.0V.

CC2540 - The CC2540 is a Bluetooth only chip with Bluetooth Low Energy 4.0 capabilities. The CC2540 uses an RF transceiver with its own 8051 MCU. It also has in-system programmable flash memory, 8-KB RAM, and many other supporting features and peripherals. Low-power sleep modes are available as well as short transition times between operating modes. This allows this product to consume very low amounts of power suitable for our project.

The CC2540 requires minimal external components, and comes with reference designs from if manufacturer, Texas Instruments (TI), which allows us to have the documentation required to implement correctly. The chip also has five different modes which can be implemented to keep energy consumption low. These include: Active Mode RX and TX and Power Modes 1, 2, and 3. With a wide voltage range of 2-3.6V this allows for full RAM and register access in all power modes activated. The microcontroller available on the chip has 8KB SRAM and 128-256KB of in-system programmable flash. The peripherals include a 12-Bit ADC with eight channel, one 16-Bit and two 8-Bit general-purpose timers, twenty-one general-purpose I/O pins, USB interface, and a battery monitor and temperature sensor.

Software wise, this part comes with a Bluetooth v4.0 compliant protocol stack for single-mode BLE. This stack comes with the controller and host including: GAP used as the central peripheral, observer, and broadcaster, ATT/GATT used as the client and server, and SMP for encryption and decryption. TI also provides software documentation including sample applications for GAP central peripheral roles, and multiple configuration options.

Wireless Communication Module Conclusion - The design team decided to choose the CYBT-423028 module to meet the wireless communication needs for this project. The chip is equipped with both Bluetooth and Wi-Fi capabilities, and most impressively has a dynamic range of frequencies to combat congestion on the ISM bandwidth. It’s electrical characteristics, plus its multi-mode options fit the needs of the project the best in remaining effective in its objectives but low power and low cost.

Along with the CYBT-423028 the team also decided to have the Texas Instruments CC2540 Bluetooth v4.0 chip as a backup. With its impressive functions and capabilities as well as low power consumption modes, it is a good contender for this project. The main issue with this chip is its lack of Wi-Fi capabilities, which is a preferred capability by the team.

As mentioned in the above research, we wanted to include both Wi-Fi and Bluetooth communication to the system. However, as the project design continued, we found it difficult to implement both even with the integrated chips. From this we ultimately decided to go with the CC2540 Bluetooth chip. Because of COVID-19, we had to implement a new approach when it came to our microcontroller design by using a development board instead. In the final design, we did not need to use or wire a separate Bluetooth chip to integrate into the system. Instead we utilized the one that was included on the Bluno Mega 1280, which just so happened to be a TI

CC2540. This saved us time as we were already prepared to use this chip, and we did not have to create or test any schematics for a stand-alone chip.

3.4 Parts Selection Summary

In this section, we is summarizing our parts selection with a table that will outline all major components that is included in the initial designs. *Tables 31 - 33* below contains this outline. Table 31 is dedicated to the sensor parts. These parts are just the major ones we are placing into our design; miscellaneous, extraneous, or passive components is excluded for simplicity. In the following tables, all parts that are crossed out have been removed from our project due to circumstances out of our control.

Table 31 is dedicated to the final sensor part decisions. This table only shows parts that are the major required sensors that we are placing into our design. Any other miscellaneous, extraneous, or passive components were excluded for simplicity.

Table 31: Sensor Parts Summary

Part Function	Part Name
~~Proximity Sensor~~	~~MP102103~~
~~Anemometer~~	~~Adafruit 1733~~
~~Barometric Pressure Sensor~~	~~BMP280~~
Humidity Sensor	SHT-85
Temperature Sensor	B57863S
Rain Sensor	YL-83
Ambient Light Sensor	OPT101
UV Light Sensor	VEML6070

The following table, Table 32, is dedicated to the final power parts selections. Similar to the sensor list, these parts are just the major ones we are placing into our design; miscellaneous, extraneous, or passive components is excluded for simplicity.

Table 32: Power Parts Summary

Part Function	Part Name
Solar Panel	ECO-WORTHY 12V 5W
Lithium Ion Battery Pack	PRT-13189
~~Plastic Battery Spacer~~	~~EV~~
Battery Management IC	MCP73842
~~Buck/Boost Controller (12V)~~	~~LM25118~~
Buck Converter (+12V)	LM2596
~~Low Dropout Regulator (3.3V)~~	~~LT1763~~

Table 33 summarizes the final parts decisions for our microcontroller and integrated Wi-Fi/Bluetooth module. As mentioned in previous sections, we had to omit the possibility of buying stand-alone chips for this module, as well as removing Wi-Fi capabilities from our project. All other miscellaneous, extraneous, or passive components is excluded for simplicity.

Table 33: Microcontroller Parts Summary

Part Function	Part Name
Development Board	Bluno Mega 1280
Bluetooth Module (Included on Development Board)	CC2540

3.5 Software Development Research

The implementation of the project's software development is discussed in the following section. While the capabilities of the hardware on the microcontroller plays a big role in the selection, the development environment and testing for the software on the microcontroller also needs to be taken into account. Whether the software development will cost more time and resources, including learning curves and extra development kits that are necessary for debugging. Both the ATmega and MSP430 microcontroller development environments are analyzed, as those are the two main boards under consideration. The different environment options the ATmega is supported by, along with the possible emulators and debuggers that can be used during the programming and

testing stages of the project are discussed. The MSP430 environment and development tools are also outlined and evaluated.

3.5.1 Microcontroller Firmware

For developing software on the ATmega 1280, there are two main processes that can be implemented: using ATmega supported Integrated Development Environments accompanied by an in-circuit debugger or to use an Arduino as a bootloader. The ATmega 1280 has two supported Integrated Development Environments (IDE): Atmel Studio and MPLAB. Both IDEs require additional debugger hardware or development kits in order to interface between the microcontroller and the computer running the IDE. The possible options for debuggers is discussed in the next section.

Atmel Studio 7 - The first option for IDEs for the software development of the project is Atmel Studio 7. Atmel Studio 7 is an integrated development platform (IDP) for all AVR microcontroller applications written in C/C++ programming languages. It supports all debuggers and development kits for AVR devices. Some the key features of the IDE that are applicable to this project are the code libraries help in Atmel Gallery, the online development plug-ins that support Arduino, and the debugging interface that incorporates real-time value tracking.

The capabilities for interrupt monitoring and tracing would be beneficial for the development of the A.I.R.E. project because of the number of sensors and interrupts that are expected in implementation. With supported Arduino plug-ins available, programming the majority of the project's sensors, that are Arduino products, is more feasible. The largest concern for implementing the Atmel Studio 7 IDE, is the complete lack of a macOS X version. With the decision to make the phone application for the project an IOS application, the need to have cross platform development and testing for the microcontroller and phone application decreases effectiveness and will cost more time and resources for the developers.

MPLAB X IDE - The other option for IDEs that is directly supported by the ATmega is MPLAB. Specifically, the MPLAB X IDE. Similar to the Atmel Studio 7, MPLAB supports C/C++ development for most of Microchips' controllers (including the 1280) and also includes a Java platform. MPLAB X functionalities can be expanded using other software libraries from the MPLAB family of plug-ins The IDE also includes a specific open-source compiler for AVR microcontrollers.

A real-time steaming data visualizer, including register and bit values, allows for easier debugging and testing within the IDE. MPLAB is also developed for the major operating systems on the market today, including macOS X. There is also a web based MPLAB Xpress IDE that allows for faster and more portable development for different aspects of the project.

3.5.2 Microcontroller Programming Hardware

As previously mentioned, both of these options require an additional debugger hardware piece. Though there a multiple options for debuggers that both IDEs are compatible with, using ATmega 1280 limits that playing field of possible debuggers to two main options: MPLAB PICKit 4 and

the ATMEL-ICE. The ATAMEL-ICE is more directed towards the use of ARM based SAM microcontrollers, and the price of the unit eliminates it for being considered for this project.

However, the PICKit 4 functionality meets all the requirements for the project at about half of the cost and is more versatile in portability. The debugger has the capability to be programmed via a micro SD card and be powered by the target board. It is capable of both 2 and 4-wire JTAG and Serial Wire Debug. The PICKit is easily connected via USB to micro-USB to the developing computer, and connected to the microcontroller via six pins as depicted below:

If development is to stay in the ATmega family, the MPLAB X IDE is used in conjecture with the PICKit 4 to program, debug, and test microcontroller for the software aspects of the project. This option does require extra hardware to be purchased, and a slight learning curve for the IDE and the debugging processes.

Arduino Bootloader - The other option for programming and debugging the microcontroller is to program a bootloader using an Arduino through the Arduino Software IDE. The Arduino would serve as an in-circuit serial programmer (ISP). The ATmega is first programmed as a part of the Arduino kit. This option requires no extra hardware, besides an Arduino, which members of the development team already possess from previous projects.

Which also means there is no learning curve for the IDE or programming process for the Arduino. The bootloader is stored in the memory of the microcontroller and is programmed as flash memory. Using an Arduino as an ISP is already managed with settings within the Arduino IDE, and only requires four data pins. The pin breakout is depicted in the *Table 34* below:

Table 34: Pin connections for the ATmega 1280 and Arduino for Bootloader

Arduino/Board	MOSI	MISO	SCK	Voltage Level
Mega1280 /2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	5V

The only extra steps needed is programming the target board, in this case the ATmega 1280, with the bootloader, and the code necessary to do so is all open source software. The ATmega is connected to the Arduino in order to burn the bootloader. Once the bootloader is burned to the target board, the microcontroller can be used as a standalone module when connected to a power source. To upload new development code, the ATmega can be connected to the Arduino via wires on a breadboard and use the USB to serial converter chip to upload code.

Software Development IDE for MSP430FR6989

The other board option for this project is Texas Instruments' (TI) MSP430FRx. The software development for this board is manageable by TI created environment and debugger modules. This guarantees compatibility with the different hardware and software aspects, when configured correctly. The development team has had experience working with TI hardware and software environments, reducing the amount of time and resources for training.

Texas Instruments Code Composer Studio - To program the MSP430 for the project the development team would use Texas Instrument's well-known IDE, Code Composer Studio (CCS). It includes a C/C++ compiler in an Eclipse like framework that allows for no down time for getting started on projects. Other third-party library and software utilities can be added straight into a project with the TI App Center. For debugging, CCS includes memory mapping debugger for memory protection and validation.

Hardware breakpoints can also be implemented within CCS to test and debug data and signal transfers between different pieces of hardware. EnergyTrace Technology is an application that can be used within CCS to analyze the microcontrollers power consumption through different processes in order to optimize it to be low power.

In order to load software onto the MSP430 from CCS, a development kit or evaluation board is needed to connect the two. The TI Launch Pad Development Kit is a common evaluation module for programming the MSP430x family. It includes a board emulation for programming and debugging code. The Launch Pad also allows for burning an MSP Bootloader. The Launch Pad is equipped with additional hardware for development that would not be used in the scope of this project and would be unused. Another development board is the MSP-TS430PZ100D standalone target socket board. Through a JTAG interface the MSP430 can be programmed through a 2-wire protocol. The standalone board is very basic and has no additional bells and whistles when compared to the Launch Pad. Making it a better choice for the scope of this project since the objective is to just download code for testing onto the microcontroller. The schematic of the MSP-TS430PZ100D board is below for referencing the JTAG connector and microcontroller orientation:

Texas Instruments IAR Embedded Workbench - IAR Embedded Workbench is TI's dedicated C/C++ compiler and debugger for MSP430 microcontrollers. Similar to CCS, the Workbench includes EnergyTrace Technology to optimize power consumption through different processes in the code. It also supports library and linker tools, straight into the project, increasing the capabilities and functionalities of the code.

It has an MSP simulator for real time operating system debugging, and TI specific operating system debugging. IAR Embedded Workbench requires similar hardware as CCS to communicate with the microcontroller, development kits such as the standalone MSP-TS430PZ100D board. The communication would also be via JTAG 2-wire protocol. However, IAR Embedded Workbench is not compatible with the TI Launch Pad development board.

Software Development Environment Conclusion - With both hardware capabilities, and software development options investigated. The development team has decided to use the ATmega 1280 microcontroller for this project. The base hardware functionality needed for the project is met by the ATmega 1280 and the software development process for the microcontroller seems more flexible than opposing TI hardware.

The additional TI hardware needed for the software development processes greatly exceeded the price of the ATMega hardware as well. Due to the ease and possession of the required hardware, the development team has chosen to program the ATmega 1280 using bootloader operations via

the Arduino. This decision saves approximately $55.00 USD for the overall project budget and will save a predicted two weeks of extra training for using the MPLAB products for development.

3.5.3 Weather Application Programming Interface (API)

In order to provide customers with a reliable and timely execution of the system's main purpose of responding to weather and possible weather threats, the software on the microcontroller will run two main programs. The two programs will run algorithms for parsing the data collected by the sensors and interpreting the Weather Application Programming Interface.

An Application Programming Interface (API), facilitates communication between a server and a database. In the scope of this project, a Weather API is used to track data of the surrounding forecasts and weather alerts. The Weather API will allow for the optimum opening and closing of the roof in response to weather predictions. The API would solely be used for predictive measures. Using alerts such as lightning strikes and tornado warnings, will add extra security to the customer's structure. A possible stretch goal for the phone application would be to incorporate these warnings and the API to send push notifications to the user on their device about weather and the roof operating statuses.

The basic data the project would need from the Weather API would be: precipitation probability, air quality, lightning strikes, and weather warnings. This data would be pulled from both the current conditions along with a twenty-four hour prediction. The purpose of the Weather API is to make predictions to have the roof react before a storm or hazardous weather conditions reach the structure, and with the weather being very temperament, there is no need to have predictions based twenty-four hours.

There are multiple Weather APIs on the market that are free, or have limited free plans, based on the number of request calls to the database. For development purposes of this project, APIs that meet the specific data requirements and are free or limited free are taken into account. Implementation of the product for customers could include a paid API for better accuracy and more analytical data.

OpenWeather API - One popular option on the market is OpenWeather API. OpenWeather's free API subscription includes current weather, five day forecast (with three hour increments), UV index, and weather alerts. The weather alerts aspect of the API allows for triggers for different weather conditions such as wind and temperature. OpenWeather's free subscription allows for up to sixty calls a minute to the database. With the free subscription the API is updated every two hours or less. The API calls can be made specific to city, zip code, or geolocation, allowing for increased accuracy in data pulling. OpenWeather API data can be returned in both JSON or XML format, allowing for more flexibility for web development.

Dark Sky API - Another option that includes a free limited subscription that meets most of the needs of the project is Dark Sky API. Dark Sky's subscription includes, but is not limited to, access to data for UV index, nearest storm distance, nearest storm direction, and liquid precipitation rate. The API forecast capability is down to the minute, but also extends to up to a seven day forecast. Location for the API is set based off of geo-location. The API has a Time Machine Request, that

returns the forecasted conditions for a specific date in the future. Data requests from the API are returned in JSON format. The first one thousand calls in a day are free, any more calls made are charged $0.0001. To remain cost free, the algorithm would be restricted to about forty-one calls an hour, or a call about every minute and a half.

Additional Lightning API - The above APIs provide data to reach most of the weather API goals, but both are missing data on lightning strikes. Measuring lightning strikes with sensors has a low accuracy due to noise and light pollution, making an API the only solution to be able to track lightning. Most lightning APIs come as a part of other paid for packages for other APIs. MeteoGroup has a lightning API that provides real time data of lighting strikes for a specific time period for a specific region. The region is determined by longitude and latitude. Requests can be made to only return lightning strikes within a certain range and certain time frame, which can be configured, of a geographical location. Request calls returns JSON formatted data.

Table 35: Weather API Comparisons

API	Maximum Free Polls	Main Functionality	Additional Cost
OpenWeather	60 per minute	• Current weather • UV Index • Weather Alerts • 5 day forecast	$40 per month
Dark Sky	1000 polls a day	• Current weather • UV Index • Nearest Storm • Time Machine Request	$0.0001 per call
MeteoGroup	Every 30 seconds	• Location of lighting strikes	Free

Table 35 holds a recap of the information, relatable functionalities, and cost for the weather API options for the software side of this project. To meet all the needs of the weather API, and to provide a timely and effective reaction of the roof, the Dark Sky API and the MeteoGroup lightning API will both be used to predict the opening or closing of the roof to avoid extraneous weather that would not be picked up by the sensors until the storm reached the structure. The limited free subscription allows for adequate request calls to the database, and provides more weather data and more accurate forecasting to the minute.

3.5.4 Sensor-Input Management

With incoming data from both the Weather API and the sensors, the software needs well defined trigger events to have the roof react. The sensors' data will work together to determine the current weather state at the unit. A weather condition or event will not be defined by just one sensor, but is confirmed by another to increase accuracy and effectivity of the unit. The storage of this data will need to be done in an external source as our microcontroller does not have enough space for

the multitude of incoming data we are excepted to have. The external storage is done through the use of a database. This section will detail the comparisons of various databases available for use, and will concluded with a decision on which database is utilized for this project.

3.5.5 Database Storage

To increase user customizability, multiple settings for the roof and the roof reactions can be changed. These changes need to be stored for each enclosure's kit. The database can also store historical data for the weather tracking and analytics. The database would be referenced from both the phone application for user preferences and data, and the microcontroller for sensor data history and Weather API data. Most of the computing power the database would need to perform would be simple arithmetic equations for converting the data from the sensors.

Simple security for logins of users would be needed, but nothing more than password protection. Password protection is provided by simply hashing user passwords. This opens the options for databases up to all the popular databases available. The biggest thing to consider with choosing a database for the project is the fact that the phone application is an iOS application. Databases that have strong compatibility with iOD applications is considered. The following sections will analyze the pros and cons for the different databases, and how applicable they are to the project.

MongoDB - MongoDB's Stitch is a serverless platform that connects MongoDB Atlas databases to both iOS and Android phone applications. MongoDB Stitch is mainly controlled with JavaScript queries for data and backend functions. Database triggers are one main functionality that would help with the functionality of this project. Triggers allow for specific non server based functions to execute whenever any data within the website is changed. They are applicable to user authentication and data changes.

MongoDB Stitch utilizes MongoDB authentication, which provides regular user authentication via username and password, but also supports external authentication through social media and third party accounts. Multiple third party libraries, such as Twilo (cloud communication interface within applications), are compatible and easy to integrate into the application backend for more functionality in the phone and web applications. MongoDB is free based on the size of the database, up to 25GB downloaded per month, which the development team does not expect to exceed, because historical data will not reach back more than forty-eight hours.

Using MongoDB would require extra time and training for the development team, as no past experience of using MongoDB for phone applications is available. Extra time and resources for combating the learning curve for the database needs to be taken into account. However, Firebase is a free open source database that would not cost the project any monetary resources.

For testing and development purposes, Stitch creates a new draft state that needs to be published every time to change existing application versions. This is helpful for maintaining and managing bugs during transition but does not support beta version testing for development releases. New drafts can be deployed straight from GitHub with a hook.

Google's Firebase - Google's Firebase Database for phone applications supports real time hosting and database solutions with cloud storage both off and online. Firebase uses NoSQL databases for real time cloud storage. This allows for custom data storage and queries in collections and documentations. Combined with Google's storage capabilities, data can be synced easily both online, and managed offline.

Google's SDK allows for pauses in data downloads and transfers for when connectivity is lost or low. Google's cloud storage also includes Google's security for files and applications. Firebase Authentication is the main security protocol, and includes usernames, phone numbers, and popular social media platform authentication for all users. Data queries are written and returned in JSON formats through Firebase.

This complies with all of the Weather API choices, keeping queries in one coherent format for the project. Firebase supports serverless web and phone application development, allowing for one less middleware, decreasing the need for additional unit or end to end testing for new APIs. Part of the development team has experience creating an iOS based Firebase application, which eliminates part of the learning curve and extra time and resources for training.

Google's Test Lab allows for a simulated testing environment that can be used to test all functionalities of both phone and web applications. This would come in handy for trying to test API triggers, that may not be as frequent due to the sporadic and uncontrollable behavior of the weather. Firebase's app distribution also allows for simple app releases to both iOS and Android devices for testing and beta distributions.

SQLite - SQLite is a popular choice for phone applications as it is an embedded database, which runs off of C based programming and SQL queries. SQLite is a free and open source domain, making it easy to apply and find documentation for implementations for this project. SQLite is a complete SQL database, including all the functionalities and services of a SQL database, plus more. SQLites provides no learning curve for the whole development team, therefore saving both time and resources. No extra resources, both time and monetary would be needed for implementing a SQLite database for the project.

SQLite runs off The biggest aspect of SQLite is that is a part of the application itself. It may not be as powerful as other databases, but it does not require as much backend, as it is literally a part of the application. It is stand alone and self-contained within the application. SQLite can be paired with other development platforms such as Swift to increase its functionality and scope. SQLite does tend to perform better with larger amounts of data, which is built up over time, but may be hard to test at the beginning of development.

Testing the application through SQLite is as simple as downloading the development app straight from Xcode. SQLite does have built in testing capabilities in different libraries to verify data insertions. There are three different branch testing that are included in SQLite for different parts of the development process: TCL Tests, TH3 Test, and SQL Logic Tests.

TCL Tests are aimed towards testing development relations. TH3 Tests cover branch testing and would be used for testing the embedded platform on the microcontroller for this project. SQL

Logic Tests are used to test the validity of the database data and relations across the whole board. The database can be tested and implemented easily through the application development environment but would require new releases to test changes.

The options for databases all include serverless solutions in order to minimize the extra step and possible intrusion of error that a server requires. The project requires storing of sensor data for the past forty-eight hours, and only requires minimal analytic computation to provide the user with the weather conditions.

No final decision for which database is used has been made, but Firebase is the leading contender. It provides vast flexibility for data both off and online. It also provides ease with control over documentation and queries for data to the application. Google's environment allows for heightened functionality and security for the database and its users.

3.5.6 Bluetooth Algorithms

For reference as to what specific hardware is used for Bluetooth, please reference section 3.4 of this paper. This section will outline the basic communication structure between the three main points of contact in this project: microcontroller, database, and user's phone. All other communications in this project are hardwired within the microcontroller.

The Bluetooth's main connection is with the user's phone for setting up the kit and change any settings or implement a schedule. The user's phone will create a connection via connected Wi-Fi or cellular data services to the project database. The following diagram illustrates the possible connections for the Bluetooth and Wi-Fi in this project. Note how the arrows are double headed to indicate that the connection will allow for data to flow both to and from the microcontroller on both communication methods:

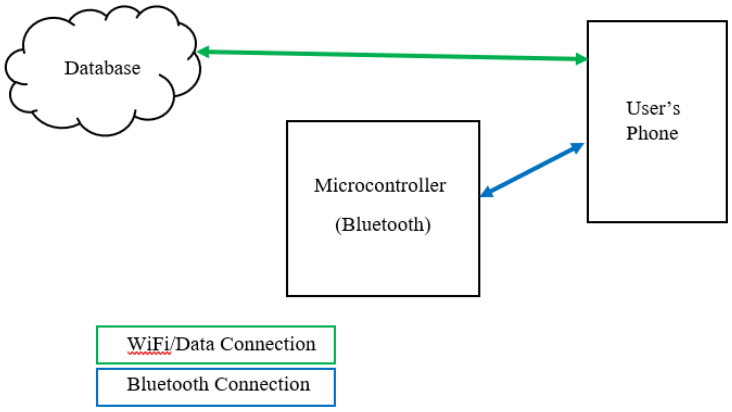


Figure 3: Bluetooth and Wi-Fi connection model

When a user sets up a new unit, they will connect to the unit via Bluetooth, and verify their unit with a serial number. They will set their default settings and schedule, and any time after initial set-up, if those settings change, the new data is pushed to the microcontroller via Bluetooth

connection. That data will also be extended into the database to be stored and referenced when a trigger event occurs.

One of the stretch goals for the project was the implementation of push notifications to the user's phone when a weather event is triggered. The push notifications is sent from the microcontroller to the user's phone via Bluetooth. Meaning a push notification's path would start at the microcontroller and be sent to the iOS application, which will push the notification to the user's phone.

3.5.7 iOS vs. Android Operating Systems

When choosing between which of the two development environments were to be used it came down to the two the title says. The team knew that a mobile application was wanted, so that quickly determined that the decision was going to come down to the two global phone leaders in the market.

Market share was one of the first things considered when looking to make a decision. Apple's iPhone currently holds the lead in the United States smartphone battle, having 41% of the US Smartphone Shipments Market Share (%) as of Quarter 2, 2019. If one is to look at the statistics globally, the story is a little different, as Android holds majority market share. So, when making our decision, it was clear that if we were to target a global audience, Android needed to be our choice.

Android - Although Android is the leader in the global market, Android fragmentation is one of the main problems that developers must take into account. This fragmentation occurs due to the open nature of Android. When initially launched, Android differed from iOS in that developers had more liberty to customize the base operating system as they pleased, leading to a bit of a different experience per manufacturer. This is an issue when the owner of Android. Google Inc. decides to push and update the base version, as it each manufacturer's responsibly to make the proper modifications in order for this newer version to be pushed to their respective devices.

This leads to obsolescence of older devices, as the business side of things pushes manufactures to stop the support older devices, forcing individuals to buy their newer devices. This causes users to be left with versions of the Android OS that lack the newer features, forcing developers to consider how far back they wish to go in support of aging versions of the operating system. Not only that, due to the competitive nature of the market, manufactures have differing device features, further complicating a developer's work. A lack of standardization means more devices, components, and software fragmentation to account for.

Apple - Fragmentation is different for Apple devices. Because Apple decided to not make their operating system open-source and is responsible for the development of both their operating system and devices, the fragmentation issue does not affect developers as much. Apple still pushes yearly updates, as Apple comes out with a new iPhone, and this new iPhone usually comes with a newer version of iOS with it, but this doesn't affect things as much.

Apple's control over their products allows for them to easily customize their software for the various iPhones still out on the market. This makes obsolescence not be as pronounced, allowing the push of the latest OS to older devices. This in turn is helpful for developers, since the concern over supporting the older devices in not as bad. Not only that, iPhones tend to all have similar features, with the few differences coming in the newer versions. These differences aren't usually too major and are easily identifiable from device to device, making a developer's work much easier.

Making an App for iOS is faster and less expensive. Although this may be a subjective statement, the consensus amongst individuals and forums in the internet claim that it's faster, easier, and cheaper to develop for iOS — some estimates put development time at 30–40% longer for Android . It is easier to develop for iOS because of the code. Android applications are generally written in Java, a language that involves writing more code than Swift.

Also, as it has been repeatedly stated, Android is an open source platform, which does add a bit of complication; Apple's closed ecosystem means you're developing for a few standardized devices and operating systems, making it easier. It is also important to note that Apple's closed nature, is what gives it its perception of being more secure, why iOS has a larger audience in the enterprise market.

Phone Application Conclusion - Market share, fragmentation/maintainability, the ease, overall cost, and security were the major players in making a final team decision. Other factors such as features and aesthetics were considered, where ultimately, Apple became the platform of choice. iOS' gestured-based interface was one that heavily influenced our decision, as iOS-unique features, such as 3D touch, among others were desired for OpenAir's application. Also, the fact that all team members have iPhones and are familiar with its environment, a preference in appearance and style was evident amongst team members, as well as the fact that having the devices available to every member made testing much easier. The lack of having the Android open-source environment did lead the team to give up a bit of freedom, however in the end, that did not matter, For all the stated reasons, it was evident that an iOS application was the best option for the purposes of the A.I.R.E. system.

3.5.8 Swift vs. Objective C

Before going over all the technical aspects of the Aire Controller, it is important to discuss why the team chose to program the application with Swift. It was already covered why for this project, the iPhone operating system was more suitable in comparison to Android Operating System. This, however, was not the only initial decision that needed to be made.

Application development in the iOS environment can be done with two programming languages, Swift and Objective-C. Of course, there are pros and cons to using the two, and the team had to analyze all these things before ultimately making a final decision. This section will compare and contrast the two-programming language and discuss why the team decided to go one route instead of the other [9].

Objective C - To begin, there are some comparative advantages to Objective-C over Swift. Objective-C interoperability with C++ and Objective C++ is one of the main ones. Because all these languages have been around for so long, developers around the world have developed a variety of libraries that are useful. This specifically applies to this sort of Internet of Things based project, as there are plenty of libraries created by individuals that allow for interfacing with a microcontroller, giving access to all its different functionalities.

Another comparative advantage is Objective-C's dynamic features like method swizzling. Method swizzling is the process of changing the implementation of an existing selector. It's a technique made possible by the fact that method invocations in Objective-C can be changed at runtime, by changing how selectors are mapped to underlying functions in a class's dispatch table.

The last advantage worth mentioning for Objective-C is the programming language's better support for writing binary frameworks. This comes into play when it comes to the business aspect of the application, as one could have protected the source code and build a business out of the created framework, since the team's application can be used for other iOS developers trying to accomplish a similar IoT project.

When looking at the disadvantages, the list outnumbers that of the advantages. One thing is that Objective-C lacks name spacing. Much like C, this requires everything to be within one global namespace, which could lead to issues if one doesn't prefix classes as it is done in common practice. Explicit pointers is another inconvenience that makes up the language, as memory management must be taken into account due to this. Objective-C allows developers to send nil objects a message. This and the lack of strict typing leads bugs that are difficult to find and solve. On top of it all, Objective-C's complex syntax does not help its case when it comes to choosing a language for development.

Swift - Swift on the other hand has a list of advantages that far outnumber its number of disadvantages, as well as the number of advantages that Objective-C possesses. For one, static typing, optionals, and optionals chaining, all make Swift a safer language. A developer is less prone to encountering issues thanks to the language's support for namespaces, functional patterns, and the simpler more concise syntax.

Playgrounds, a Swift based application available in the Apple App Store allows for interactive development, a huge advantage for the development within the team. On top of it all, the biggest advantage of using Swift has to be the SwiftUI, which is a declarative framework that gives developers the ability to make user interfaces for multiple platforms. The preview of the SwiftUI is found within XCode itself, which allows developers to avoid any sort of iPhone simulator, which is usually filled with bugs that add difficulties to a programmer when creating an application.

Of course, to properly be able to compare and contrast the two programming languages at hand, it is only fair to list the disadvantages that developers face when using Swift. The first, which is an advantage of using Objective-C, is the lack of direct access to the thousands of useful C++ libraries that exist, by far the biggest disadvantage. From the business perspective that was discussed earlier, the lack of module format stability makes it more difficult for the team to share the code in a private manner as a binary framework.

Lastly, Swift is known to compile at a slower rate than Objective-C, something that does not affect the making of this project. It is very evident that using Swift was the best way to go about creating the Aire Controller application. Although some aspects of the application development may have been easier with the larger pool of resources that come with Objective-C, Swift's comparative advantages made it very clear to the team which route to take.

4. Standards and Design Constraints

The following tables are used to demonstrate the different constraints and standards that guided the design of the project. These constraints and standard parameters are formed in conjunction with the identification of the project's requirement specifications and are also decided based on the end user's needs, the manufacturer's resources, and a plethora of other outside factors. Below are tables 36 - 39 that break are broken up into hardware and software standards, followed by the hardware and software constraints.

Table 36 specifies all relevant hardware standards and specifications for our project that aided us in our parts decision.

Table 36: Hardware Standards

Description
The Bluetooth module selected will comply with Bluetooth Core Specifications 5.1
The Wi-Fi module will comply with IEEE 802.11 and broadcast on the Industrial, Scientific, and Medical (ISM) band of 2.4Ghz
All electrical and electronic components will comply with RoHS 3 directive 2015/863
All sensors will comply with IEEE 2700-2017 Standards for Sensor Performance Parameter Definitions
The solar panels will comply with AES/SS IEEE 307-1969 Standard Definitions of Terms for Solar Cells
To maintain manufacturability, electrical components chosen for the project shall be manageable for hand soldering

Table 37 specifies all relevant software standards considered when creating the initial design of our firmware and iOS Application.

Table 37: Software Standards

Description
To program the chosen microcontroller, a debugger is used via a USB connection, and it is used in compliance with USB 3.0
Microcontroller software will need to be written in ANSI C

The next two tables 38 and 39, similar to the ones above, will serve the same purpose, showing the hardware and software constraints that were considered throughout the design of the project.

Table 38 specifies all constraints considered when creating our project. These limited some of our ideas for our design while also providing a more specific path for us to take during initial design.

Table 38: Hardware Constraints

Description
The total cost of all hardware ordered for this project should not exceed the sponsor's stipend of $2000 USD.
The wireless communication system shall contain a Bluetooth module compliant with the Bluetooth 5 standard
The wireless communication system shall contain a wi-fi module compliant with IEEE 802.1
All devices shall be compliant with UART, SPI, and/or I2C capability for communication with the microcontroller
All passive components shall come in packages greater than 0608 and all integrated circuits shall have leads extending out from its package
All electronic components shall be RoHS-compliant

Table 39 specifies all software constraints and limitations placed on us during the programming process.

Table 39: Software Constraints

Description
iOS application constraints is based on the iOS version and phone model for design and layout of the app
App release is constrained by Apple Inc.'s app requirements.
The Bluetooth chip CYBT-423028-02 code is developed within the Wiced integrated development environment.
The PICKit 4 cannot be connected to pull up resistors or capacitors on the data or clock line.

As the project continues our constraints and specifications can and will change, but during this stage we have found the above limitations to be our main concerns.

4.1 Design Impact of Standards and Constraints

As with all projects, there exists standards and constraints that can impose wide-spreading impacts to the designing of the system. The following paragraphs will narrate the impact of the standards and constraints applied to our project.

The hardware standards applied to the project will ensure that the sensors and wireless communication modules is able to interact with our chosen microcontroller without the use of extensive hardware for interfacing said communication. The RoHS standards applied to our project will maintain a low environmental impact for our system as a whole, leading to a greener future. The sensor and solar panel performance standards ensure that our sensors and solar panels is the best in the industry.

Software standards applied to our project will ensure that the microcontroller we choose to utilize is programmable in a language that all of our team members are familiar with, which includes ANSI C and MIPS Assembly.

The constraints in our project arise from the from multiple sources. The first constraint, relating to the overall cost of our design, is a constraint applied to the project due to the budget allocated for this particular design. Totaling $2000 USD, our budget will need to be maintained in order to keep our system low-cost while getting the most performance out of our limited component count.

The constraints related to the wireless communication used by the modules will ensure that our microcontroller can interface with any Bluetooth or Wi-Fi devices that may represent any user that

wishes to utilize their standard industry device to connect with our system. The digital communication constraints relate to the typical method of communication between microcontrollers and other digital components.

The constraints related to manufacturability and package size will allow us to hand-solder parts when assembling our prototypes. Finding packages with these constraints is not very trivial due to the diversity of parts available for all applications from a myriad of manufacturers.

The software constraints represent constraints placed on us by the specification of application we will use. Mobile devices utilize primarily iOS or Android operating systems. Since we have chosen to use the iOS operating system, our constraints will relate to the iOS and Apple terms and conditions of use of their operating system.

5. Project Hardware and Software Design Details

Section 5 is designated for initial design details based on our strategic parts selection from Section 4. Each subsection in this will relate to how individual submodules is designed based on the applications notes from datasheets, or from our own designs based off material learned from the curriculum so graciously provided by UCF. This section will also include depictions of how all of the project subsystems will come together in the end to create a final working project. This begins with initial schematic designs and software ideas before creating PCB layouts and program design.

5.1 Hardware Design

This section includes any physical designs that is used. This entails the subsystem design of the power module including the battery management system (BMS) and the voltage regulation system. This also includes the circuit design of all sensors, and how they connect to the microcontroller. This will also include discussions about the power, microcontroller, and motor modules. Lastly, a depiction of how all hardware aspects will connect together to create the final project layout is discussed as well. The high-level lay out for the hardware aspects for this project are depicted below.

5.1.1 Power Module

The following section details all initial and proposed design solutions for our power module printed circuit board (PCB). This includes battery management system design, all voltage regulation designs, and the implementation of the solar panel. All initial schematic designs is shown in the section followed by implementation of these schematics onto our proposed PCBs.

Battery Management System - MCP73842 - The battery management system will utilize the MCP73842 to oversee the charge and discharge cycles of the lithium ion battery pack. The BMS IC is be configured similar to the application schematic provided in the datasheet. An example of this is provided in *Figure 4* below.

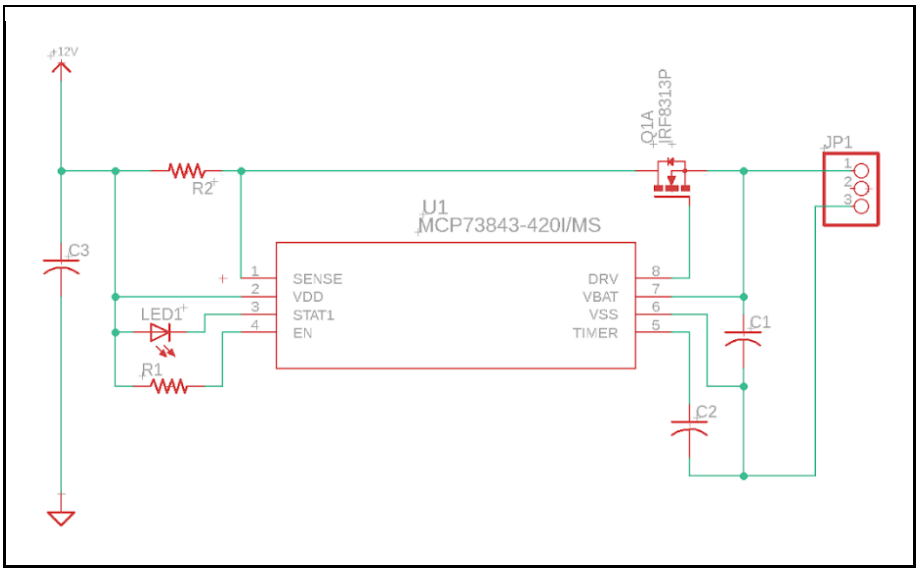


Figure 4: BMS IC Single Element

The optimal efficiency for the 12V buck/boost controller occurs when the input voltage is between 12 and 20 V. For this reason, we is setting up a 3-series configuration of lithium ion battery cells. In order to accommodate this, we is setting up 3x BMS IC elements in series that is responsible for the safe charging and discharging of the lithium ion battery pack. Because of the design of our battery management and voltage regulation system, the 12V regulator is the first in line after the BMS, therefore the BMS must be configured to work with the 12V regulation system. All other regulation system will follow the 12V regulator and won't be directly connected to the power sources.

12V Buck/Boost Controller - LM25118 - The 12V buck/boost converter we chose to use is the LM25118. Utilizing the datasheet, we came up with an initial design schematic, which can be found in *Figure 5*below. This converter requires the most external circuitry as it takes all of the power from the source and converts it to the required 12V. This output voltage then goes into the 5V regulator and to the Bluetooth module. Having a correct and sophisticated design for this portion of the power module is extremely important if we want the project to operate as required.

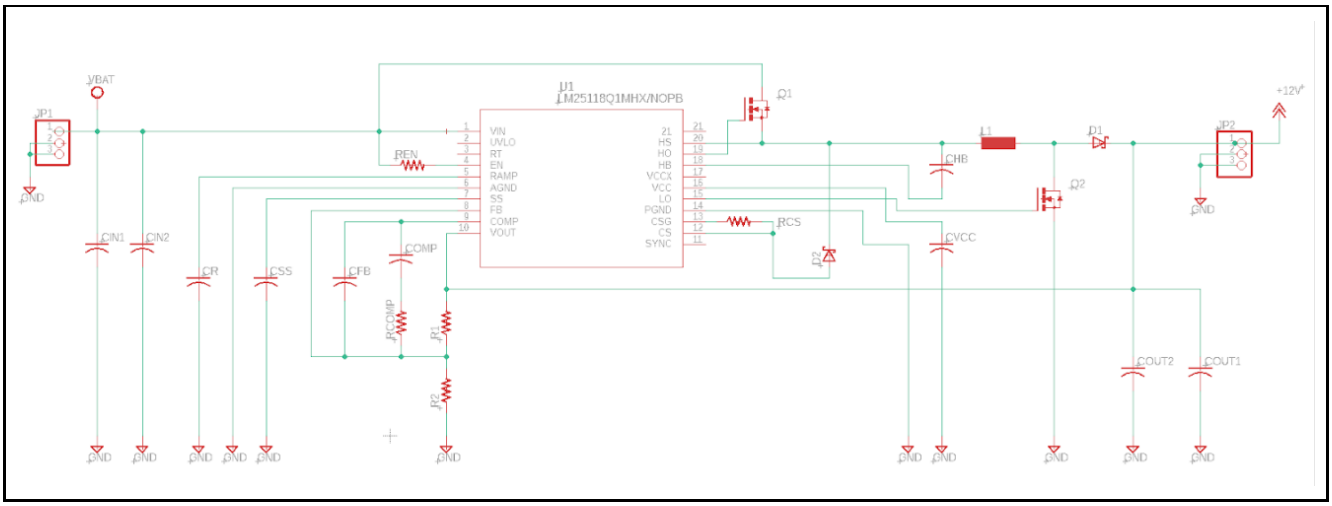


Figure 5: Initial design of 12V Buck/Boost Controller

In the schematic, CINx and COUTx are bypass capacitors responsible for filtering out any voltage spikes that might be picked up in the environment. CR, CSS, CFR, COMP, and RCOMP are components recommended in their typical application schematics. R1 and R2 serve as a voltage divider for providing feedback.

On the right of the buck/boost controller contains the output components. Q1, L1, and D1 serve to provide high-side output current, while Q2 and D2 serve to provide low-side output current. The other components are recommended in the datasheet's typical application schematics.

Two jumpers serve as temporary pins for supplying the battery voltage, VBAT, and output voltage, +12V. These jumpers is replaced with a more appropriate pin header when final designs are created.

5V Buck Converter - LM2596 - The 5V buck converter we chose to use is the LM2596. Utilizing the datasheet, we came up with an initial design schematic, which can be found in *Figure 6* below.

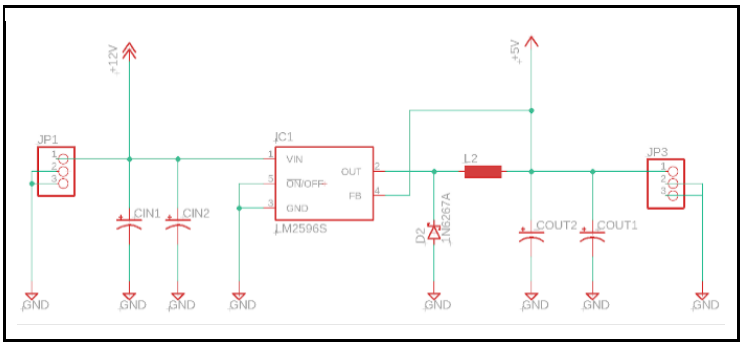


Figure 6: Initial design of 5V Buck Converter

In the schematic, CINx and COUTx are bypass capacitors responsible for filtering out any voltage spikes that might be picked up in the environment. D2 and L2 are responsible for providing the output current during normal operation.

Two jumpers serve as temporary pins for supplying the input voltage, +12V, and output voltage, +5V. These jumpers is replaced with a more appropriate pin header when final designs are created.

3.3V Voltage Regulator - LT1763 - The 3.3V voltage regulator we chose to use is the LT1763. Utilizing the datasheet, we came up with an initial design schematic, which can be found in *Figure 7* below.

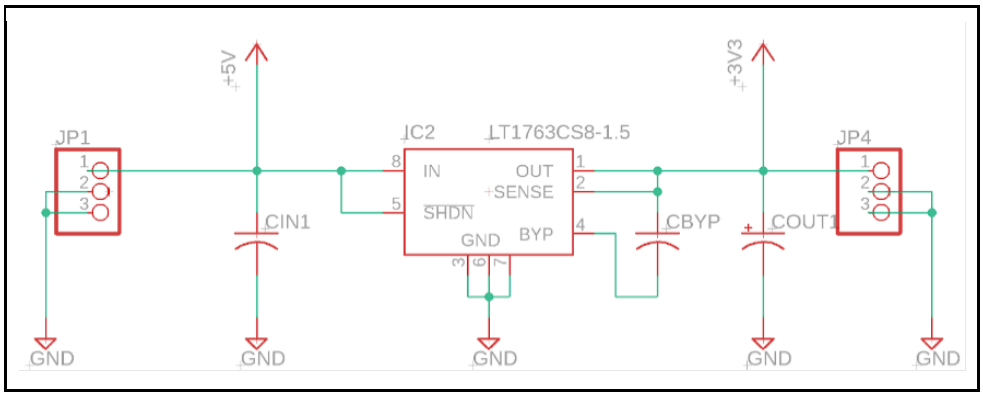


Figure 7: Initial design of 3.3V Voltage Regulator

In the schematic, CINx and COUTx are bypass capacitors responsible for filtering out any voltage spikes that might be picked up in the environment. CBYP is a bypass capacitor required for normal operation.

Two jumpers serve as temporary pins for supplying the input voltage, +5V, and output voltage, +3.3V. These jumpers is replaced with a more appropriate pin header when final designs are created.

Reverse Polarity Protection - Our initial design for reverse polarity protection is shown in *Figure 8* below.

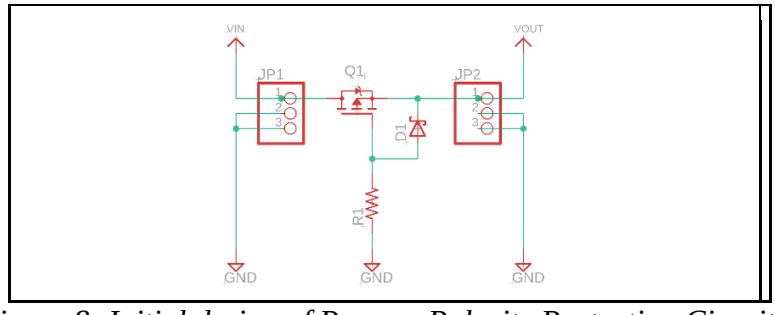


Figure 8: Initial design of Reverse Polarity Protection Circuitry

In the schematic, Q1 is the primary component responsible for protecting the Vout circuitry from negative input voltage. When positive voltage is applied, it turns on and allows current to flow.

Two jumpers serve as temporary pins for supplying the input voltage, VBAT, and output voltage, Vout. These jumpers is replaced with a more appropriate pin header when final designs are created.

5.1.2 Sensor Module

The following sections display our proposed initial designs for our sensor schematics and pin layouts. Each sensor requires a voltage regulating circuit from one of our chosen regulators displayed in sections 5.1.1. These regulating circuits are not displayed here, only the components of the circuit which directly connect to the sensors are laid out in the following schematics. All schematic designs are based off provided schematics given in the datasheets, as well as any documentation found in the datasheets or through outside research. All schematics will then connect together on the final PCB layout to create a working and compact sensor layout.

Proximity Sensor – MP102103 - Our chosen proximity sensor is the MP102103. Below in *Figure 9* is our proposed design, where the output pull-up resistor value will depend on the supply voltage. The datasheet for the proximity sensor recommends an external pull-up resistor of 1kohms or 2.4kohms for 5V and 12V supply voltages, respectively. In the circuit, VCC is the supply voltage, which should be in the range of 4.5 - 13 V. This schematic was integrated into our sensor PCB, but was never used as we had to remove this part from our final design.

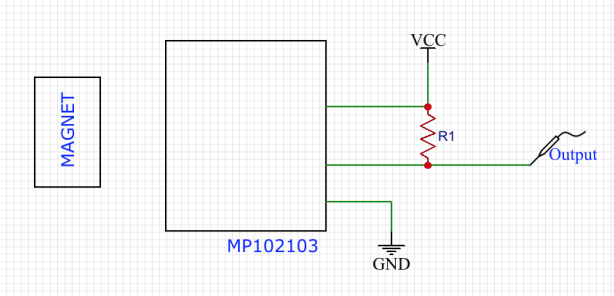


Figure 9: Initial Design of Proximity Sensor

Anemometer – Jacksking – Detailed in this section is our proposed schematic design of the anemometer we intended to use in our system. The chosen anemometer for this project is the Jacksking cup anemometer. Because it uses pulse width modulation (PWM) to measure the wind speed, acting as a switch, a debounce circuit is needed to stabilize the signal. Because wind is unpredictable, or asynchronous, this debounce circuit will need to connect to an interrupt handling software.

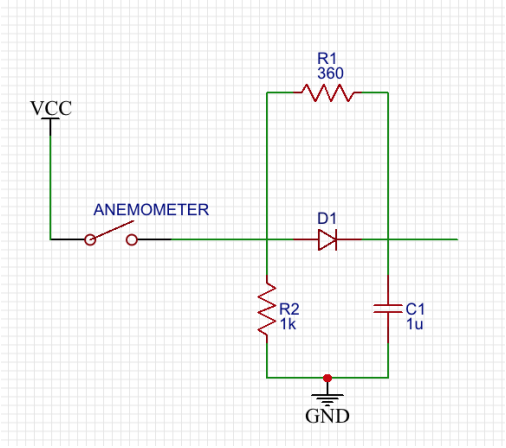


Figure 10: Initial Design of Anemometer

In this circuit shown in *Figure 10*, the capacitor stabilizes the output to prevent any overshoot, or extreme voltage dropping, while the diode controls current flow throughout the circuit. Our diode recommendation is the 1N4148 because it is simple to implement, can work with a wide range of input voltages, and works well overall for almost any application. Unfortunately, as mentioned in previous sections, this circuit was integrated onto the PCB, but was left unused.

Barometric Pressure Sensor – BMP280 - The barometric pressure sensor chosen is the Adafruit BMP280 microchip. Our proposed schematic for this sensor is seen below.

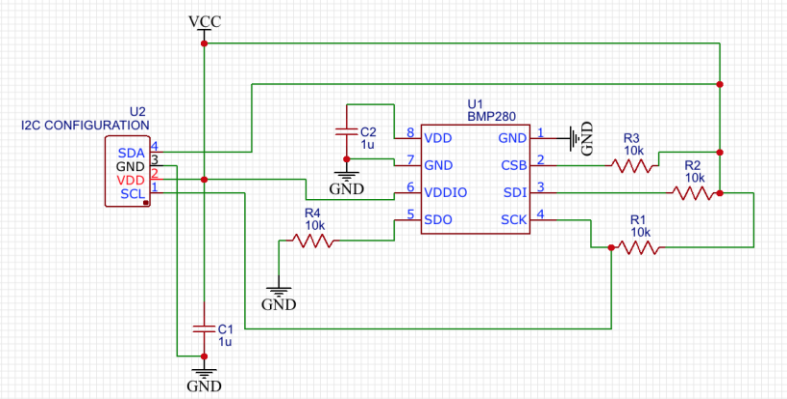


Figure 11: Initial Design of Barometric Pressure Sensor

When building this circuit as shown in *Figure 11* we must keep in mind that this sensor should only be operated on at a supply voltage of 3.3V, and the chip select (CSB) and serial data output (SDO) pins are only required if interfacing with SPI communication. Even though we ran into issues with other sensors, we were fully prepared to integrate this sensor and test it. However, during soldering and testing, this component broke and we were unable to buy a new one before UCF placed restrictions on the locations in which we were allowed to order parts from.

Humidity Sensor – SHT-85 - Our chosen humidity sensor is the SHT-85, which measures temperature and humidity, however we will only be focusing on this sensor's humidity sensing capabilities.

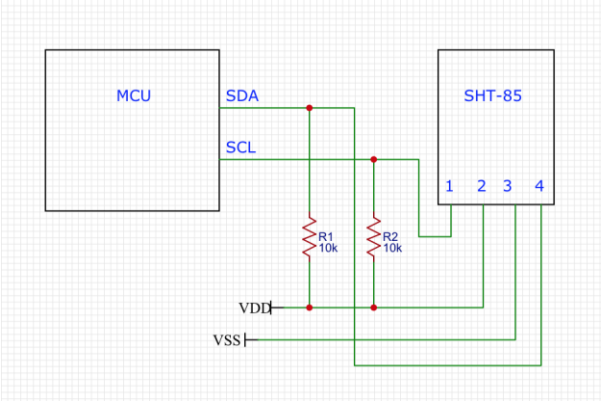
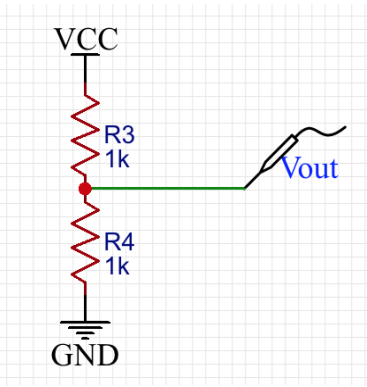


Figure 12: Proposed Humidity Sensor Design

This proposed circuit design in *Figure 12* equips the chip sensor with two pull up resistors, to create a higher output, and a stabilizing capacitor. The pull-up resistor values are typically 10k ohms but will ultimately depend on our bus capacity. In order to avoid signal contention, SCL and SDA must be driven low.

Temperature Sensor – B5786S - The chosen temperature sensor is the B5786S bead thermistor. To measure the temperature, the following voltage divider is proposed with the following equations that is implemented in the software. There is no need for added stability components due to the fact that the input to this device will already be a stable regulated value.



$$R_{thermistor} = R_{balance} \cdot \left(\frac{V_s}{V_{out}} - 1\right)$$

$$\frac{1}{T} = \frac{1}{T_o} + \left(\frac{1}{\beta}\right) \cdot \ln\left(\frac{R}{R_o}\right)$$

Figure 13: Proposed Thermistor Design with Relevant Equations

Where T_0 is the reference temperature, R_0 is the thermistor resistance at reference temperature, R is the thermistor resistance at the measured temperature, T is the newly measured temperature, and β is given in the component datasheet and depends on the thermistor resistance at reference temperature.

Rain Sensor – YL-83 - Our chosen rain sensor is the YL-83, which is a capacitive plate sensor. Because this sensor comes in two parts: a collector board and an electronic board, we do not need to design a circuit for this sensor. The sensor connects to a 5V power supply, ground, and analog and digital output pins. The electronic board is equipped with everything that is needed in order for this sensor to operate appropriately. All that is required is following pin layout as shown in *Figure 14*.

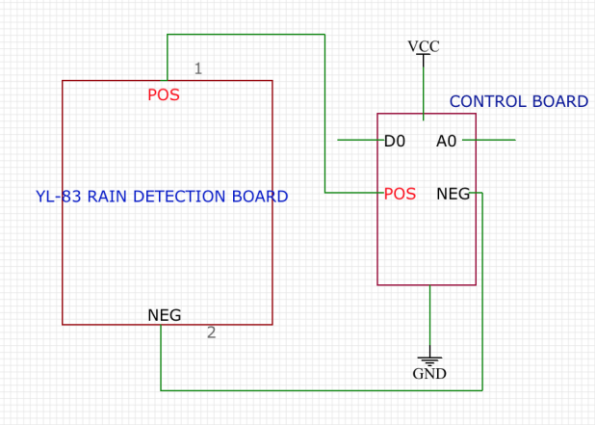


Figure 14: Initial Pin Layout of YL-83

Light Sensor – OPT101 - The light sensor that has been chosen for this design is an OPT101 integrated circuit. Because it is an integrated circuit chip, we will not have to worry about leakage current, noise handling, etc. Below shows our proposed design, which only includes a capacitor for output stability purposes.

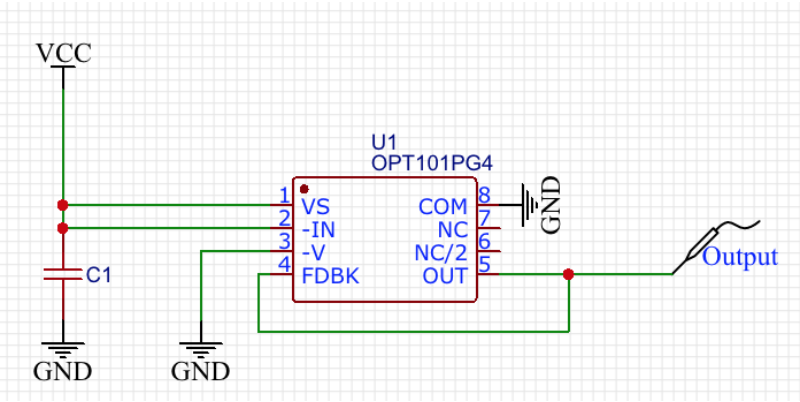


Figure 15: Initial Light Sensor Design

UV Sensor -VEML6070 - For UV protection and sensing we are using the VEML6070 UV sensor. Illustrated in *Figure 16* is our proposed schematic with each pull up resistor to be 2.2k ohm as recommended by the manufacturer, Vishay.

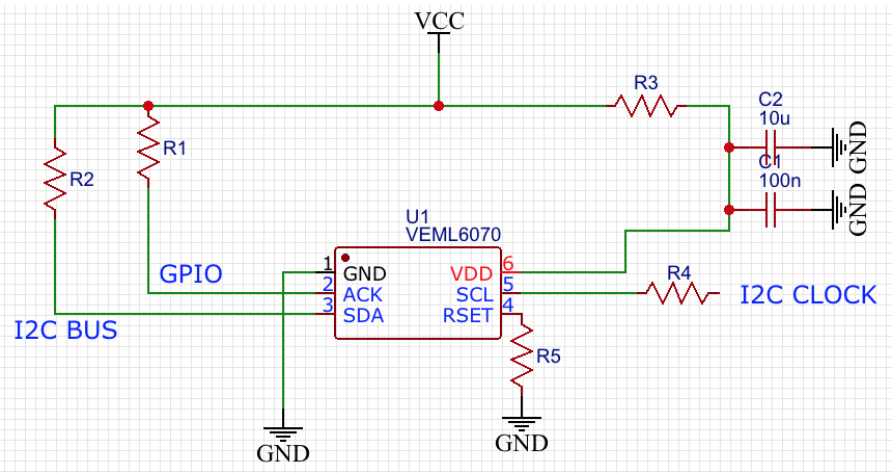


Figure 16: Initial Design of UV Sensor

If the supply is disturbed, unstable, or carries a lot of noise, the capacitor C1 and the resistor R4 are implemented to stabilize the input to the chip. The pull-up resistors are used to pull-up the output coming from the VEML6070 to the microcontroller.

5.1.3 Motor Module

The Progressive Automations - 07 motor's dimensions, all in inches, are drawn in *Figure 17*:

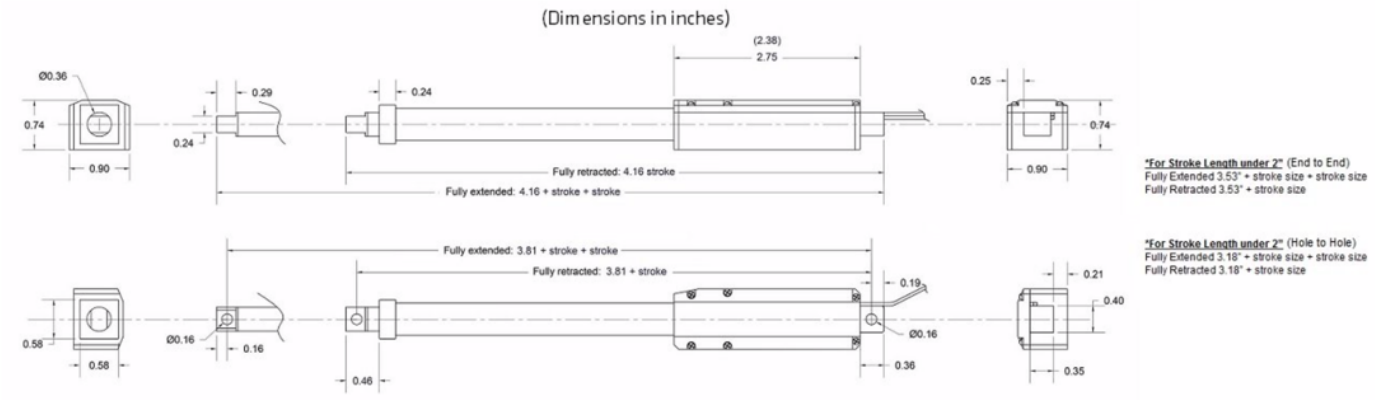


Figure 17: PA-07 Dimensions

The shape and size of this linear actuator was a main factor when choosing this specific design. As discussed earlier in the prototype section, it was easy to make modification to the top face of the initial module, since its two-inch thickness was more than enough to fit the height and width dimensions into the carve and trim done into the face. The cost of the linear actuator itself was also a good motivator for the team to decide to go with it, as these motors appear to have a cost over $100.00 in the market.

Giving some insight on how the functionality of the “PA-07” motor works, allows the reader to understand the reason for the overall design of the electrical side of things. The actuator is extended by a +12 VDC signal and retracted by a -12VDC signal.

One could go about providing these opposite Voltages by using a rocker switcher to do so, connected to the power supply. Due to the nature of this project, however, it being an intelligent system, the use of a physical switcher would not be useful. Luckily, any time one wants to add intelligence into a linear actuator application, you can use a micro controller and relays to control the actuator, since relays are effectively a switch controlled by electricity.

The relays used in this project to control the linear actuator are called 'Single Pole Double Throw' (SPDT) relays. These relays are 5V relays and is connected to the system’s microcontroller through a 5V relay board (The voltage necessary to power the microcontroller) [p]. The combination two SPDT relays, allowed us easily to reverse the polarity of electricity going to the actuator motor.

By controlling the polarity going to the motor, we can control the direction of travel for the linear actuator. Figures 18-21 serve as demonstration of how the different components are wired up in order to complete this action:

Figure 18: SPDT Relay Configuration Step 1

A small wire is used to connect the Normally Open (NO) connection on relay #1 with the with the Normally Open on relay #2. This is for our source +12VDC that is used for the linear actuator motor. These come from the + 12V Power lines provided from the lithium ion battery discussed in the power section of this experiment.

Figure 19: SPDT Relay Configuration Step 2

The same process is done for the Normally Close (NC) pins. A small wire is used to connect the Normally Close (NC) connection on relay #1 with the Normally Close on relay #2. This is for our Ground that is used for the linear actuator motor.

Figure 20: SPDT Relay Configuration Step 3

These two wires can effectively be connected to the power line for the A.I.R.E. systems, connecting the first and second wire (red and black in the photos provided) to +12VDC and Ground respectively.

Figure 21: SPDT Relay Configuration Step 4

With this combination, we are able to program the microcontroller and trigger the proper relay based on the action that wants to be accomplished. Whether it be to open or close the roof. Refer to the microcontroller algorithm section of this document in order to understand the logic in the code behind this operation. Two of these smaller systems is needed in order to complete the task for both actuators/roofs [10].

5.1.4 Microcontroller Module

For uploading new software onto the board during development, the microcontroller is connected to a debugger. The debugger is the interface between the computer and the microcontroller. The PICKit 4 is the debugger, paired with the MPLAB X IDE, chosen for this project. Please see software development section of this paper for more information on why the PICKit 4 along with MPLAB X were selected. One important note is that the PICKit allows for in-circuit debugging, which is extremely helpful during the project design proccess.

The following schematic illustrated in *Figure 22* is the initial design for connecting the PICKit to the microcontroller when it is in-circuit.

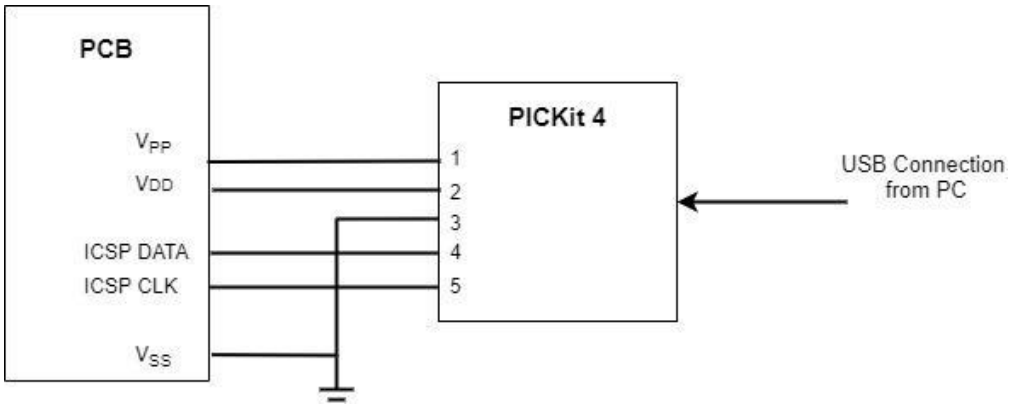


Figure 22: Initial Design of In-Circuit Debugger

5.2 Software Design

The following section and corresponding subsections will cover the software aspect of A.I.R.E., delving into detail regarding design, algorithm, intricacies, and implementations for both back-end and front-end of the iOS application. Lastly, communication between the application and controller is explained, from the application side of things, focusing on the Bluetooth procedure required for every subsystem to be connected within the overall A.I.R.E system.

5.2.1 iOS Application

A Swift-based application is used for the design, development, and testing phases of the software. Swift is a powerful and intuitive programming language for macOS, iOS, watchOS, tvOS and beyond, which is exactly the reason it is chosen as the programming language used for this project. Swift allows the team to provide a user-friendly interface that provides the following capabilities:

- Open/close roof
- Change default attributes
- Modify accesses and such based on account permissions
- Access weather metrics collected by system

All the stages of the Swift 5.1 based software development is completed using XCode 11.4 Integrated Development environment for macOS. As one reads through the paper, it is noticed that the use of the word simple is thrown out a variety of times; This was a priority during the development of the application, making a simple GUI that provides its user with a large amount of capability.

Application Front End (User Interface) - On the very first initial opening of the application, the user is asked to login using a unique Username and password. This is initially provided by OpenAire to a certain user, who will have authoritative rights from the start and will then be able to create and provide login credentials for application access. These login credentials are a

requirement, as application is inaccessible unless proper authentication is executed. Once the user successfully logs in, he or she will no longer need to go through the procedure again, as it is expected that this is the user's personal device, therefore will not need to log in every time. Refer to the database section of this document for more information regarding security and this topic.

Login Page - The design of this login page is simple. It will contain the OpenAire logo on top, with login and password text fields below the image respectively. At the bottom there is a "Sign in" button, as well as a "Need Help?" button below it. Application color design/combination is based on the sponsor's company colors, white, aqua blue, and grey, headlining the color combination. When the user taps on the Username or password text field, the individual will input the respective strings.

Once done, the "Sign in" button shall be pressed, which will trigger the click event, signaling for the algorithm to process the strings within the text fields. These strings is processed by the back-end design of the application, being passed to "login()" method, which will identify whether the credentials inputted lead to a successful login.

If proper credentials were passed, then "SettingsView" is displayed, else, the user is notified that login credentials were invalid and will have to redo process. If the "Need Help?" button is pressed, then click event will cause a "UIAlert" to appear, which is pop up that will notiy the user to contact Open Aire in order to provide him/her with the credentials needed.

If proper logging procedure is followed, in other words "login()" determined that credentials inputed are correct, the application takes the user into the "Settings View". It is important to note that this is the default main page, in which algorithm is programmed to display at initial application opening and/or login. This page, however, is not the only page that will make up the user interface.

At the bottom of the screen, there is a ribbon containing three different buttons: "Settings", "Controls", and "Metrics". The click event of each of these buttons takes the user to each of the respective pages, each one having unique functionality based on the capabilities of the application. Following this paragraph is a description of each of the pages within the user interface, including those accessed through the different buttons in the ribbon, as well as pages within these pages.

Settings Page - The "SettingsView" is displayed once user presses the "Settings" button, triggering its click event. The purpose of this part of the application is for the user have access to two features that the application provides. The first one is the capability of modifying the default setting set for the sensors. The user is able to modify how the microcontroller triggers the opening and closing of the roof that the A.I.R.E. system is connected to; this part of the application is further discussed later on in this document. The second capability is for administrators to modify users. This includes creating users as well as modifying a specific user's rights; these determine whether a user can modify settings, manipulate the roof structure through the use of the controls page, or have view only access of the "WeatherMetricsPage" for the collected metrics by the system's sensors.

These two capabilities are accessed through the click events of two buttons within the "SettingsView", each event effectively redirecting the user to whatever page the individual desired

based on the button he or she pressed. As the title of the buttons will describe, the “Roof Settings” button redirects the user to the “RoofSettingsView” while the “User Authorizations” button redirects the user to the “UserAuthorizationView”, with if conditionals behind the clicking event’s algorithm to determine whether the user has access to either of these two setting modes.

User Authorization Page - The “UserAuthorizationView” will display at the click of the “User Authorizations” button event with a conditional in the backend preventing access to this feature if the currently logged in user does not have the right. As previously said, only administrators, with power ‘1’, can access this page, getting full control of viewing the users within the database as well as being able to modify their names and power.

The layout is effectively a Table View that is populated by the list of users registered in the Firebase database. Each cell within the table view will consist of the employees first and last name, followed by their power. At the tap of any of these populated cells, the user is redirected to ‘User Modifications Page’ where he or she will have the ability to change the tapped cell’s name and power information. More about the User Authorizations Page is discussed in the section below.

User Modifications Page - The “UserModificationView” will display at the tap of a cell within the User Authorization Page Table View. The employee’s information is modified in this page. The layout consists of a Text Field and a segmented control divided into three segments, values 1 - 3, representing the employees’ power. These fields are populated by whatever employee was tapped on the previous page. Once the administrator modifies the user’s name or selects another segment than the one currently selected, he or she will have to tap the update button at the bottom of the View in order for the changes to take effect in the back-end side of things. Once the update goes through, a ‘UIAlert’ will pop up stating “Employee update complete, Changes should be reflected in User Table”, and the changes should effectively be reflected in the Firebase database as well as in the Table View once the user goes back to the User Authorization Page.

Roof Settings Page - The “RoofSettingsView” is the second settings page that a user with the proper rights has access to. As the title describes, this page gives the ability to change how the A.I.R.E. system functions. OpenAire often works on retractable roofs that are intended for recreational pools. For this reason, the team picked a set of default settings that dictate whether the roof is at an open or a closed state. If system detects a pressure lower than 29.54 inHg, ultraviolet radiation higher than 8 UV, a wind speed larger than 25 MPH, or physical drops of rain, A.I.RE.’s algorithm assumes bad weather and closes the roofs. The team understood that the project was not only going to be used over pools however, which is why this page was implemented.

The “RoofSettingsView” is laid out in a simple manner for administrators to have access to these key attributes mentioned and have the ability to modify them. Five “Text Fields” are laid out in a vertical manner, the “High Temperature”, the “Low Temperature”, and the “Atmospheric Pressure”, “UV”, and “Wind” Text Fields. These are objects that display editable text and sends an action message to a target object when return is tapped. Right next to them, in the same vertical manner, is a “Segmented Control” for each text field; these display multiple segments each of which functions as a discrete button. This discrete button with serve as an indicator to signal whether to open or close the roof at the value set in the Text Field. Below these five set of objects, is a rain pressure Segmented Control. Next to each of the segmented controls is the update button

for the attribute it is next to. If this button is pressed, the cellular device will send a string with the data encoded within it, data which the microcontroller is able to understand and modify default attributes based on what was set by the user in the application. This is done via Bluetooth, and more about these Bluetooth commands is discussed in that Bluetooth section.

The "Atmospheric Pressure" text field is will by default be set to the number 29.54 inHg; If the that value is increased, the user is effectively raising the threshold pressure at which the A.I.R.E. system will open or close, this being determined by the Segmented Control by its side, which can be changed between the two discrete options by just a tap. In other words, if the text field is set to 29.54 with the segmented control to open, the roof will open when the pressure is measured to be 29.54 inHg or greater.

"High Temperature", "Low Temperature", "UV" and "Wind" Text Fields and Segmented Controls will work the same, with the same logic of a discrete button to determine whether the roof opens or closes at the set threshold. The rain pressure Segmented Control standing alone at the button acts more like a switch, where if the rain sensor detects rain, the administrators have the ability to either cause the roof to open or close.

Controls Page - Going back to the main page of the application, the "ControlsView" gives the user the capability of opening or closing the roof. As mentioned, this page is reached upon pressing of the "Controls" button in the ribbon. The layout consists of one simple button labeleled 'Open/Close'. Upon just the tapping this button, the system will trigger the microcontroller to send a signal to the motors of the roof structure in order to either open or close it, based on whatever state it is currently in.

Weather Metrics Page - The "WetherMetricsPage" displays all the different weather metrics that the A.I.R.E. system is capable of measuring. This is based on all the sensors that were implemented in the electrical interface; Temperature, air pressure, humidity, UV intensity, and wind speed. Following the color theme that has been discussed throughout this paper, the page is split into two panels, one consisting of the data metrics that are measured from physical sensors placed for the vicinity, while the other coming from the weather source utilized for information in the area.

In each panel, Text labels containing strings serve as labels for each of the statistics, followed by a string that represent the actual data. The information for the first panel representing the physical sensors' data can be displayed using a table format. The second panel is formatted in a similar manner, with the string labels changing based on the information provided by the API. In other words, instead of certain metrics not provided by the weather API such as UV intensity, air pressure, etc., the panel displays precipitation chance data, temperature from weather source, etc. Refer to the weather API section of the document to see API functionality and algorithm procedure.

iOS Application Back End - For the purposes of this document, one will refer to the back-end as the applications interactions with the database as well as how the application functions in the back end . This includes overviews of the application's classes, how algorithms store the user data information and weather metrics in the database, as well as how it gathers information from the Weather API.

Bluetooth Communication (Application Side) - One of the main selling points of the A.I.R.E system is the ability to control an OpenAire retractable roof from a distance. Communication between standalone electronic devices has become a norm in the tech world, so much so that users have come to expect that wireless devices can and should gather/analyze data about whatever purpose the technologies have.

For this reason, the team knew from the start that our intelligent system needed to be Internet of Things based, with capabilities of controlling the retractable roof, as well as gather and show the data collected by the system's many sensors to the user. For this reason, Bluetooth capabilities are a requirement for this project. This section will cover how Bluetooth communication is achieved through the front-end side of this project, providing an overview of the frameworks being used to accomplish functionality, as well as walking the reader through the code of the iOS application that allows for the wireless magic to occur [11].

Bluetooth Framework - When deciding what programing language to use to develop the Aire Controller, one of the main considerations that was not mentioned in the Objective-C vs. Swift section, was available Bluetooth frameworks and libraries available in the two. Bluetooth communication is achievable through both, however, having access to Apple's "Core Bluetooth" framework using Swift was just another deciding factor in favor of Swift. The Core Bluetooth framework lets your iOS and Mac apps communicate with Bluetooth low energy devices.

The framework is an abstraction of the Bluetooth 4.0 specification for use with low energy devices. That said, it hides many of the low-level details of the specification from the developer, making it much easier for one to develop apps that interact with Bluetooth low energy devices. "Low energy" is an important term to be highlighted, as "Core Bluetooth" only deals with Bluetooth Low Energy (BLE) devices. The framework is effectively an API for Bluetooth 4.0, which is why having this was a requirement when selecting project parts that deal with the Bluetooth side of things.

Classic Bluetooth is not achievable through this framework, which for the purposes of this project was not wanted. Communication with classic Bluetooth devices such as wireless speakers can drain battery power at a fast rate, whereas BLE uses much less power, since it is used to communicate small amounts of data. That fact in it of itself made this form of Bluetooth communication the perfect fit for the A.I.R.E. system, as it is small amounts of data that is being communicated between the project and the cellular device.

BLE protocol is based on a client/server and consumer/producer model. In the overall system, there are situations where the iPhone and the microcontrollers interchange roles. The team used Apple's documentation to guide the programming based on this model, here are some excerpts that are worth inserting in this document. From the Core Bluetooth perspective, "The delegate of a CBPeripheral object must adopt the CBPeripheralDelegate protocol.

The delegate uses this protocol's methods to monitor the discovery, exploration, and interaction of a remote peripheral's services and properties. There are no required methods in this protocol." When it comes to the central, or the client, "The delegate of a CBPeripheral object must adopt the CBPeripheralDelegate protocol. The delegate uses this protocol's methods to monitor the discovery, exploration, and interaction of a remote peripheral's services and properties. There are

no required methods in this protocol." In the overall system, there are situations where the iPhone and the microcontroller interchange roles.

When it comes to receiving sensor data and displaying it to the user the iPhone is the client, or the device that needs data; the Server, or the device that has data, is the microcontroller. One could state the opposite when it comes to the iOS application sending commands in order to control the retractable roof. Because of this, the Bluetooth class is split into two parts, each covering one of the two cases. The proper parameters are passed into each class when initiating an instance of it.

In order to make use of the peripherals, one must connect to them by scanning for the advertising packets of data the peripheral is constantly broadcasting, a process called advertising. These packets are relatively small bundle of data that may contain useful information about what a peripheral has to offer, such as the peripheral's name and primary functionality.

The central must scan and find the peripheral, which once detected and connected to, stops advertising. The code is written to listen for and connect to the peripheral with the specific service being sought. This specific service is uniquely identified by the UUID of the microcontroller's service. The method called to accomplish this is 'centralManager?.scanForPeripherals(withServices: nil)'. This method effectively updates specific 'CBPeripheral' fields important to us when connecting.

It is important to note that before this, one must ensure that Bluetooth is on before scanning for peripherals, which is why conditionals were places prior to this function call in order to ensure that the Bluetooth is at proper status. If the CBCentralManager.state method returned anything but '.poweredOn' case, the 'self.bluetoothOffLabel.alpha' parameter was set to 1.0, which lets the algorithm know that it cannot scan for peripherals if not on a powered on state.

The next step in the process discovers what peripheral devices that phone can connect to, makes the proper referencing to the different important fields, and effectively connects. By proper referencing, what is meant is that one must store a reference to the peripheral in a class instance variable and adopt the ViewController to the CBPeripheralDelegate protocol.

Other steps such as stopping the scan are included in the algorithm to save battery life. After all these steps are completed, one connects to the device using the function 'centralManager?.connect()'. Once connected the algorithm looks for the services of the peripheral using the 'discoverServices' method followed by looking for the characteristic of interest within the service of interest, using the 'discoverCharacteristics' method.

After confirmation, one subscribes to the specific data wanted from the characteric of interest. From this one obtains binary values that are not readable to humans. In order to decoded, one must go to the GATT specification page for the subscribed characteristic; The first byte is metadata (Flags) about the rest of the data. With this information one is able to write the proper algorithm to obtain the right data. The code will constantly be looking to connect to the microcontroller, as the data is always changing. That in essence summarize the Bluetooth side of things when the iPhone is the client and the microcontroller is the server.

Class Design - The application's class design was organized based on the different pages discussed in the application front end section. In other words, each respective page is a class of its own, with the classes' event-based methods taking care of all the functionality of the application. The following section will detail each class that make up the application, covering how they interact with one another, the backend of the application, and other features that make up the Aire controller. Each class' methods and member variables will also be detailed in this section of this document.

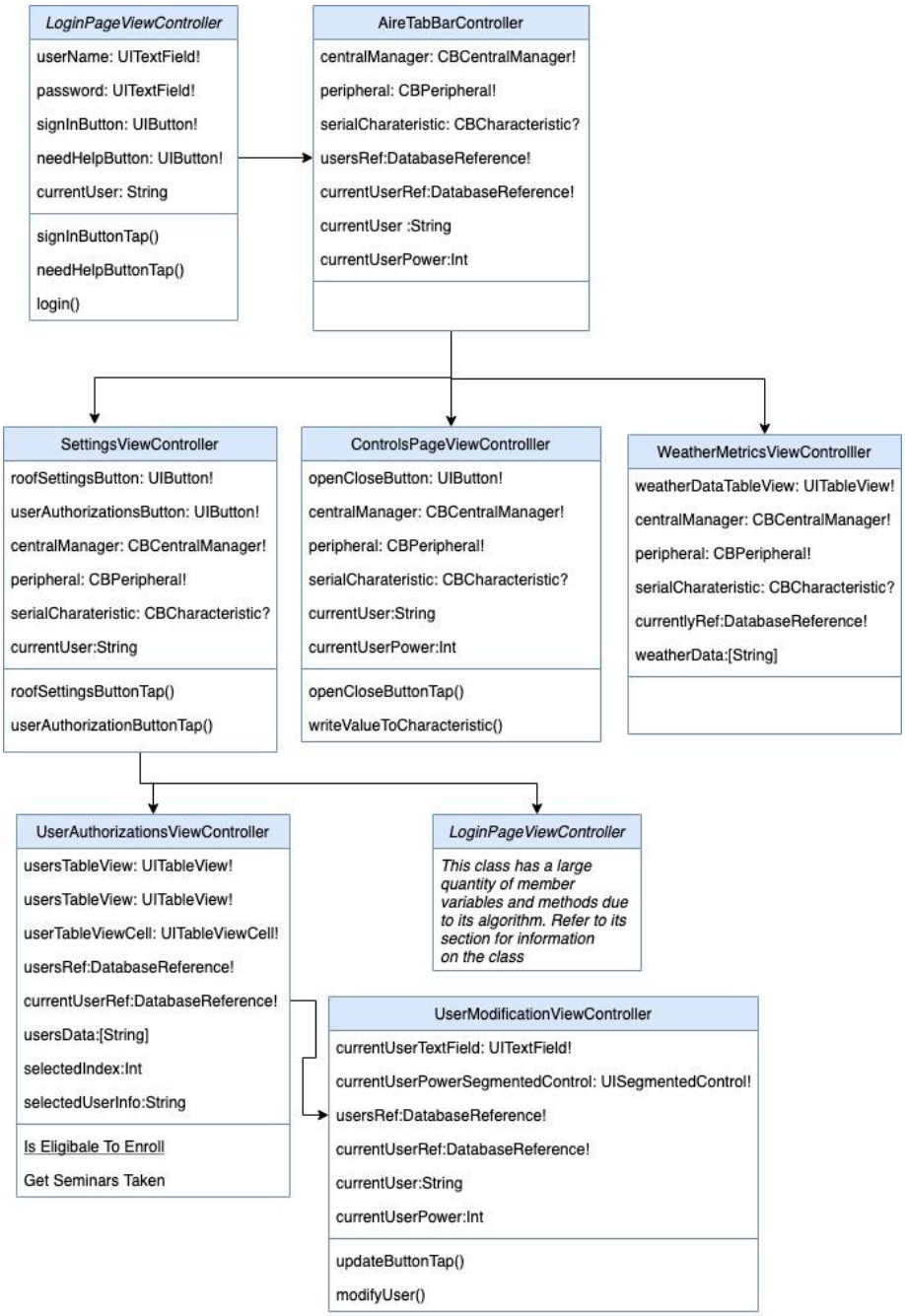


Figure 23: iOS Application Class Diagram

LoginPageViewController Class - The LoginPageViewController class is the simplest of the classes. This class is made up of four member variables and four methods. The four member variables is ‘userName’ and ‘password’, two ‘UITextField’ type variables that are initially set to be empty alongside two UIButtons type variables for the ‘signInButton’ and the ‘needHelpButton’. As their name states, the two are intended to hold the current user and their password, data that is set when login credentials are inputted by the user in the LoginPageView. Once the information is inputted and the Sign In button is pressed, this event will trigger the “SignInButton()”method, which simply calls the “login()” method, which is where the authentication algorithm takes place.

The login method, which was discussed in the in the Login Page section, directly interacts with the database and effectively analyzes the data within the UITextFields for proper security procedure. Essentially, the inputted Username and Password will go through a set of conditionals which will determine what the application does from there. The first conditional will check if the UITextField strings are empty. If this is the case, a UIAlert will pop up saying "Missing Username and/or Password Field(s)." If this is not the case, in other words the UITextFields are not empty, and else statement follows. This conditional calls the “Auth.auth().signIn()” method from the ‘FirebaseAuth’ library. This method effectively checks whether the username and password parameters passed to the function is part of the Users in the Firebase Authentication. These users are modified from the Firebase Project website itself. If the “Auth.auth().signIn()” returns ‘nil’ for its error parameter, authentication credentials were validated and the ‘performSeague()’ method for the "TabBarControllerSeagueIdentifier" is called, which leads to being redirected to the SettingsView, meaning the user is now able to use all the application’s features. If the credentials are not valid, a UIAlert pops up saying "Invalid Credential(s)", meaning the user was not able to successfully login and has to try again. Not only that, this seague also initializes the “AireTabBarController” class, which initializes the applications tab bar controller, responsible for a lot of the functionality implemented into the application. More is discussed on this class in its respective section.

The last method within the LoginPageViewController is the ‘needHelpButton’ function, which as the name describes, is invoked at the press of the ‘Need Access Button?’. This short method will simply be a UIAlert Call with the following message: “Contact Open Aire in order to be provided with account credentials for your company.”

AireTabBarController Class - The “AireTabBarController” class is responsible for three key things: Initializing the three pages that application is made up of, initializing Bluetooth communication with the microcontroller, and lastly pull the weather metrics data from the Firebase database. It was decided to do all these things in this class because this is the first thing loaded once login authentication is done successfully. Not only that, because the “AireTabBarController” has relationship segues with the three main pages of the application, data that is needed across every view controller can easily be transferred to one another because of this segue connection.

The class is of type ‘UITabBarController’, but also extends the CBPeripheralDelegate protocol as well as CBCentralManagerDelegate protocol. These protocols are necessary for Bluetooth Low Energy functionality and are explained in more detail in the Bluetooth section of these documents. There are four member variables: ‘centralManager’ and ‘peripheral’, which are a

‘CBCentralManager’ and ‘CBPeripheral’ types, representing ‘Core Bluetooth’ library properties needed for connection. ‘serialCharacteristic’ of ‘CBCharacteristic’ type serve as the Bluetooth characteristic that is used for communication. Again, these are explained in more detail in the Bluetooth section of this document. The last member variable is a string array called ‘postData’, which is the array used to store the weather data pulled from the DarkSkyApi.

All the methods within this class are used for Bluetooth discovery and connection, except for the “viewDidLoad()” method. This method is contained within every class and is executed as soon as the respective object associated with the class is initialized. One modified this method a bit in order to have a few things initialized from the start. The ‘CBCentralManager’ protocol is executed which starts of the Bluetooth procedure. After that, a reference to our Firebase database is made to the child node "kitID/DarkSkyApiData/currently". This exact node within the database holds the data that we want for later use within the application. It contains a variety of current weather information metrics; however, we just pull the apparent temperature, the temperature, humidity, wind speed, and chance of rain. To pull this date, we call the firebase function ‘observe’, which returns a dictionary type variable of all the nodes under the child node referenced (kitID/DarkSkyApiData/currently"). From this, we easy save the data we want into string variables, and then append them into the ‘postData’ array, which is later used for the Weather Metrics Page.

ControlsPageViewController Class - The ControlPageViewController class is the class in the application that allows the user to control the roof, in other words open or close it. The member variables within it are the ‘centralManager’ and ‘peripheral’, which are a ‘CBCentralManager’ and ‘CBPeripheral’ types, representing ‘Core Bluetooth’ library properties needed for connection. ‘serialCharacteristic’ of ‘CBCharacteristic’ type serve as the Bluetooth characteristic that is used for communication. These were all initialized at the AireTabBarController Class, therefore do not need go through the CBPeripheralDelegate or the CBCentralManagerDelegate protocols; a simple cross reference to the ‘AireTabBarController’ class suffices to obtain all Bluetooth capability in this part of the application. This is all done in the “viewDidLoad()” method. The last member variable is a ‘UIButton’ type variable, the ‘openCloseButton’, which represent the button that the name itself describes.

There are two methods, other than the ‘viewDidLoad()’ method that find themselves within this class, the ‘OpenCloseButtonPressed’ and the ‘writeValueToCharacteristic’. The first is called at the tap of the ‘openCloseButton’, and the first thing it does is check whether or not the current user can open or close the roof. This is done by checking the ‘currentUserPower’, which was obtained from the ‘AireTabBarController’ class; If the user has an authorization level equal to 3, then he or she cannot send any commands to the microcontroller. After that, the method goes through another conditional to check if the application is connected via Bluetooth; this is simply done by checking that the ‘serialCharacteristic’ variable does not equal ‘nil’. Once these conditionals are met, the method builds a string that represent the Bluetooth command for opening and closing the roof. More on this specific string command and how they are constructed is discussed in the “Bluetooth Commands” section. Once the string is constructed, the string is converted to a ‘UInt8’ array, which is just an array that holds the ascii value of each of the characters. A loop then iterates through the array of ASCII values, calling the

‘writeValueChaacteristic’ function at each iteration. This function requires parameters that include a ‘CBCharacteristic’ and ‘Data’ types; the characteristic passed is ‘serialCharateristic’ obtained from the ‘AireTabBarController’ and Data passed is the character’s ASCII value at the current iteration within the Bluetooth command string UInt8 array. The values are sent one by one, as the microcontroller’s algorithm is written to process each value one by one. Once in the ‘writeValueToCharacteric’ method a conditional checks whether the given characteristic contains the property, ‘.writeWithoutResponse’, which effective writes to the microcontroller. If so, then the ‘peripheral.writeValue’ method is called, which writes the current UInt8 value, in Hexadecimal, to the microcontroller.

WeatherMetricsViewController Class - The last major class at the backend of the application is the ‘’WeatherMetricsViewController’ class. As discussed, this page is responsible for displaying the data from ‘DarkSky API”. More specifically the apparent temperature actual temperature, humidity, wind speed, chance of rain, nearest storm, pressure, and UV index.

It is important to note that this class is not responsible for anything but displaying metrics; everything else regarding the specific thresholds that affect how the roof operates are modified within the firmware of the system or within the “Roof Settings Page” of the application. In order to do this, the class has to inherit from the “UITableViewDataSource”, and “UITableViewDelegate” classes, and must conform to the protocol functions, which is discussed in this section.

The variables are “weatherDataTableView: UITableView!”, “currentlyRef:DatabaseReference!” and an array of Strings “weatherData”. The variables are all initialized in the viewDidLoad() method. In order to populate a table view that displays the wanted data from Datk Sky API, one had to initialize the “currentlyRef” variable using the following Firebase library default method, “Database.database().reference().child("kitID/DarkSkyApiData/currently")”. What this method does is effective make the variable points to the exact Node that holds the data in the application’s database. Once the node is properly referenced, the Firebase observe method is called. What this method does is return a snapshot of all the data under the referenced node. This snapshot can be converted into a “dictionary” data structure, effectively copying all the data under the node into this “[String:Any]”, or array, type object. It is important to remember that this information is pulled from another algorithm hosted on Firebase which pulls the data from the API and posts it to the database every five minutes. Once this is done, one pulls the weather statistics mentioned in the first paragraph and appends them to the “weatherData” array of strings.

Once this array is populated, once can populated the Table View. As mentioned in the first paragraph in order for a View to have a table View, the class must conform to protocol. To conform to the “UITableViewDataSource”, and “UITableViewDelegate” class protocols, the same functions that one had to adhere to in the UserModificationsViewController class, one had to apply here as well. In these functions, one establishes properties for the Table View, including number of sections number of elements, and the data source for the metrics that are displayed. This is why an array of Strings was used to store the data from the database, in order to provide these protocol functions the needed information to have the wanted Table View.

SettingsViewController Class - The ‘SettingsViewController’ class serves as the backend to the Settings Page. The member variables for this page consist of the UIButtons ‘roofSettingButton’ and ‘userAuthorizationButton’, and the previously discussed Bluetooth Core variables centralManager, peripheral, and serialCharacteristic. As in the controlsView, these were referenced from the ‘AireTabBarController’ class in the ‘viewDidLoad()’ method. The current user, ‘currentUser’ and his current Power, ‘currentUserPower’ are also part of this class, refernced from the same way the Bluetooth variables are.

The class’ methods come down to two additional from the ‘viewDidLoad()’ method, ‘RoofSettingsButtonTap()’ and ‘UserAuthorizationButtonTap()’. As their name states, tapping each of these buttons will redirect you to either the Roof Settings View or the User Authorization View. Prior to the action seague for either of the two buttons being executed, a conditional has to be met; the ‘userCurrentPower’ variable that referenced when the View was loaded must equal one. If this is not the case a ‘UIAlert’ will pop up, stating that the current logged in user does have the authorization to navigate into the desired page. If the conditional is met, the user is redirected to the page corresponding to each of the buttons.

It is important to note that prior to the next View appearing, the necessary data for each page is transferred over through the prepare function. This function is a function that is invoked in the background when an action seague is executed. Swift provides the developer the ability to modify this function, which allow for variables to be set for the page which the seague redirects the user to. For when being redirect it to the Roof Setting page, the Bluetooth Core vairables are initiated in the prepare function, as Bluetooth functionality is needed on that page. The User Authorizations page does not require any data that was initiated in another View.

RoofSettingsViewController Class - The Roof Setting View controller class consists of a total of twenty member variables. Because of the large quantity, each of them will not be listed in this document. The reason for the large amount of member variables is because they all represent the same three objects within the View, a Text Field, a Segmented Control, and am Update Button. Each of these pertains to a specific weather threshold that the user can modify. These are, low temperature, high temperature, pressure, wind speed, UV Index, and Rain.

Effectively the text field represents the value for which the roof ishave a certain way, the segmented control is split into an open and close segment, and then the update button is what actually sends the information to the microcontroller via Bluetooth for that particular metric. For that reason, the Bluetooth variables that have been mentioned throughout this Software Design section are also part of this class. Recall that they were initialized in the prepare function in the Settings View page.

When it comes to the methods, there are ten methods additional from the viewDidLoad() method. Nine of them are the methods called upon the tapping event of their respective button while the tenth is the ‘writeValueToCharacteristic’ method discussed in the ‘ControlsViewController Class’ section. Each of the update buttons will work like the “Open/Close” button in the Controls View; after the conditional of Bluetooth connectivity is met, then the Bluetooth command string is made, followed by the conversion into the array holding each of the character’s ASCII values in ‘UInt8’ format, and finally iterating through the array and calling the ‘writeValueToCharacteristic’ to send

each individual character ASCII value one by one. It is worth mentioned that the Bluetooth command string is made from the combination of the data and the open or close segment chosen by the user.

UserAuthorizationsViewController Class - This class has a similar layout to the WeatherMetrics View Controller class, therefore is structured in a similar manner. Since the class represents the User Authorization page, the users and their powers are all represented in a Table View. The variables for this class are “usersTableView: UITableView!”, “usersRef:DatabaseReference!”, and a userData String array. These variables are all initialized in the ‘viewDidLoad()’ method, where using ‘Database.database().reference().child("kitID/Users/")’, the usersRef variable is initialized. As explained in the WeatherMetricsViewController Class section, this method returns the Child node holding the data within the database desired, in this case the list of all the employees registered in the database.

Once this is obtained, the observe method is called, obtaining the snapshot of all the data under the referenced node. This snapshot is then converted into a “dictionary” data structure, effectively copying all the data under the node into this “[String:Any]”, or array, type object. Then the algorithm iterates through all the elements in the dictionary and appends them to the ‘userData’ array. Similar to the WeatherMetricsViewController Class, this String array makes it easy to provide the protocol functions with the needed information to properly make the Table View.

As mentioned, this class is structured similarly to the WeatherMeticsViewCotroller class, however, it has an additional method. This function is invoked upon the tap of any of the cells within the Table View, and what it does is redirects the user into the UserModifcation View, where the administrator is able to modify employee information. In order to do this, all the function has to do is call the action seague to be redirected to the said page. It is worth noting that upon the call of this seague, its prepare function, similar to the SettingsViewController Class was modified to initialize the ‘currentUser’ variable, as well his/her ‘currentPower’, which are pulled from the selected cells string. More on the ‘userModificationController’ class is below.

UserModificationViewController Class - This class represent the UserModificationView. The member variables are: ‘DatabaseReference!’ types usersRef & currentUserRef, a ‘currentUser’ String and ‘currentUserPower’ Int, and finally a ‘currentUserTextField’, which as the name says ‘UITextField’, and a ‘currentUserPowerSegmentedControl’ Segmented Control. Their names are self-explanatory as to their function. Upon initialization of the class, or ‘viewDidLoad()’, the text Text Field and Segmented Control are initialized to whatever employee and respective power was passed in the UserAuthorizationsViewController class.

The user is able to modify the UITextField as well as the segmented control within the page. Once the user modifies and he or she hits the update button, that is when the classes methods come into play. The first method is the ‘updateButtonTap’ which begins by initializing the usersRef variable through the same ‘Database.database().reference().child("kitID/Users/")’ method. Similar to previous explanations, the observe method is then called, and its snapshot is initialized as a dictionary.

Once this dictionary is initialized, the algorithm iterates through each child node under the referenced node from the Firebase database, in other words all the employees. At each iteration there is a conditional checking whether the employee's information at the current node matches that of the 'currentUser'. If so, the 'modifyUser()' method is called, passing the proper user information parameters. Essentially what the algorithm is doing is iterating through all employees in the database until it matches the employee currently being modified.

The 'modifyUser()' method begins by initializing the 'currentUserRef' variable using the same method; 'currentUserRef = Database.database().reference().child("kitID/Users/\(user)")'. The user String variable is the one passed from the iteration in the updateButtonTap method. This function then simply calls the '.setValue' method, which adds a new child node to the currently referenced node within the Firebase Database. The information within this is the currentUserTextField String as it is currently typed, as well as the selected power within the segmented control. Once the user is added, the old child node that contained the employees information as it previously stood is removed using the Firebase method '.removeValue'. After all the steps are completed, all variables are updated, and a 'UIAleart' is called saying, "Employee Update Complete", message: "Changes should be reflected in User Table". The data should be indeed updated in the table within the User Authorization View, as the table is reloaded every time that view is loaded.

5.2.2 Firmware Algorithms

The main purpose of the microcontroller in this project is communication. Communication between the six sensors and the roof, sending requests for measurements and interpreting the received data, and sending a signal to trigger a roof event if necessary. Communication with the database and API servers for information about users and predictive weather. Communication with the user to set up a new unit or update previous settings. This section will decompose and analyze each of these objectives and outline how the software development team plans on structuring the microcontroller software.

Firmware Interrupts - Along with the logic flow algorithms that the new sensor measurements are traversed through, the other main aspect of the firmware code is the three interrupts that actually drive the microcontroller. The three specific interrupts are a timer interrupt, UART receive complete interrupt, and external interrupt. These interrupts drive the specific actions the microcontroller needs to take when an event is triggered or needs to be triggered.

The specific timer interrupt that is used is the Timer/Counter 3 Compare Match A interrupt. Timer 3 was chosen to be the main project timer due to it's precision being one of the sixteen bit timers, and due to it's interrupt vector location. The Timer 3 interrupt vector falls below the UART receive complete interrupt, and we need that interrupt to not be hindered for any reason. The Compare Match A part of the interrupt is due to the specific timer mode selected for the project. The timer counter increments until it reaches the value in counter register A, which in this case is the value to create a four second interrupt, and then executes the timer interrupt. Using this mode allows for higher precision and control of the timer counter.

The Timer 3 interrupt service routine is executed seven times, incrementing a Timer 3 global counter, to achieve an overall delay of thirty second. On the eight execution of the interrupt service routine the function for polling all the sensors for new measurements is called and the Timer 3 global counter is set to zero.

The second interrupt that is utilized for this project is the UART0 Receive Complete interrupt. This specific interrupt is to handle any new bluetooth communications between the iOS application and the microcontroller. This interrupt vector resides high on the interrupt table, which is important for this project, as the bluetooth commands sent to the microcontroller need to be handled first and foremost. These commands can change the roof status or can change the thresholds of the sensor measurements, which if not handled could cause the roof to malfunction when it comes to the desired behaviors.

The UART0 interrupt service routine immediately reads the data in the UART Data Register 0 for the command that has been sent. The command is then compared to the hardcoded command values, which describes the specific values and bluetooth command algorithm, and calls the respective functions for changing thresholds or behaviors, or changing the roof status.

The last major interrupt in this project is the external interrupt for the wind sensor. The specific interrupt is External Interrupt Request 4, this is due to the location of the anemometer pin, connected to the microcontroller on pin six. The pin is triggered to execute an interrupt on the falling edge of the input signal.

The External Interrupt Request interrupt service routine handles the polling and measurement of the anemometer. When the interrupt service routine is executed it increments the number of rotations of the anemometer in the time between the current, and last interrupt. After four interrupts have been executed a sample is required. The sample for the wind speed is calculated by using the following equation, where T is the amount of time since the first interrupt until the last:

$$\text{Velocity} = \text{Rotations} * (2.25 / T).$$

Firmware Setup - Peripherals- When the microcontroller powers on, it runs through the set up code for the project once. This code includes setting up the microcontroller peripherals and initializing any sensors. The microcontroller peripherals that are used for this project are the Universal Asynchronous Receiver/Transmitter(UART), Pulse Width Modulation (PWM), Analog to Digital Conversion (ADC), and Two Wire Serial (I2C). All the peripherals need to be initialized in some way at the power on of the microcontroller.

The UART for this project is initialized to use the UART0 channel. The baud rate must be defined and used during the initialization. RTo keep with the low power consumption and to have better accuracy, according to the ATmega datasheet a baud rate of 4800 or 9600 would suffice. The team selected to use a baud rate of 4800, and to convert the baud rate into the required sixteen-bit baud rate register, UBRR, the following equation is utilized from the datasheet:

$$UBRR = \text{Clock Frequency} / (16 * \text{Baud}) - 1.$$

The UART0 is structured to be eight bits, with no parity bit, one stop bit, and run in asynchronous mode. The UART0 is able to both transmit and receive data. Most importantly, the receive complete interrupt bit must be set in the UART register during initialization to allow for the interrupt service request to execute.

The PWM is used to create the signal to the actuators to move the roof when an event is triggered. The PWM is quite simple to initialize, as the duty cycle for the PWM is 100%, since the whole 3.3V is needed to be outputted to the relays. When a PWM is required, the pins are written to be high until the roof is completely open or closed. To initialize the PWM, the specific pins need to be written to be output pins. This is done by the single line of code, by setting pin 5 in port G to be 1 (DDRG |= (1<< DDG5)).

ADC peripheral for this project is used to monitor the analog voltage signal of multiple sensors, which is then used to calculate the respective measurements for those sensors. The ADC is initialized by setting the pins as input for the analog signals, setting the register values for the ADC, and starting the first conversion. The first conversion takes place at the power on of the microcontroller because it takes an abnormally longer period of time to complete. It takes about twenty-five ADC clock cycles to complete, when on average a normal conversion takes about twelve clock cycles. In order to eliminate any delay this may cause, the first conversion is started when the ADC is initialized. The pins for the ADC are set to input by writing all the pins in port D to be 0. For this project the ADC is initialized to be running in free mode, left adjusted, and use the internal 3.3V as reference. ADC is left adjusted to cause the eight most significant bits of the conversion to be in the ADCH register. This project does not require that the precision of the measurement be any better than eight bits. This also makes reading the ADC result much easier and faster.

The final peripheral is the I2C communication that needs to be set up in order to communicate with different sensors to command for measurements and readings. The I2C clock is set to 100kHz to meet the clock frequency needs of all the sensors using the two wire communication.

During the setup and initialization of the peripherals the global interrupts are disabled to ensure that none of the initializations throw an interrupt that the microcontroller is not prepared to handle yet. At the end of the setup, the global interrupts are re-enabled to allow for the peripherals to execute the proper interrupt service request if applicable.

Sensors Firmware- When the timer interrupt request service calls the function for polling all the sensors for new measurements. The polling consists of reading a new ADC conversion or commanding a new measurement to be taken and then read over I2C. I2C is set in master by sending the START command first over the I2C wire. The following section breaks down the polling process for each sensor for the specific peripheral it goes through.

The humidity sensor communicates with the microcontroller via I2C. To request a new measurement, the microcontroller sends the START signal. The START signal is followed by the slave address 0x88 (in write mode). The command medium repeatability for measurements, 0x24 and 0x0B, is sent followed by a STOP from the microcontroller. Once the measurement is completed the microcontroller sends another START signal followed by the slave address 0x89 (in read mode). The sensor then responds with two sixteen bit measurements. The first being the

temperature measurement followed by the humidity measurement. The measurements are sent in two eight bit segments. Each segment must be followed by an ACK from the microcontroller. Once both sixteen bit measurements are received a NACK and a STOP signal are sent from the microcontroller. The sixteen bit humidity value is divided by 65535 and multiplied by 100 to calculate the relative humidity percentage.

The barometric pressure sensor follows a very similar structure for polling new measurements, as it also communicates via I2C. The slave address for the barometric pressure sensor is dependent on the SDO pin being connected to ground or voltage. For this project it is connected to Vcc, resulting in a slave address of 0x77. When it write mode the slave address becomes 0xEE, and in read mode 0xEF. The barometric pressure sensor however needs to be initialized before the first measurement. A START signal is sent by the microcontroller, followed by the slave address 0xEE. The control register 0xF4 needs to be written to 0x05 to enter the single measurement trigger mode. This allows a new measurement to be taken whenever 0x01 is written to the control register. The calibration registers, which are specific for each sensor, must also be read and stored for calculating the pressure value. The calibration registers reside at 0x88 to 0xA1. To read in the calibration registers and calculate the barometric pressure value, the manufacture, Bosch Sensortec, library is used for efficiency.

The sensor measures both temperature and barometric pressure. The temperature measurement is needed for calculating the pressure value. They are read in similar fashion as the sixteen bit measurements from the humidity sensor, but these measurements are actually twenty bits. But the last four are not used in this implementation because that level or precision is unnecessary. Once the measurements are read they are used in the Bosch Sensortec library functions to calculate the barometric pressure value in Pascals.

To poll the light sensor the VEML6070 library provided by the manufacturer is used. The basis of the library is off of I2C communication. The library provides reading and writing functions, along with all necessary addresses for the I2C protocol. Another helpful part of the library is the hardcoded UV level index it includes. The library function returns a single integer that depicts the UV index risk level.

The rain and temperature measurements are obtained via ADC. Both sensors use the same ADC functions. To start a conversion the ADC channel for the respective sensor is stored in the ADMUX register and the ADSC bit in the ADC status register is written to 1. The ADC channels for temperature and rain are channel zero and channel one respectively. Once the conversion is completed the eight bit result is read from the ADCH register.

For the rain sensor, to calculate the value of the input voltage the following equation is used: Voltage = ADC * 3.3 / 256. The closer the input voltage is to 1.0 volts, the more water there is on the sensor. The closer the input voltage is to 3.0 volts, the drier the sensor is. Through testing it was found that 2.4 volts was an unacceptable voltage threshold for determining if it was raining enough to trigger a “raining” event.

To calculate the temperature from the input voltage two equations are utilized. The values are based on the implemented circuit, for this project the implemented circuit results in the following constants: **β**= 3998, Balance resistor (Rb) = 10 kOhms, and Room Temperature (To) is 298.15

Kelvins. With those constants the ADC value is used in the following equation to calculate the resistance at the current temperature: R = Rb * (1023 / ADC - 1). With the calculated R value, the following equation gives the measured temperature (T):

$$1/T = 1/To + (1/\boldsymbol{\beta}) * \ln(R/Rb).$$

All the values that are returned from the polling process are stored into a measurement structure that is used later to check against the threshold values to determine if inclement weather is present. These values are also used to average the sensor measurement since the last data push to the iOS application.

Sensor Default Limit Values - Six different sensors are used within this project to provide current conditions of the structure's surroundings. Some of the sensors alone can give an accurate interpretation of the weather, while others need to be paired with data from another sensor to be able to be implicit. A few of the sensors have multiple purposes, providing extra measurements. Not all of the sensors have predefined extremes for the conditions that they measure. Some of the sensors conditions and limits is defined by the user based off of location or purpose of the structure.

Wind has been studied enough, both on water and on land, to have a set of wind speed ranges to define a "windy" condition. These values come from the Beaufort Wind Force Scale which has been used for centuries and was last updated in 1955 [12][1]. Beaufort Scale 6, defining wind speeds between 25-31 miles per hour, is the range the team has decided to use to define a "windy" condition.

During this kind of wind, large tree branches isgin to wave and whistling occurs in electric and telephone wires. The default value for windy is defined within the software to be 28 miles per hour. This wind speed will need to be recorded at least three times in a ninety second, or hold for 15 seconds to trigger a change in the roof's position.

With the selected rain sensor, precipitation and the intensity of precipitation, is measured. The sensor allows for a sixty second delay between droplet detections to accurately decipher between actual rain or mist-like conditions. No default values are needed for a raining condition, an analog signal is returned with the intensity in the format of a dynamic voltage.

Ultraviolet radiation is another weather measurement that has abundant research that can provide the project with predefined default limits. UV has a predefined index from the Environmental Protection Agency (EPA), which is based off the guidelines for UV Index reporting from the. World Health Organization. The UV Index from the EPA is broken into three groups: Low, Moderate to High, Very High to Extreme. For the purposes of the project, the third group, Very High to Extreme, is analyzed. A UV Index of 10 is a preferable limit to set as the default for the system. At a UV Index of 11, skin damage can occur in as little as fifteen minutes [7][2].

Humidity and Temperature go hand in hand to create the Heat Index. The Heat Index indicates how hot it really feels. Using measurements from both sensors the system can calculate the Heat Index, The National Weather Service has a Heat Index table that is based off of air temperature and relative humidity. The Heat Index is calculated from an equation that has been refined from

the research of Lans P. Rothfusz in 1990 [8]. The equation, which is used to calculate Heat Index for this project is:

$$\text{Heat Index} = 42.379 + 2.04901523 \cdot T + 10.1433127 \cdot RH - 0.22475541 \cdot T \cdot RH - 0.00683783 \cdot T^2 - 0.05481717 \cdot RH^2 + 0.0012287 \cdot T^2 \cdot RH \cdot 0.00085282 \cdot T \cdot RH^2 - 0.00000199 \cdot T^2 \cdot RH^2$$

Dangerous levels, where the likelihood of heat disorders such as heat stroke and dehydration, occur when the Heat Index reaches about 105-108°F with prolonged exposure. Extremely dangerous levels occurs when the Heat Index reached around 120°F. The default value for Heat Index is 110°F, as that is the middle range for dangerous levels.

While Heat Index can be calculated, and certain values is set as defaults for the system, alone temperature and humidity is mainly based off of user's preference. The preference can change based on the structure's purpose. Default values for the two measurements will still be implemented, but users is prompted to change them during setup. Unlike other measurements however, temperature will have both maximum and minimum limits. The default maximum value for temperature is 100°F, as prolonged exposure at this temperature is likely to cause heat disorders. The default minimum value for temperature is 32°F, as that is the freezing point for water. For humidity, the default value is 82% RH.

The maximum relative humidity for Tampa, Florida was 78.4% [13][4], so 82% was selected as the default to add in a small buffer for sensor measurement error. However, at different temperatures, relative humidity can vary greatly. At lower temperatures, relative humidity can spike into the 90th percentile [14][5]. Whenever the relative humidity limit is reached, the temperature needs to be measured and referenced. If the limit is reached, and the temperature is above 80°F, then the roof trigger event shall occur.

Barometric pressure is a good indication of what weather conditions are fast approaching. Usually rising barometric pressure is a sign of good weather, while decreasing pressure indicates a storm is approaching. Barometric pressure changes due to altitude, for this project it is assumed that the default is at sea level, meaning altitude is zero.

At sea level barometric pressure is defined to be 101.325 kilopascals. To calculate a new barometric pressure, in the event that a system is not at sea level, the following equation is used to calculate the new pressure in Pascals, where Height is the difference from sea level [15][6]:

$$\text{Pressure} = 101325 \cdot [1 + (-2.256 \times 10^{-5}) \cdot \mathit{Height}]^{5.256}$$

With the base barometric pressure, any changes is measured, and the previous state of the pressure is referenced. Unfortunately, pressure changes are very minute and can take a longer time to occur, because of this, the barometric pressure readings is used along with other sensors and information from the Weather API to trigger a roof event.

The following Table 40 reviews the different conditions and their respective defined default values that is hardcoded for the initial installation of the system. During setup, the user is prompted to change these values, if they so desire. These values can also be changed throughout the life cycle of the system.

Table 40: Defined default values for the system

Measurement/ Condition	Value	Unit
Windy	28	Miles per Hour
Raining	NA	NA
Temperature (Maximum)	100	Degrees Fahrenheit
Temperature (Maximum)	32	Degrees Fahrenheit
Relative Humidity	82	%
Heat Index	110	Degrees Fahrenheit
Barometric Pressure	101.325	KiloPascals
UV	11	UV Index

Roof Default Behaviors - The roof of the project will also have default actions that is hard coded in for the initial installation of the system. These default actions is in response to the measurements/conditions the sensors report, along with the data that is pulled in from the Weather API. Just like the sensor defaults, the roof defaults can be changed during set-up and at any time during the life cycle of the system. Two different set of defaults is outlined, one for a pool structure, and one for a garden/greenhouse structure. This is to demonstrate the different capabilities the roof will have with both the sensors and the Weather API.

These values are examples of possible roof behavior and would change on user preference. They are also based off of the default values for the sensors found in the previous section. For a structure with a pool, any kind of hazardous weather conditions such as rain, strong winds, and damaging UV the roof should be closed to protect patrons. A pool structure may not care about maximum temperatures but would likely care when temperatures drop and reach freezing. A greenhouse structure on the other hand may want more rainfall to be allowed within the structure to decrease the use of watering systems. These examples show the vast options that the roof can react to and what possibilities that the structures can have.

If the measurement or condition changes for the listed aspects, the current state of the roof would be examined. If the state is the opposite of the defined behavior value, an event would be triggered to power on the roof mechanics to change the state of the roof to match the defined behavior.

Table 41: Roof Default Behaviors

Measurement/Condition Changes	Roof Behavior for Pool Structure	Roof Behavior for Greenhouse Structure
Windy	Close	Close
Raining	Close	Open
Temperature (Maximum)	Open	Open
Temperature (Minimum)	Close	Close
Relative Humidity	Close	Open
Heat Index	Close	Close
Barometric Pressure	Close	Open
UV	Close	Open

Whenever the roof is in the open state, the central air conditioning or any kind of climate control should be turned off. When a roof event is triggered, the new state should signal to the climate control systems to be powered on or powered off. This saves power and efficiency for the structure.

Software Algorithm Flowchart - To better encapsulate the connection between the sensors and the behavior of the roof, the following flowchart in Figure 23 depicts the basic logic behind the software algorithms for the microcontroller.

The diagram shows how the data received from the sensors is put through different conditionals to determine the current state of the weather. The arrows indicate the directions of the flow chart, and green arrow indicates a ‘TRUE’ for the conditional as the red arrow indicates a ‘FALSE’ for the conditional. If the data received from the sensors indicates inclement weather, the current status of the roof, whether it is open or closed is checked, along with the defined behavior of the roof for that weather. If the two do not match, then the microcontroller will send a signal to either close or open the roof based on the defined behavior.

The flow chart also depicts how different sensors interact with one another in order to make accurate measurements on the current weather. The temperature and humidity sensors interact together, while also being used separately for their own measurements. The barometric pressure

sensor also interacts with the Weather API to check predictions for the forecasted weather of the area.

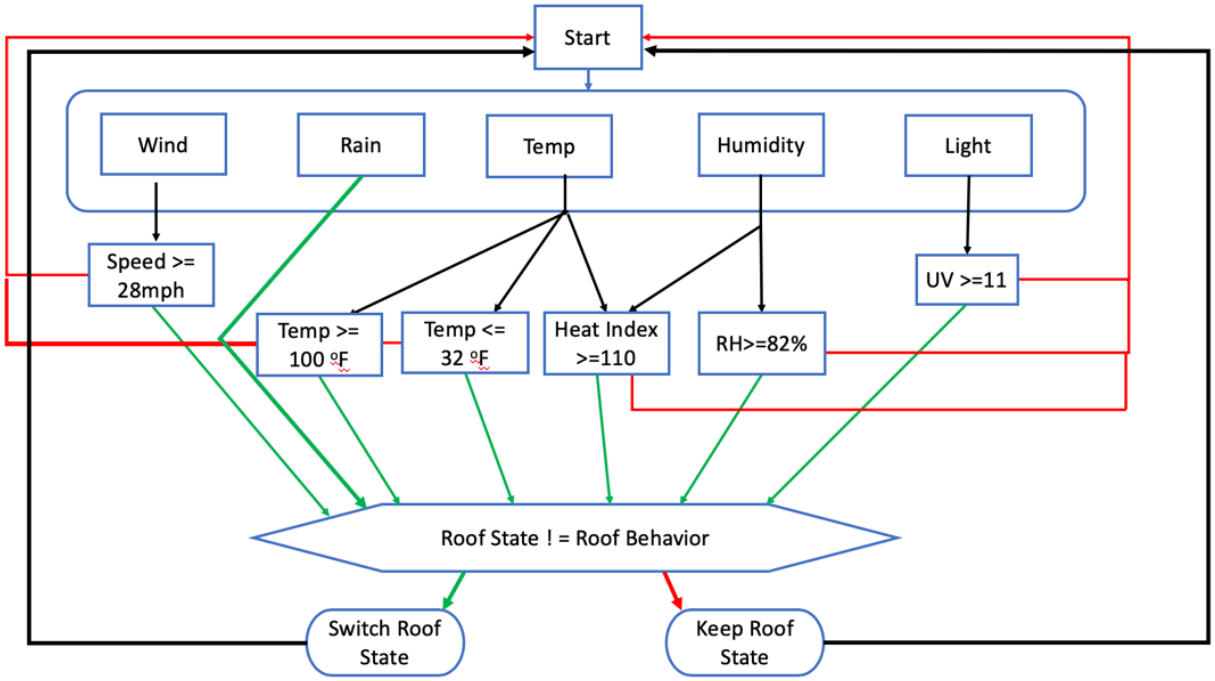


Figure 24: Software Flow Chart for Sensor and Roof Connection

Additional Light Sensor Algorithm - The other purpose of the light sensor in this project is to measure the amount of natural sunlight the structure is receiving. Based off this measurement, the lighting inside the structure is responsive, either increasing in brightness or decreasing. This will save power consumption for the overall lighting of the building. When there is ample sunlight, the interior lights will remain off or dimly lit, and similarly when it is dark outside the interior lights is brighter. This also serves as a good implementation for structures that are green houses. When the weather is cloudy and there is not much light for plants, internal lights is turned on to combat the lack of sunlight. To implement this design, the light sensor's measurements of lux and UV light is used.

In order to determine which light levels the visible light sensor and UV light sensor should signal a change, the type of environment inside of the enclosure must be determined. The average amount of full daylight is around 10,752 lux according the National Optical Astronomy Observation (NOAO) [16][7]. The NOAO has recommendations for the amount of illumination that should be present for different activities. These lighting recommendations are important as they feed into the phycological enjoyment of an activity. Keeping a room too dark or too light can make a situation very uncomfortable even though it might not seem like a big aspect at first glance.

The *Table 42* below extracts some of those recommendations that are relevant to this project.

Table 42: Illumination Recommendations from NOAO

Activity/Space	Illumination (in lux)
Dinning Areas	150-200
Lobbies/Atrium	200
Physical Fitness Space	500
Outleased Space	500
Supermarket	750
Detailed drawing or mechanical work	3000

These values could serve as possible predefined default values, based on the type of structure the user selects during setup. If the illumination from outside does not meet the value for the activity or space, the microcontroller would signal the interior lights to turn on or turn on to a certain brightness.

The following flowchart shown below in Figure 24 shows the behavior for light sensing algorithms. Using the recommended illuminations provided above in *Table 42* The following decision flowchart was made. Once the program starts and the system powers on, the light sensor reads in lux data, and checks if it is greater or less than the default lux value. If it is less than, the lights inside will turn on or get brighter if they are already on. If the measured value is greater than the default, the lights inside the enclosure will turn off as to not waste energy.

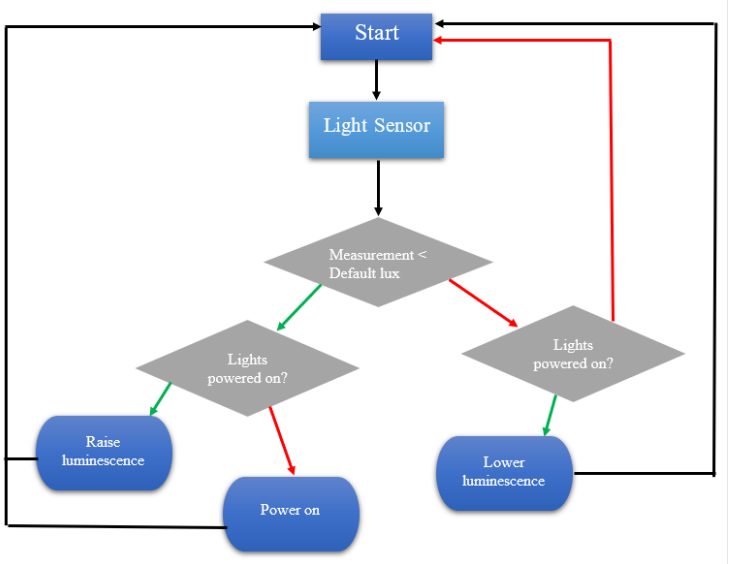


Figure 25: Light Sensing Algorithm

Database Design - The database is used to offload larger data from the microcontroller to increase storage space and runtime performance. The two main purposes of the database is storing user settings and logins, and data for the characteristics and statistics from the API and sensors for the weather. The data from the API is stored for analysis and statistics for twenty-four hours before being overwritten. This will decrease the amount of storage needed for the database and decrease time for calculations and data retrieval.

For implementing the user settings and login storage, three tables is created. The first table, titled 'Users', will hold the user identification number, system number, and hashed password. The user identification number is the primary key for the table and is an auto incremented number. The system number is the hardcoded identification number for the kit that is on the microcontroller. This can be thought of almost as a username, as each microcontroller will have a unique identification number itself. The hashed password is stored and verified on the database side to protect the user's identification and data. The system number and hashed password is created and stored during the initial setup of the system.

To increase security, but also increase the usability of the system the table titled Permission will hold user identification number, approved, and access. The user identification is pulled from the Users table and serves as the primary key of the table. Access is an integer and is used to distinguish between which kind of user is interacting with the roof. The different kind of users for the system is administrator and employee. Administrators is allowed to change measurement and setting preferences, behaviors, and schedules of the roof. Administrators will also have all the capabilities of an employee user. Employees will have access to control the lighting, movement of the roof, and view weather statistics. An administrator must approve new employee users before employee users can login, and that is managed by the approved Boolean column.

The multiple settings for the system, is stored in a table titled Preferences which will hold the user identification number, schedule identification number, behaviors identification number, temperature maximum, temperature minimum, relative humidity, UV Index, heat index, light, and structure type. The user identification is a primary key, and is selected from the Users table. The scheduled identification number is a foreign key referencing the Schedule table. The behaviors identification number is a foreign key referencing the Behaviors table. Structure type is the type of structure the system is being used for, i.e. garden house or physical fitness space.

The rest of the values for the table are the user's preferred threshold values. These measurement threshold values along with the expected behaviors is collected during the initial setup of the system. If the user makes no change to the threshold values, the default values outlined in section XX, is stored. By collecting the what type of structure is being used, the default values for the expected roof behaviors can be changed.

The different expected behaviors of the roof based off user preferences is stored in a table titled Behaviors, which will hold behaviors identification number, on wind, on rain, on UV, on temperature max, on temperature min, on relative humidity, on heat index, and on API. Behaviors identification number is an auto incremented primary key for the table, and is the value the Preferences table will use for reference. The remaining values is the user's expect roof behavior

for the trigger on that particular measurement. These values is collected during the initial setup of the system, and if unchanged, is the default behaviors as aforementioned.

An optional schedule can be put in place for the opening and closing of the roof. The Schedule table will hold time one, behavior one, time two, behavior two. Time one is the first instance that the roof is expected to trigger a behavior, and that corresponding behavior is behavior one. While it is expected that at the second time instance the roof behavior is the opposite, the user will still be able to schedule a second time instance and expected behavior which is stored in time two and behavior two. There is no default schedule, so schedules is created as the user requests.

The weather data is broken into API and sensor tables. The data from the API is stored for analysis and statistics for twenty-four hours from its original poll request, before being overwritten. This will decrease the amount of storage needed for the database and decrease time for calculations and data retrieval. The data from the API is used for predictions and analysis, so two tables is used to break those purposes down. The sensor data is stored for thirty-six hours before being overwritten, since the sensors are a more accurate measurement of the weather at the actual structure the data is stored for a longer duration.

Storing sensor data is done in the Sensors Collection table, which will hold the system identification number, average temperature, average relative humidity, average barometric pressure, average wind speed, average precipitation, average light, average UV, average heat index, and the maximum and minimum value for each of those measurements.

To reduce the amount of data being collected and the number of pushes to the database only the average, maximum, and minimum value for each measurement is recorded. The average is calculated using the current measurement with the stored current average. The values for averages is reset at every 0:00 of the day. But the data in the database is stored for thirty-six hours for analysis and statistics that is pushed to the phone application.

The first table for storing data from the API is API Predictions, which will hold weather alerts, precipitation rate, precipitation type, nearest storm, nearest storm direction, lighting strike distance, wind speed, and wind direction. This data is updated whenever a new poll request to the API returns. This table is expected to be the most queried and updated. The second table for storing data from the API is the API Collections table which will hold the same values as API predictions, but they is averaged throughout the day and is reset at 0:00 every day.

Google Firebase and API Polling Algorithms - Another reason the team decided to utilize Google Firebase as the database of choice for the project is due to the fact that it supports cloud functions. Cloud functions allow for background code to execute based on a trigger from the Firebase database. It is typically JavaScript or TypeScript, but for this project the software team decided to write the cloud functions in JavaScript. To create functions, the Firebase CLI commands are utilized. These commands allow for easy management of the cloud functions for the Firebase UI. For the scope of this project, using cloud functions is free for the first twelve months which is more than enough time for the completion of this project.

The Weather API polling is completed in the background of the database and mobile application by the cloud functions. The triggering event will either be new weather data being stored in the

database from the iOS application or an elapsed time since the last API poll. Timing for polls can be found in the Weather API selection section. Both Weather APIs that were selected is polled and their data stored in their respective database tables.

Moving the Weather API polls to the cloud functions has multiple benefits. It allows for constant API polling, no matter if the mobile device with the iOS application is connected to Wifi or data services. The data can be aggregated by another cloud function, and old data can be deleted from the firebase with yet another function. This keeps the weather data up to date and helps keep the total data stored within the database smaller. Using the cloud functions also offloads extensive tasks from the mobile application, so it is able to perform efficiently and is not bogged down with additional computations and server interactions.

Utilizing Firebase cloud functions allows the team to continue its goal in project efficiency and reliability. Having up to date weather API data to help predict roof behaviors is made simple and low cost with the few cloud functions to poll the data, aggregate the data, and delete old data.

Bluetooth Communication Algorithm - To communicate between the iOS application and the microcontroller specific commands are sent via Bluetooth. The main data that is sent to the microcontroller are the new thresholds for the sensor measurements along with the main open and close command for the roof. Each command is formatted in similar fashion, but all commands start with a single character to represent the sensor the new threshold is for, or the command to change the status of the roof.

For commands that change new threshold values the starting character is followed by the threshold float value in the command format of [0xXX , 0xXX , 0x2E , 0xXX , 0xXX , 0x(4F/43)]. The first four commands represent the float value itself, with the whole number, the decimal point, and the decimal value of the threshold. The sixth command is the representation of the roof behavior for the given threshold. The value is either “C” or “O”, representing closed or open. The command for a new threshold is sent to the microcontroller whenever the threshold value or the open or close preference is changed on the app. In total the commands for new thresholds or new roof behaviors contain seven hexadecimal values in total.

The rain sensor is slightly unique compared to the other sensors as it does not have a threshold, only a Boolean if it is raining or not raining. For the rain sensor, only the roof behavior command can be changed for the preference of the roof when it is raining. The rain command only contains two sub-commands. The preceding command letter followed by the value of “C” or “O” for the roof status preference.

The only other command sent to the microcontroller is the open and close roof command. This command is denoted by the character “S’. It does not contain any other hexadecimal values. The microcontroller decides what the current state of the roof is and operates the relay and actuators to the opposite state.

In total there are six commands sent via Bluetooth to the microcontroller. The commands specific formats and justifications are outlined in *Table 43*. The specific values for each command are also outlined in both decimal and hexadecimal because both are used to determine the specific command and the respective sensor the value is for. The hexadecimal value is used for

differentiating between commands and the decimal value is to specify the specific sensor when assigning the values.

Table 43: Bluetooth Commands

Command Character	Decimal Value	Hexadecimal Value	Sensor/Threshold or Command	Command Format
W	87	0x57	Wind	[0xXX , 0xXX , 0x2E , 0xXX , 0xXX , 0x(4F/43)]
U	85	0x55	UV Index	[0xXX , 0xXX , 0x2E , 0xXX , 0xXX , 0x(4F/43)]
L	76	0x4C	Temperature Min	[0xXX , 0xXX , 0x2E , 0xXX , 0xXX , 0x(4F/43)]
H	72	0x48	Temperature Max	[0xXX , 0xXX , 0x2E , 0xXX , 0xXX , 0x(4F/43)]
R	82	0x52	Rain Sensor	[0x(4F/43)]
S	83	0x53	Open/Close	N/A

5.3 Mechanical Design

Although this project's main focus is the electrical and software design of the A.I.R.E. system, there are some mechanical aspects of the project that require some level of explanation and detail. This section is a small overview of the mechanical side, including the housing needed for the electrical components as well as the prototype design that is used to demonstrate the functionality of the system.

Proposed Enclosure – To house the electrical components of the system, we wanted to 3-D print a rectangular prism like enclosure designed out of Acrylonitrile Butadiene Styrene (ABS) plastic. This encasing would have protected the electrical equipment from accidental damage or the slight ingress of any liquids or substances that could potentially damage the electrical parts. This protection would be provided by the design of the enclosure itself, as well as the materials used to make it.

ABS was the perfect material for the enclosure as it is cheap and has water resistant properties after a bit of treatment. Once the design is printed, an acetone treatment along with using water-proof epoxy to seal every dimension of the shape together would make the enclosure suitable for the purposes of this project.

In summary, the proposed design consists of two rectangular prisms, both without a top face and sharing the bottom face, effectively making one structure. The dimensions for the prism are 12 inches in height, and 6 inches for the width and length. The purpose of removing the two faces is so that one can place the electrical components inside of one of the sides to then enclose it with another ABS-based face, using water-proof epoxy to properly seal. There is holes in that "in-between" face, allowing for air flow into electrical components, although heat is not an expected issue when final developments are completed.

The other side (the other prism) would house the sensors that require to be exposed to the air. This include the atmospheric pressure sensor and temperature sensors. The holes in the in between face is used to connect the sensors to the microcontroller. Other Sensors is connected the same way as well, with waterproof wires because of the fact that these will need to be placed in the places outside of box. This is more specific to rain and ultraviolet sensors.

It is important to note that this design will only be good if the rectangular prism is placed on the side of a building. This is the only way that one can waterproof the whole system, while still exposing the sensors to the outside environment to gather the proper data.

Proposed Prototype - With OpenAire being a company that makes custom retractable roof enclosures, it was important to test the functionality of the A.I.R.E. system on a model roof structure. Of course, this model could not be an actual sized roof, as that would drive project's costs up dramatically and would be difficult to demonstrate. Luckily, the intention for creating this system was to show that we could control the roof based on the intelligence of the A.I.R.E system, which is driven by all the sensors data, and how back-end algorithm uses that information to retract or contract. For this reason, the most important thing was to be able to see that we could trigger a motor to activate based on the microcontroller's signal.

This is the whole concept behind the team's prototype design, where two linear actuators are built into a building like structure and are activated based on the microcontroller's signal. Upon activation, these motors output a force that drives a roof back and forth. More detail about the whole structure and how it operates is discussed below.

The first part that is discussed is the building structure itself. The model is meant to be similar to the buildings which OpenAir works in, mainly meant to house pools. The built prototype consists of 3 separate entities that are then connected. The first entity is a 10x10 inch cube, with two faces removed, the bottom and one of the sides. The walls (The side faces of the cube) are 1 inch in thickness, while the roof (Top face) is two inches in thickness. The reason for the larger thickness on the roof is because a carve and trim of .75 inches in height and .90 inches in width is needed to satisfy linear actuator detail design, with the purpose of placing this motor within the altered section of the face. Refer to the linear actuator section for more detail regarding it.

The top of the roof will also have two linear rails, with carriage blocks, about 12 inches in length, placed 4 inches from the back edge of the roof; since they railings are longer than the reaming part of the roof, they is outside of the tops area, which is the intention. This is furthered discussed in the next paragraph. The placement has to do with the fact that these railings are used to drive another roof, a 6x6 inch square on top of the rail's carriage blocks, serving as the retractable roof. This is pushed by the linear actuator from the top of the first module, onto ser. To give a better depiction of what is being discussed, refer to the sketches below, as well as the free body diagram which will explain why things were chosen as they were.

The second entity is simple, two wooden walls, 6 inches in height, 12 inches in length, and 1 inch wide, connected vertically to the first module, as well as the third model's module. There is no top face between these two side faces because the 6x6 roof on the railings on the first module will slide onto the top of these two faces. This is done through the force exerted by the linear actuators.

The railings will come off the first module's area and will continue on top of the side wall's top edge, allowing for this to happen.

This is effectively how the retractable roof is simulated, or at least the first half. The other side of the two vertical walls will have the third module, which is a copy of the first module just on the opposite side. Same functionality, with the same specifications as the first module. Things were made this way in order for the two sliding roofs to meet in the middle and close. In order to prevent collisions between the two sliding roofs that may cause damage, a proximity switch sensor, connected to the microcontroller, is used to communicate to the linear actuators to stop exerting force. This is more of a safety net, as the dimensions of the modules were selected specifically to prevent any form of problems of this nature. Refer to the sensors section of this paper.

The physics behind this design is based on the following free body diagram:

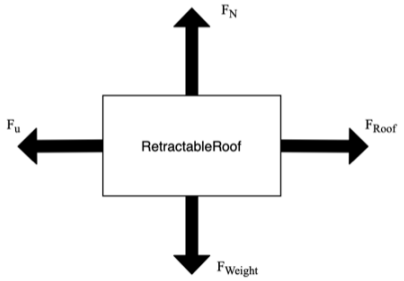


Figure 26: Retractable Roof Free Body Diagram

It is known that the coefficient of friction for ball style carriages is 0.002-0.003. These are the carriage style for the linear railings selected, the "Iverntech MGN12 300mm Linear Rail Guide with MGN12H Carriage Block for 3D Printer and CNC Machine".

This low friction coefficient allows for a large amount of weight to be moved in the x direction without a lot of force needed. This is part of the reason that the "PA-07 linear actuator" was chosen, aside from its dimensions that made it easy to fit in the design of the prototype structure. As 12V are applied to the actuators the actuator is able to extends it stroke, 6 inches in length, with a force of 5 pounds. If one converts these 5 pounds into Newtons, 22.24 N, one can solve for the total Mass that can be moved:

755.69 Kg is the maximum amount of mass that the actuator can actually move, more than enough of a threshold for the mass of the roof. The dimensions and the overall force that "PA-07 linear actuator" outputs are more than fit for the purposes of this model.

Final Building Design – The final building design is shown below in the following SolidWorks drawing. It is a 32"x10"x11" rectangular box where the top panels include indents for the actuators to fit in, and the plexiglass panels for the roof is attached to the actuators. This design was done with ¾" plywood in mind as it is inexpensive, easily accessible, and lightweight enough to fit within the specifications we sent for ourselves. The size constraint was altered slightly as the size of the actuators played a large role in deciding the enclosure dimensions.

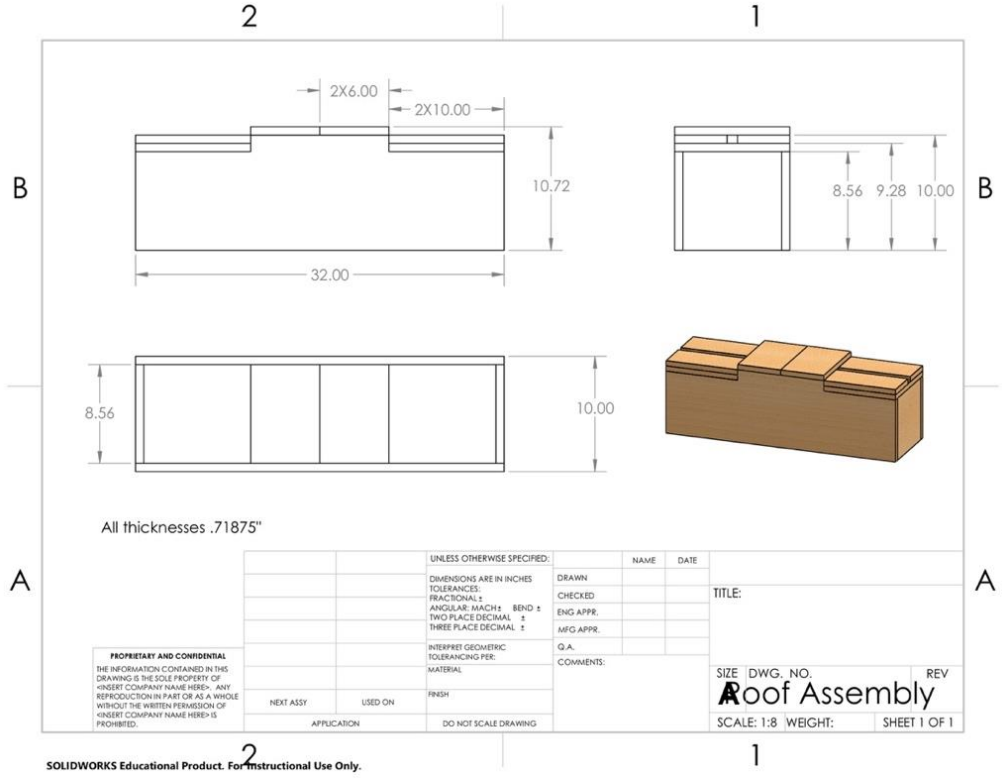


Figure 26: SolidWorks design of enclosure

In the final build, this structure is made from medium density fiberboard (MDF) plywood. The material changed from the original plywood choice as one of the group members already had MDF in their possession pre-senior design. Instead of spending unnecessary money, we went with MDF which was the same thickness and general weight of the proposed plywood.

The solar panel is attached, at an angle, to the side of the building, and the sensors will attach to the corner of the roof out of the way of the plexiglass panels. All other components that are not required to be outside, is inside the enclosure away from the roof's opening. This is to protect the essential working components from damage due to outdoor weather conditions. As mentioned above, we wanted to build another structure to place all working components in in order to protect them from the weather. However, due to COVID-19 restrictions we were unable to use the university's on campus printing lab, and no one in the group would have been able to obtain a 3-D printer feasibly.

6. Prototype Construction and Coding

As with all projects, the initial stages of ours isgin with combining our initial designs into an integrated schematics section for PCB layout. The first section is dedicated to illustrating these integrated schematics and their respective PCB board layouts. In the next section, we is extracting our gerber files from the PCB design software and comparing different PCB vendors to get the best price, service, and lead time for assembly and delivery. Once that is finished, we will put

together our final designs for the software that we is programming both for our mobile application and the embedded software on the microcontroller. In this section, we will also discuss how we is programming the microcontroller, and our PCB design will reflect it as well.

6.1 Integrated Schematics

This section illustrates all of our PCB designs based on our hardware design from section 5.1. it is broken into three sections: Power Module, Microcontroller, and Sensor Module. Each section shows our PCB designs for each module and any additional comments about our hardware design.

Integrated Schematics and Board Layout for Power Module - The integrated schematics for the power modules are compiled into the figures in this section. These designs are based off of the schematics we proposed in the previous sections. Each design shows the final schematic layout followed by the final printed circuit board (PCB) design.

The integrated schematics for the battery management system are illustrated in *Figure 27*. As discussed in previous sections, three battery management systems are connected together in series. This is done because we need to regulate and protect a large input before sending it to the voltage regulators.

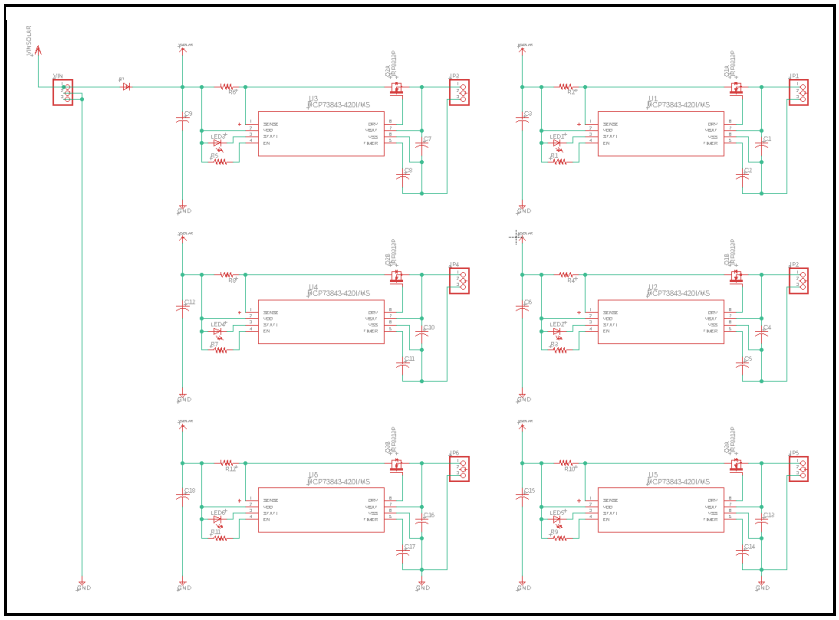


Figure 27: Integrated Schematics for Battery Management System

The board layout for the battery management system is illustrated in *Figure 28*. In this figure, red signifies the top copper layer and blue signifies the bottom copper layer.

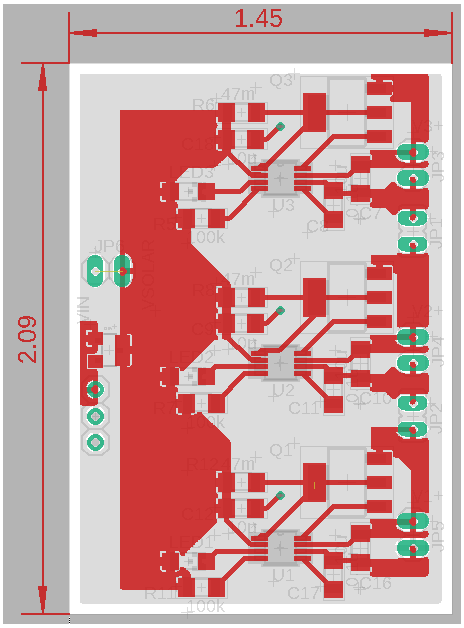


Figure 28: Board Layout for Battery Management System

The integrated schematics for the voltage regulation stages are illustrated in *Figure 29*. As seen, the largest schematic is for the 12V regulator as it is arguably the most important step in this process. Creating a stable 12V is imperative as the next voltage regulation steps will run much more smoothly, and ensure no voltage errors throughout our system.

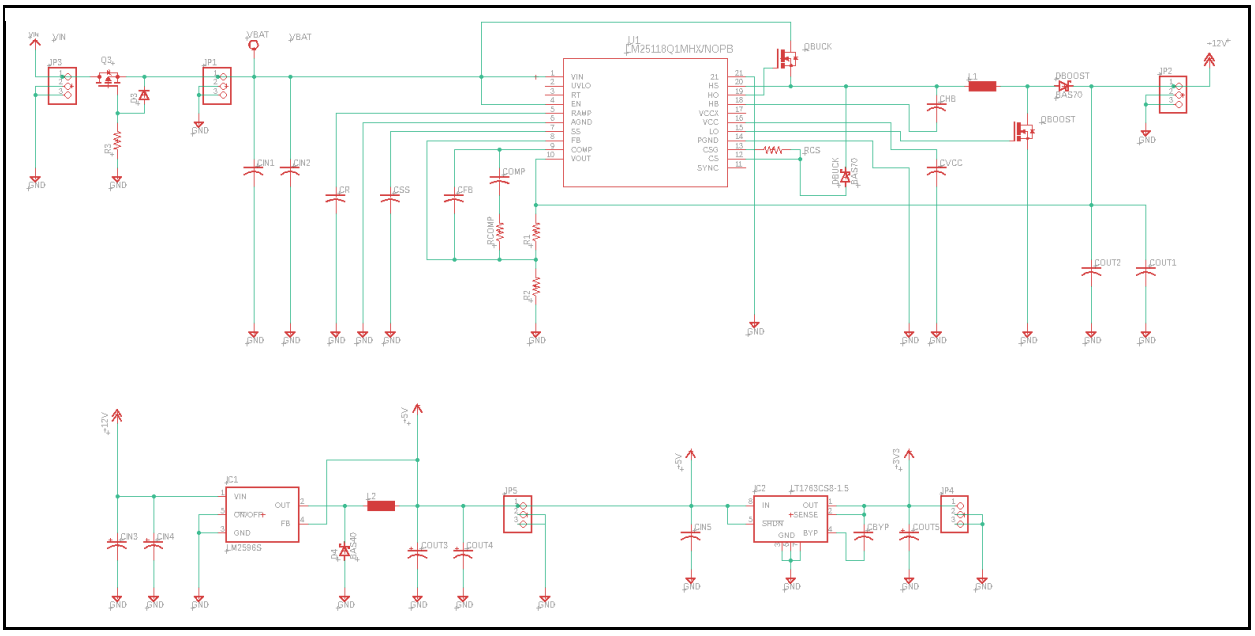


Figure 29: Integrated Schematics for the Voltage Regulation Stages

The board layout is illustrated in *Figure 30*. In the latter figure, red signifies the top copper layer and white is the bottom copper layer.

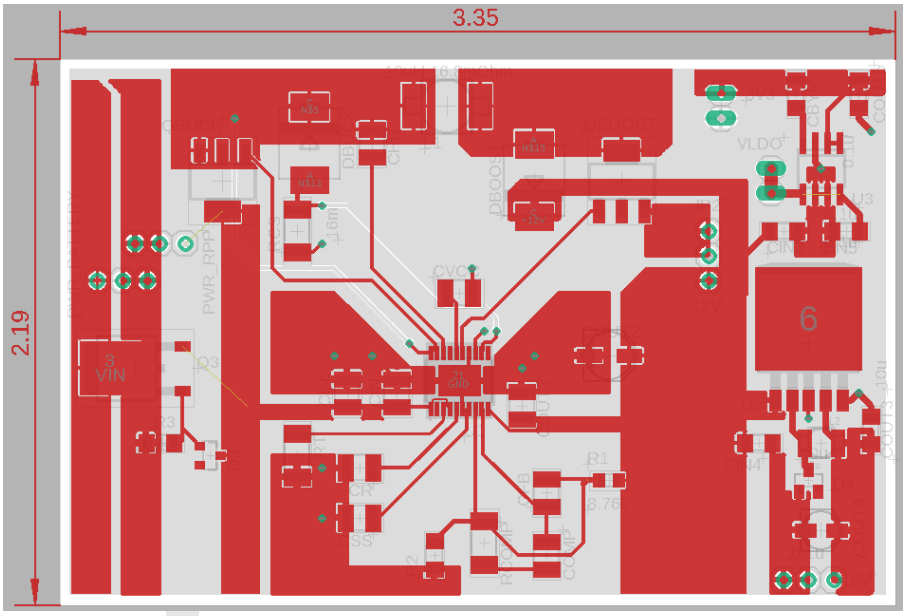


Figure 30: Board Layout for the Voltage Regulation Stages

Integrated Schematics and Board Layout for Microcontroller Module- The integrated schematics for the microcontroller module are compiled into the figures in this section. The integrated schematics for the microcontroller are illustrated in *Figure 31*. As shown all ports are connected to jumpers and a few external components to regulate the signals coming into the microcontroller. Each jumper connects to either the power module, motor module, Bluetooth module, or one of the various sensors on the sensor module.

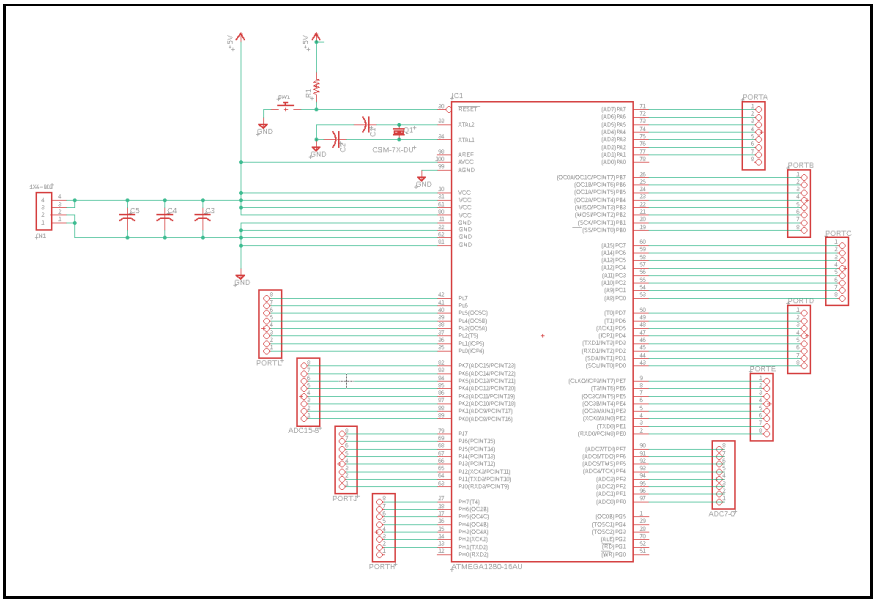


Figure 31: Integrated Schematics for the Microcontroller

The board layout for the microcontroller is illustrated in *Figure 32*. In this figure, red signifies the top copper layer and blue is the bottom copper layer.

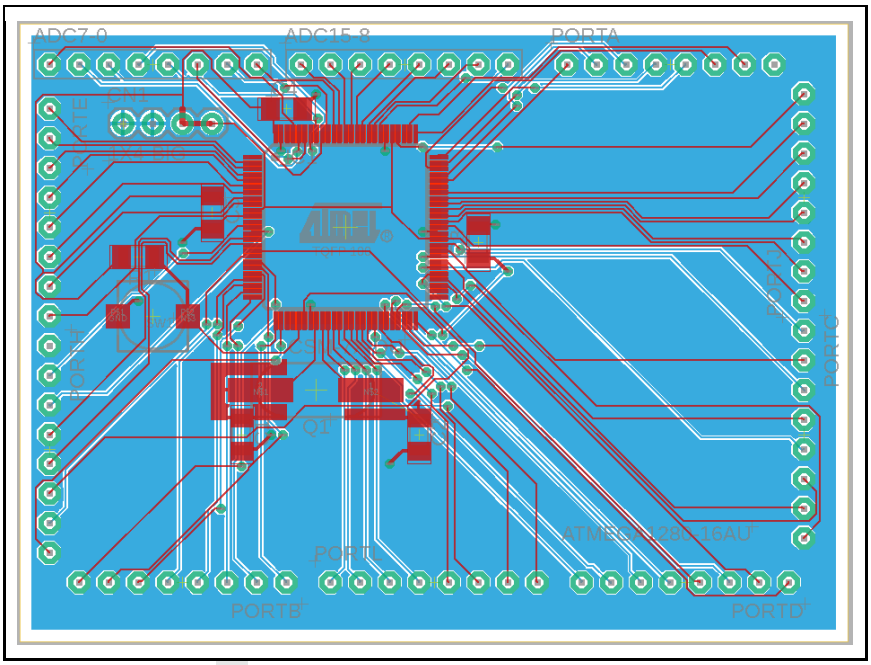


Figure 32: Board Layout for the Microcontroller

Integrated Schematics and Board Layout for Sensor Module - In order to create a design in which all of our sensors operated as desired, we went with a multiple part PCB design. One PCB is placed outdoors holding all of our sensors. This must be directly outdoors to operate correctly and accurately , and it will connect to the microcontroller PCB inside the building. The proximity sensors that need to be in other various places around the building will also connect back to the inside MCU PCB. *Figure 33* depicts the overall sensor board layout.

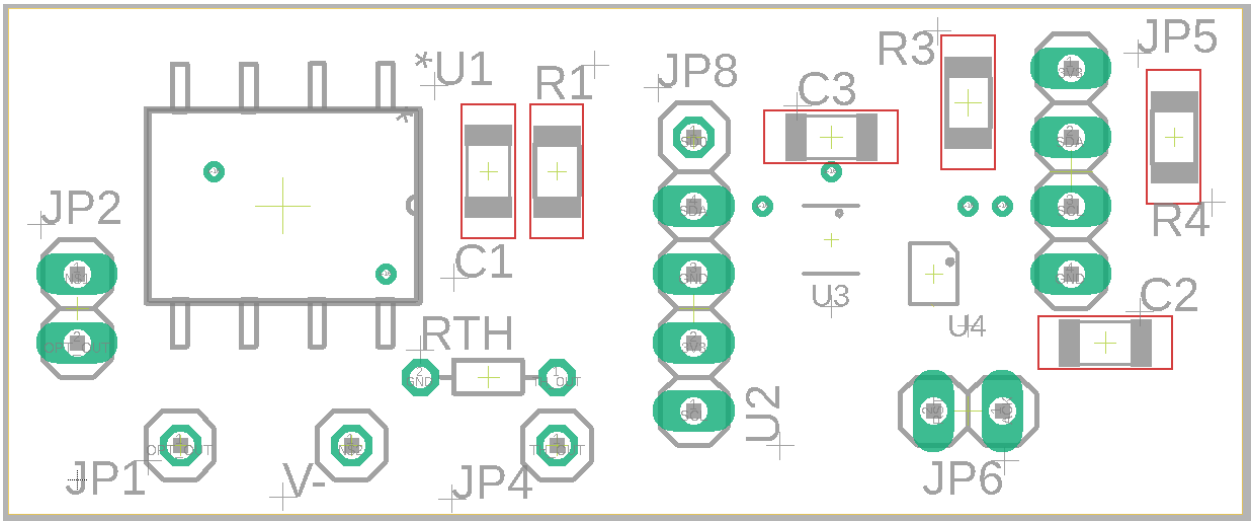


Figure 33: Sensor Module Placement

Figure 34 depicts the top layer of the sensor module. One important note is the placement of the copper layer being kept away from any sensors that may be heat sensitive, or may be in accurate due to the external PCB heat.

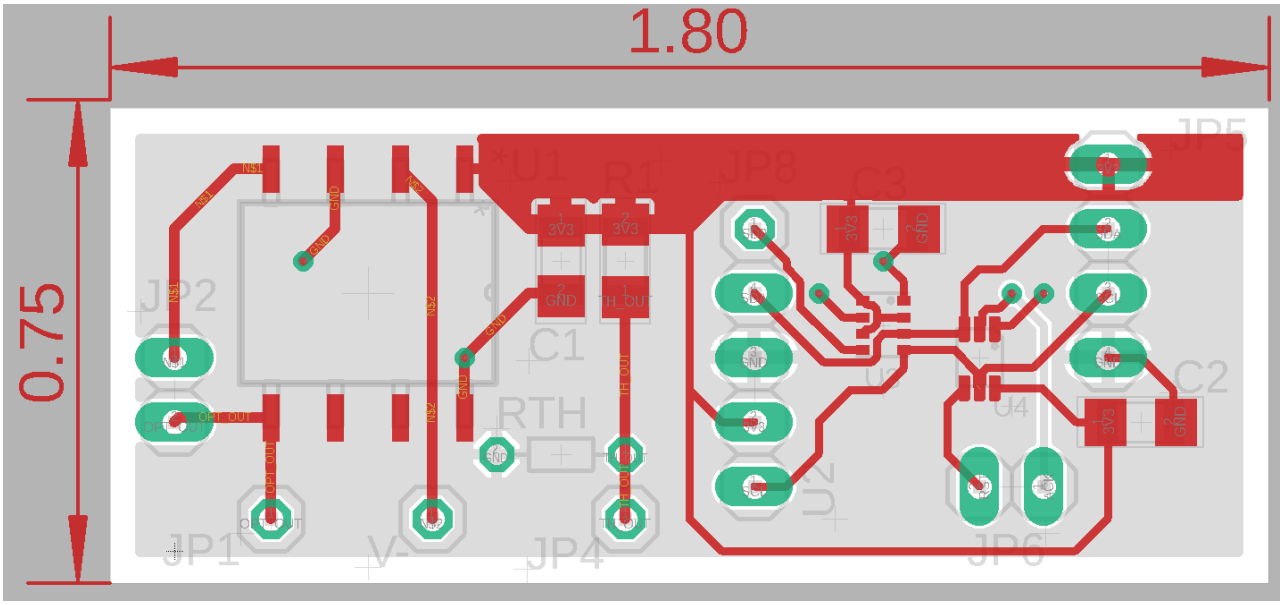


Figure 34: Top Layer Board Layout for Outside Sensor Module

Figure 35 depicts the bottom layer of the sensor module PCB. Green depicts any components or through hole placements in the PCB

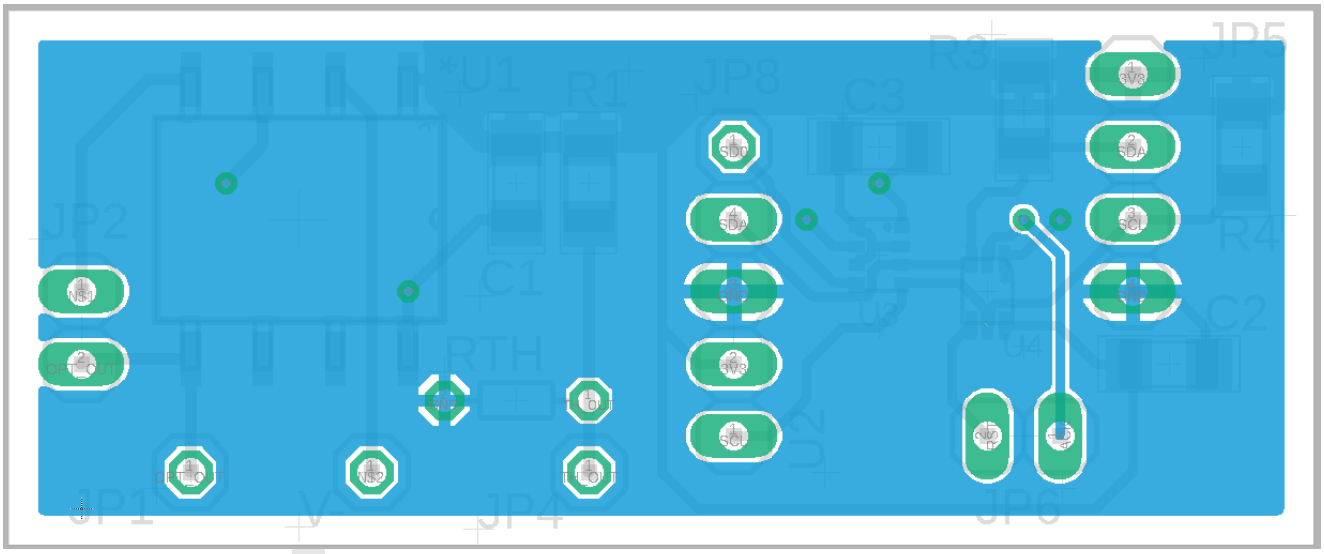


Figure 35: Bottom Layer Board Layout for Outside Sensor Module

6.2 PCB Vendor and Assembly

For any electronic project with a deadline, the PCB vendor used is almost as critical as the designing and testing of the prototypes built. Therefore, in this section, we is reviewing the different available PCB vendors. Our major criteria for selection involve price point, lead time, assembly services, and previous customer reviews.

The available PCB manufacturers that we have come across, with a demonstrable service record, include JLCPCB, PCBWay, and 4PCB. For our PCB needs, JLCPCB offers $4 per 5 boards, $7 in shipping, and a 3-week lead time for delivery. They also offer assembly of the PCB in-house. PCBWay offers $5 per 5 boards, $17 in shipping, and a 2-week lead time for delivery.

While PCBWay offers a faster lead time, we is using JLCPCB for our initial PCB vendor for the low cost and assembly service. Our team is familiar with JLCPCB, and so we heavily favor using them for our PCB manufacturing needs.

In terms of assembly, a large majority of the components we are purchasing and assembling onto the PCB are surface-mount-technology. Therefore, we may need external aid in assembling the final prototypes of the project. For this reason, we is seeking the services of Quality Management Services due to their extensive history aiding in UCF's senior design projects and convenient proximity to the UCF campus. For assembly that we are able to complete on our own, we is utilizing the UCF-provided soldering stations to assemble what is needed

7. Prototype Testing Plan

In order to test the functionality of the software on the microcontroller integrated with the hardware for the rest of the system, a test plan is followed throughout the life cycle of the project. The test plan is broken into multiple parts to test all the components extensively as singular pieces and as components in the system.

7.1 Hardware Test Plan

This section will describe our plans for testing all of our hardware modules. Beginning with how we plan to order our designed PCBs, then going into the test plan for the sensors, power module, and microcontroller. Separate testing is done first before combing all hardware together, for easier debugging for any errors in our system.

7.1.1 Sensor Module Testing Plan

Each sensor is tested separately to check calibration, communication protocols, and accuracy. These kind of unit tests will allow the development team to catch any errors in the basic functionalities. The sensor is tested at its respective measurement boundaries. The communication is checked for both sending and receiving signals. Sensors that are used in multi-sensor measurements, such as the temperature and humidity sensor, is tested together for the same criteria

which a single sensor is checked. The table 44 include examples of tests that were completed to check the multiple sensors.

Table 44: Sensor Test Plan

Sensor	Test
Temperature	Sensor is exposed to four different temperatures, two of which is at the extremes of the range, in order to check for the validity of the measurement and the relay of the data to the microcontroller.
Humidity	Sensor is exposed to four different humidities in order to check for the validity of the measurement and the relay of the data to the microcontroller.
Wind	Sensor is exposed to different wind speeds from different directions to check that the wind is detected and the speed measured accurately.
Rain	Sensor is exposed to different rates and volumes of water to check that the signal to the microcontroller is accurate and timely.
Barometric Pressure	Sensor is monitored for 24 hours and compared to the National Weather Service's data for those 24 hours. Verify that the Weather API is polled whenever there was a change.
Light	Sensor is exposed to different light levels and UV indexes to check accuracy. Verify that the signal to change interior lights is sent when light lumination changes.

The roof actuators will each be tested separately to check calibration, communication, change in location, change in direction, and for uniformity in their force applied. Signals is sent to the actuator to move in the positive direction, then the signal will change for the actuator to move in the negative direction. This will test the communication between the microcontroller and the actuator, and the functionality of the actuator in one simple test.

Once verified, the components is added to the team's constructed PCB and more testing is completed. The sensors will go through the same testing as mentioned in Table 44. The actuators will also go through the same testing as before to demonstrate the total functionality. This is to verify that the soldered connection did not cause any damage to the component. This testing will also highlight any interference a component might have on another component.

After the final verification of the component's measurements and functionality on the PCB, the system will undergo multiple fully immersive artificial tests. These artificial tests is created by the development team to test the range of the sensors. Each sensor will have at least two tests that are created specifically to target that sensor. There is five tests that will test the system as a whole.

7.1.2 Power Module Testing Plan

The power module, which consists of a solar panel system, lithium ion battery pack, battery management system, reverse polarity protection, and three stages of buck/boost and voltage regulators, will have its testing plan outlined below.

Solar Panel Test Plan - The solar panels will need to be tested to ensure that their datasheet specifications are relatively accurate and that they are capable of supplying the amount of current necessary for charging a lithium ion battery at 0.5 C. According to its datasheet, the ECO-WORTHY 5W 12V Solar Panel can supply a maximum of power of 5 W. At a charging voltage of 1 A at 4.2 V, the solar panel will need to be able to handle this 4.2 W charge.

In order to determine if the solar panel is capable of this, the solar panel is placed outside at noon, in order to input the maximum amount of sunlight, and connected to a load of 1 ohm, 10-Watt resistor. The voltage across this resistor is measured as well as the current flowing into the resistor to determine if the solar panel will truly be capable of supplying 5 W of power.

Battery Pack Test Plan - The lithium ion batteries we is using will need to ensure that the capacity it is rated for is true to within +/- 1%. The lithium ion batteries we are using are rated for a nominal capacity of 2.6 Ah and can handle a maximum discharge rate of 0.2 C to a minimum of 3.0 V. This translates to a total discharge time of 5 hours, supplying a maximum of 0.52 A per cell. In order to provide more current, a 3-S, 2-P (3 series, 2 in parallel) configuration is used to supply up to 1.04 A with a supply voltage range from 9 – 12.6 V.

To test the battery pack's capacity, each individual lithium ion battery is connected to an active load that maintains 0.5 A of output current and the battery's voltage is monitored from its peak voltage of 4.2 to 3.2 V. Every 15 minutes, the battery's voltage is written down, and, at the end of discharging to 3.2 V, the data is input into excel and analyzed.

To test the battery pack's performance, the batteries is set up in a 3-S, 2-P configuration and an active load of 1A is maintained from a peak output voltage of 12.6 V to a minimum of 9.6 V.

Due to the danger posed by these tests, the active load will have to have a precision current limiter of 1.01 Amps and, if possible, a fuse rated for 1.05 A is placed in between the batteries and the active load.

Battery Management System Test Plan - Due to the integrated circuit's nature of already coming with safety features, the main goal of the testing of the BMS IC is towards its current handling capability and ability to charge our 3-S, 2-P configuration of lithium ion batteries.

The BMS IC's safety features, which includes its overcurrent and overtemperature protection, is tested by essentially abusing the BMS IC. The BMS IC's outputs is applied to a very low, 1-ohm, 10-W power resistor to observe if it terminates its output current to avoid overcurrent. In another test, a lighter is waved across the BMS IC multiple times to observe if overtemperature protection activates at the time specified in the datasheet. Once these safety features are determined, then the BMS IC will have its performance measured in the following paragraph's testing plan.

To test its performance and overvoltage protection, the BMS IC is set up as it is in the initial design in Section 5.1 and then connected to the battery pack system. During a 2-hour window, the batteries' voltages is measured to ensure that each voltage remains at a similar voltage to the batteries surrounding it. Once this window is concluded and the BMS IC has ceased its charging, the voltage of each battery that it has charged is measured to ensure that they are all 4.2 V. This will ensure consistency in the supply voltage of our battery pack.

Reverse Polarity Protection Test Plan - The reverse polarity protection circuit is testing to ensure that it can handle up to –20V without conducting electricity on the load side. The reverse polarity protection circuit isgin with –10V at its input, respective to ground, and increased to –50V over a period of 2 minutes. To measure the magnitude of its output current, a load of 100 ohms is applied to the output and the output measured throughout the testing. If there exists an excessive output current of >1 mA, then the reverse polarity protection circuit will need a new type of power MOSFET capable of handling the input voltage better.

12V Buck/Boost Controller Test Plan - The buck/boost controller that will take in the battery's voltage and output a steady 12V is tested to measure several key performance and standard metrics: efficiency and buck/boost capability.

According to its datasheet, the LM25118 should have an efficiency of between 85 – 95% when its input voltage is between 10 – 20V and its output voltage is 12V while its output current is between 1 – 2 A. To ensure that the LM25118 chip we have is capable of doing this, we will supply the buck/boost controller with multiple voltages, maintain its output voltage at 12 V, and apply an active load of 2 A at its output. The input voltages that is applied to the buck/boost controller will range from 10 – 20V, with increments of +1V. The efficiency at each increment is measured by comparing the input and output power. This test will also measure the buck/boost controller's buck/boost capability.

5V Buck Converter Test Plan - The 5V buck converter's testing plan is geared towards performing two key measurements: the buck converter's efficiency and output line regulation at high current loads.

According to its datasheet, the LM2596 should have a typical efficiency of 80% when supplying 3A of current and 5V of voltage with an input voltage of 12V. To test this in accordance with our own requirements, the LM2596 is supplied with input voltages of 11V, 12V, and 13V, have a steady output voltage of 5V, and be connected to an active load of 2 A. The input and output power is compared to determine if the efficiency is ≥80%.

According to its datasheet, the LM2596 should have an output line regulation of a maximum of +/- 0.1% output voltage change with an input voltage of 12V, output voltage of 5V, and output current of 0.1 A. This is scaled up to +/- 0.5% output voltage change of similar input and output voltage, but with a load current of 1 A. This will ensure that the output voltage remains within the range of 4.5 – 5.5 V.

3.3V Regulator Test Plan - The 3.3V regulator's testing plan is geared towards measuring its efficiency, since in its role, it will only need to provide a relatively small amount of power.

According to its datasheet, the LT1763 can supply an output current of 500 mA. Our testing plan will ensure that it is capable of doing so with an efficiency of ≥80%. To test this, an input voltage of 5V is applied, and an output of 3.3V is maintained with an active load of 500 mA connected to its output. The input and output power is compared to determine if the efficiency is ≥80%.

7.2 Microcontroller Firmware Test Plan

To test the software implemented on the microcontroller both unit tests and end to end test is implemented. Unit tests will allow the development team to check the validity of the data through different procedures and communications. While end to end test will check the connectivity of the multiple components on the board and their ability to exchange data accurately and efficiently.

Unit tests is implemented first for testing the microcontroller software. The unit tests do not require that the whole system be in place. In fact, it is preferred that unit testing take place when only a single sensor or component is in place. Each measurement from a sensor is read and stored in a temporary variable. That specific measurement is checked before and after executing the logical tests for checking threshold values, as depicted in image XX. Testing with each measurement also allows the development team to check the expected signals for changes in roof behavior. The expected signals is validated with unit tests by checking the creation of the signal after receiving a trigger signal, and checking the actual state of the signal.

To validate the connection and communication with the database, the measurement is checked when received from the sensors, then it is checked again before being sent to the database, and that same measurement is read back from the database after a short delay (approximately three seconds). It is expected that the two values should match. The value will also be checked within a database management user interface for correct positioning within the different tables and columns, specifically checking the values of the primary keys to line up. This validation is used for each measurement in the database to ensure the structure of the database and queries are correct throughout.

End to end testing is implemented once the whole system has been constructed. Each sensor has two end to end test. The first end to end would be between the microcontroller and the sensor, for both sending and receiving signals. This is tested by checking each sensor triggers its appropriate response, or lack of response, from new stimuli. The appropriate response for testing is based off of the roof's default behaviors as outlined in Table XX. The other end to end test for validating the software on the microcontroller is to ensure that sensor data is being stored properly in the database. This is done similarly as the unit test but the value of the measurement is controlled by the development team, and will only be checked in a database management interface.

7.3 Mobile Application Test Plan

No different than then other software-based test plans, the iOS application is tested using both the end to end and unit testing methodologies, as both procedures fit the different facets of the application. End to end unit testing is done to test the application when it comes to functionality and ensuring that the user is able to access every page within the application. During this procedure, no variables is altered, and no action which would affect the roof's activity is invoked. These is tested during the unit step testing phase of the Mobile Application test plan, where different parameters within the application such as Roof Setting fields, User Authorization fields, and Roof Control fields is modified and tested in order to ensure that respective field changed or remained the same based on whatever is expected for the variable to do.

The end to end process will consist of login into the application with an administrator user and password; if one is able to successfully login to the application without a problem, one can continue with the procedures. Once within the application, the test will then lead each of the different pages and sliders in which a user can end in. In other words, one will ensure to enter all the different pages that are accessed based on the buttons that are pressed, and also make sure that the proper sliders are displayed when 3D pressing the buttons that open a slider.

The test isgin in the controls page, since it is the default start page when the user first logs in. As discussed in the page's section of the document, this page consists of a few different buttons that will invoke a slide once they are pressured touched. During this end to end testing, one ensures that each of the buttons do what they are supposed to once these pressing events take place. In this part of the testing, the sliders will not be manipulated, as that will take place during the unit test phase of the procedure. Once all the buttons are verified to be working, the team goes on to test the settings page. The settings page has two buttons, one the 'Roof Settings' the other for the 'User Settings'. To test this part end to end, one just presses each of the buttons and ensures that the application redirects the user to the page that corresponds. In other words, pressing the 'User Settings' button leads to the modifiable table of the SQLite database, while pressing the 'Roof Settings' button leads to the page with all the 3D press buttons, which are tested the same way as they were tested in the controls page. Once again no values are manipulated. Last step in this end to end testing phase would be going into the weather metric page, which involves tapping its respective button in the ribbon, and ensuring that the user is taken to the correct page and can view all the metrics.

As mentioned, the unit test will consist of ensuring that the variables and other fields within the different pages of the application remain constant or changed based on whatever the user does. Because of the large quantity of variables that make up the backend and frontend of the application, the section will only cover certain examples, as a lot of the different variables are tested the same way, which means there is no need to repeat the procedure repetitively. The main focuses behind this unit test procedure is ensuring the user logged in remains constant throughout the whole process, and verifying the different fields in the controls page and setting pages can be modified as the user desires.

In the settings page, an authorized user is allowed to modify the different attributes that would cause the roof to open or close based on what the user desires. Whether this be pressure, humidity,

temperature, or any other of the important metrics mentioned, all these are modified in the same manner, therefore unit testing them is approached the same way for all of them. In other words, to verify that these variables are manipulated as desired, one effectively modifies the attribute at hand, and then checks the corresponding value at the register level, ensuring that the value remains the same throughout all procedures, except for going through the event of actually changing the threshold value through the slider in the settings page. One also effectively unit tests one of the fields by checking if the parameter holds true versus real time measurements and is reacting as expected based on these real life measurements. In other words, if one were to set the roof to close when the temperature is greater than ninety degrees Fahrenheit, one can verify this by making sure that this statement holds true, and that the system sends the proper signal to close the roof if the condition is met. Because the fields are similar in the controls page, a similar process if followed for the fields in that page, where one manipulates the variables, whether that be the open/close Booleans, or the brightness levels of the lights, and checks that the values are changed in the register level and cause the system to properly behave based on these changes.

The second type of field that can be altered and unit tested is the system's database values, more specifically the permission fields for all the users in the table. As known, an authorized user is able to access the User Settings page and manipulate the different permission fields for each user registered in the system. To test when one of these parameters are changed, there are three main things analyzed. The first is ensuring that once a field is changed within the user settings page, the change is reflected within the database itself. This is simply done by changing a field via the application, and then cross referencing whether this change was implemented by querying the information from SQLite database via another source. The second thing to be sure of, is that once this specific parameter is modified, the behavior of the application changes, and that the user modification actually affects what the modified individual can or cannot access. Lastly, one ensures that changing a field does not affect any other variable and remains constant throughout any procedure executed in the application other than the event of changing the field. An example of this procedure would be if testing whether a user has 'Roof Control Rights'. If one is to change this field to false, this unit test would consist of verifying that this specific cell within the table is 'False' when querying it from an external source. Then one would verify whether the change actually took place by testing whether the user modified did indeed loose access to controlling the roof. Lastly, the individual executing this testing procedure shall ensure that this specific field doesn't affect any other fields within the database and remains constant unless changed.

8. Project Operation

This section will detail the final working design of our project. As the semester progressed multiple changes had to be made to our initial design. Due to the unforeseen circumstances of the COVID-19 pandemic, constraints and requirements of the project changed drastically. This section includes any updates to our final design using the technologies discussed in sections previous, as well as discuss the addition of new technologies as the team adapted to these unprecedented changes.

8.1 Hardware Operation

The operation of the hardware for the final prototype are described in this section. The sensor module, power module, and microcontroller module operation are elaborated upon in the following subsections. Each subsection will include details of the final design and how to use and operate each subsystem to create the final product.

8.1.1 Sensor Module

In order to connect the sensor PCB to the microcontroller development board, one needs six jumper wires in total. The four-pin header at the top of the sensor PCB was designed to be used to connect the development board and the sensor PCB for I2C communication. From pin one to four, they will need to be connected to Vcc(+5V), SDA(SDA), SCL(SCL) and GND(GND), respectively. The pin header nearest the thermistor will need to be jumped to Pin Analog_0 of the analog pin header on the development board.

The external sensors connected to the development board include the rain sensor, which needs to be jumped to Pin Analog_1 of the analog pin header on the development board. In addition, it must be connected to Vcc(+5V) and GND(GND) of the development board in order to exchange power.

8.1.2 Power Module

In order to operate the power module, some extra wires are needed. The solar panel, set up next to the enclosure, needs to be connected to the input voltage pin header of the BMS board. The output voltage pin headers of the BMS board should be connected to the high and low terminals of the lithium battery cells.

The ultimate ends of the lithium battery pack are connected to the input terminals of the power PCB. The power PCB regulates the range of voltages provided by the battery to +12V for input power to the rest of the weather system. The +12V output of the power PCB is connected to the Vin pin of the microcontroller development board, which has a +5V and +3V3 regulator on board.

The +12V output of the power PCB is also connected to the power pins of the actuators that drive the glass roof. The +5V of the development board is connected to the relay system that toggles operation of the actuators, in addition to the sensor PCB and rain sensor.

8.1.3 Microcontroller Module

Our chosen microcontroller for this project is the ATMega 1280. However, due to the circumstances, we were unable to complete the designing and prototyping of the microcontroller PCB. We were also unable to order any additional PCBs. Because of these challenges we have decided to use the BlunoMega1280 development board for this project. To replace the PCBs, breadboard circuits were made, if needed, to connect all components to the Bluno Mega 1280.

8.2 Firmware Operation

This sections details the operation of the firmware. Below is the final pins and ports used for the operating system. These pins are based off of the specific peripherals used in the system along with the available connections on the development board. If the pins are connected correctly, and the system is powered on, nothing else should be done to the firmware to operate the roof enclosure.

Table 45: Final Connections of all the Sensors

Device/Sensor	Pin	Port/Pin	Peripheral
~~Wind~~	~~6~~	~~PE4~~	~~Interrupt~~
~~Pressure~~	~~43/44~~	~~PD0/PDI~~	~~I2C~~
Humidity	43/44	PD0/PDI	I2C
Temperature	97	PF0	ADC
Rain	96	PF1	ADC
Light (UV)	94	PF3	I2C
Actuator Relay	1	PG5/PG11	PWM
BLE (RX)	2	PE0	UART
BLE (TX)	3	PE1	UART

8.3 iOS Application

This section details the workings of the iOS Application developed. The first page of the application is the log-in page. The user can input their username, which should be their email, and the password provided by the administrator. Once they log-in, there are three more pages available. The first page is the settings and user authorizations, where the user is able to access roof settings where they can input numbers to change the thresholds for the sensor values and see the authorizations for different users on the application, respectively.

The second page of the application contains the control ability. By pressing the “Open/Close” button on the screen, the user can toggle the opening or closing of the roof manually. The third and final page of the application displays the weather metrics that have been extracted with the weather API DarkSky.

8.4 Mechanical Operation

The building was made out of Medium Density Fibrewood (MDF) and was put together using nails and wood glue. After it was built, the actuators were placed inside the roof indentations, and a plexiglass type material is attached to the actuators to demonstrate a retractable roof. The relay system is connected to the +5V and GND of the development board as described in the mechanical design section above, with the input bits being jumped with a pin jumper to pins 4 and 11 of the development board Analog Output pin header. The outputs to the relays will connect to the

actuators with one relay connected to the negative terminal of the actuator and the other relay to the positive terminal of the actuator. The relays, triggered by the output PWM signals from the development board, will determine whether the actuators receive +12V, -12V, or 0V. The actuators will extend their arm, retract their arm, or stop moving with each respective signal.

9. Administrative Content

All administrative content is discussed in this section. This includes milestone discussion, which involves discussion about our time-keeping, and budget discussion, which tracks how well we are able to keep within our initial budget estimates.

9.1 Milestone Discussion

The initial project milestones for the entire Senior Design project are incorporated into two separate tables, one for Senior Design 1 and the other for Senior Design 2. Initial milestones for Senior Design 1 are compiled into *Table 46* below.

Table 46: Initial Project Milestones for Senior Design 1

Number	Milestone	Completion By
1	Divide and Conquer v1 Discussion with Richie	Sept 25
2	Sponsorship and financial details finalized	Sept 27
3	Major electrical component orders placed	Sept 30
4	Build structure foundation	Oct 6
5	Complete build of roof and mechanical operations	Oct 12
6	Finish research stage	Oct 19
7	Implement power	Oct 25
8	Begin iOS app development	Nov 1
9	Research write up 50-70% complete	Nov 3
10	Master switch for open/close functionality on prototype board complete	Nov 10
11	Begin programming microcontroller	Nov 11
12	Begin PCB designing stage	Nov 18

Further milestones we had set for ourselves throughout the duration of this project are seen below in Table 47.

Table 47: Project Milestones for Senior Design 2

Number	Milestone	Completion By
13	Add sensor connections to PCB	Dec 14
14	Set up Database and Server for phone application	Dec 10
15	Phone application prototype complete	Dec 31
16	Order PCB manufacturing	Jan 4
16	Implement WIFI & Bluetooth on development	Jan 5
17	Implement motor functionality on development board, and have a structure prototype	Jan 10
18	Master switch for open/close functionality on app	Jan 12
19	Implement software onto board	Jan 25
20	Sensor Testing	Feb 8
21	App Testing	Feb 15
22	Full comprehensive Testing (Indoors)	Mar 20
23	Full comprehensive Testing (Outdoors)	Mar 30
24	Final documentation editing and website complete	April 15
25	Senior Design Showcase	End of April

9.2 Budget and Finance Discussion

Our initial project budget is displayed in *Table 48* below.

Table 48: Initial Project Budget

Module	Item	Qty	Item Price	Cost Estimate
Microcontroller	Microcontroller	2	50	100
Power	Power Module	1	20	20
	Battery Management System	1	20	20
	PV panels	2	15	30
Sensors	Temperature Sensor	1	10	10
	Barometric Pressure Sensor	1	50	50
	Pressure Plate Sensor	1	15	15
	Humidity Sensor	1	25	25
	Anemometer (Wind speedometer)	1	50	50
	Adaptive light sensor	1	10	10
Wireless	Wi-Fi Module	1	20	20
	Bluetooth Module	1	20	20
Peripherals and Drivers	Motor Driver	1	10	10
	Fan Driver	1	10	10
	Stepper Motor	1	20	20
	Fan	1	10	10
Miscellaneous	Miscellaneous Passives	1	50	50
	PCB Design and Fabrication	5	5	25
	Glass Tinting	1	100	100
	Waterproof Enclosure	1	200	200
Total Cost				$735

APPENDIX OF DATASHEETS

[a] ZF Electronics, “Magnetic Proximity Sensors (Hall Effect),” MP1021 datasheet, June 2017.

[b] Jacksking, “Jacksking Anemometer, 360 12V-24V DC Pulse Signal Output Aluminum Alloyed Wind Speed Direction Sensor Measurement Tool,” B07SN1V427, Oct. 2019.

[c] Bosch Sensortec, “Digital Pressure Sensor,” BMP280 datasheet, May 2015 [Revised Jan. 14].

[d] Sensirion “Datasheet SHTW2”, SHTW2 datasheet, 2017 [Revised July 2017]

[e] Little Fuse, “Leaded Thermistors”, PR103J2 datasheet, 2018 [Revised Feb. 26 2018]

[f] Texas Instruments, “OPT101 Monolithic Photodiode and Single-Supply Transimpedance Amplifier”, Opt101 datasheet, Jan. 1994 [Revised Jan. 2015]

[g] Vishay, “UVA Light Sensor With I^2C Interface”, VEML6070 datasheet, 2019 [Revised Sept 12, 2019]

[h] ECO-Worthy, “Polycrystalline Solar Panels”, ECO-Worthy datasheet 2019

[i] Great Power, “Lithium Ion Battery”, PRT-13189 datasheet, July 7, 1028

[j] Microchip, “Advanced Single or Dual Cell Lithium-Ion/ Lithium-Polymer Charge Management Controllers”, MCP73842 datasheet, 2013

[k] Texas Instruments, “LM25118 Wide Voltage Range Buck-Boost Controller”, LM25118 datasheet, July 2011 [Revised Mar. 2018]

[l] Texas Instruments, “LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator”, LM2596 datasheet, Nov. 1999 [Revised May 2016]

[m] Linear Technology, “500mA, Low Noise, LDO Micropower Regulators”, LT1763 datasheet, 2019

[n] Atmel, “8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash”, ATMEGA 1280 datasheet, 2014

[o] Cypress “EZ-BTTMXRWICED® Module”, CYBT-423028-02/CYBT-423054-02/CYBT-423060-02 datasheet, 2019 [Revised Sept. 26, 2019]

[p] Progressive Automations, “PA-07 Datasheet”, PA-07 datasheet, 2019

APPENDIX OF REFERENCES

[1] S. W., "Hall Effect Sensor," Electronics Tutorials, 2019. [Online]. Available: https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html. [Accessed 31 October 2019].

[2] S. D., "How Rain Sensors Work," Don's Mobile Glass, 2016. [Online]. Available: https://www.donsmobileglass.com/ask-jacques/rain-sensors-work/. [Accessed 31 October 2019].

[3] S. W., "Light Sensors," Electronics Turtoials, 2019. [Online]. Available: https://www.electronics-tutorials.ws/io/io_4.html. [Accessed 31 October 2019].

[4] A. T., "Working Principle of a Photodiode," ElProCus - Electronic Projects for Engineering Students, 2019. [Online]. Available: https://www.elprocus.com/photodiode-working-principle-applications/. [Accessed 31 October 2019].

[5] T. Kinney, "Proximity Sensors Compared: Inductive, Capacitive, Photoelectric, and Ultrasonic," Machine Design, 1 September 2001. [Online]. Available: https://www.machinedesign.com/sensors/proximity-sensors-compared-inductive-capacitive-photoelectric-and-ultrasonic.

[6] "Anemometer," Omega Engineering, 2003. [Online]. Available: https://www.omega.nl/prodinfo/anemometers.html.

[7] "UV Index," United States Environmental Protection Agency, 2019. [Online]. Available: https://www.epa.gov/sunsafety/uv-index-1#day3.

[8] "The Heat Index Equation," National Weather Service, 2014. [Online]. Available: https://www.wpc.ncep.noaa.gov/html/heatindex_equation.shtml.

[9] I. Hassan, "Swift vs Objective C in 2019," Medium.com, February 2017. [Online]. Available: https://medium.com/swiftify/swift-vs-objective-c-comparison-32aba9dad4e3. [Accessed 3 December 2019].

[10] Shit Automation, "How to control a linear actuator with an Arduino and relays," Shift Automation, 20 January 2016. [Online]. Available: https://shiftautomation.com/control-linear-actuator-with-arduino-and-relays. [Accessed 30 October 2019].

[11] A. Jaffee, "Working with Core Bluetooth in iOS 11," AppCoda, 18 April 2018. [Online]. Available: https://www.appcoda.com/core-bluetooth/. [Accessed 20 November 2019].

[12] "JetStream Max: Beaufort Wind Force Scale," National Weather Service, 2019. [Online]. Available: https://www.weather.gov/jetstream/beaufort_max.

[13] "Monthly Weather Forecast - Tampa FL," Weather Atlas, 2019. [Online]. Available: https://www.weather-us.com/en/florida-usa/tampa-climate.

[14] NOAA, "Weather Observations for the Past Three Days," NOAA, 2019. [Online]. Available: https://w1.weather.gov/data/obhistory/KMCO.html.

[15] MIDE, "Air Pressure Altitude Calculations," MIDE Technology, 2019. [Online]. Available: https://www.mide.com/air-pressure-at-altitude-calculator.

[16] NOAO, "Reccomended Light Levels," National Optical Astronomy Observatory, 2015. [Online]. Available:

https://www.noao.edu/education/QLTkit/ACTIVITY_Documents/Safety/LightLevels_outdoor+indoor.pdf.

[17] A. Jaffee, "Working with Core Bluetooth in iOS 11," 18 April 2018. [Online]. Available: https://www.appcoda.com/core-bluetooth/.