Cloud Motion Vector Sensor System

Monitoring and Predicting Output Power of a Photovoltaic System in Real-Time

Worcester Polytechnic Institute
Department of Electrical and Computer Engineering

Major Qualifying Project

Authors:
Ferreira, Jonathan
Lewis, Tim
Sauter, Evan

Advisor:
Prof. Mughal, Maqsood A.

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements of the Degree of Bachelor of Science.

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on the web without editorial or peer review.

03/25/22

Acknowledgements:

We would like to thank Worcester Polytechnic Institute for allowing us the use of lab space, the rapid prototyping labs, and the use of an on-campus site for the implementation and testing of the CMVS system. We are also grateful for the assistance of the graduate students, Habeebullah Adua and Mahammad Hammad Uddin, helping with design and modeling of the CMVS system. Most of all, we are sincerely grateful to Dr. Maqsood Ali Mughal for advising this project, providing us with materials, and for the insight and guidance which facilitated the completion of this project. We would also like to thank William Appleyard, the Technician in the Electrical and Computer Engineering department, for facilitating the acquisition of components for the project. Furthermore, we would also like to thank the technical support team at Eversource for their continuous feedback on the project.

Abstract:

Transient weather conditions often decrease the deliverable output power of photovoltaic (PV) arrays and threaten the ability of the PV array to provide power to meet load demands, which leads to utility companies to provide backup energy sources to supplement the remaining demand. The switch from PV to backup power is not instantaneous, so it is important to predict the PV system output power to prevent energy shortages. We refined an ambient light sensor system to predict this output power. The system uses measured irradiance to compute cloud motion parameters such as cloud size, speed, and direction, to predict changes in power generation due to cloud cover. The system will enable grid operators to better understand the effects of PV power variability on the grid.

Executive Summary:

Around the world, distributed PV power generation systems are being deployed at a rapid pace. This is causing technical problems across the utility sector as reverse power flows and voltage fluctuation occurs more in distribution feeders. There is also a rise in real and reactive power transients that affect the operation of the bulk transmission system. Traditional voltage control devices such as line voltage regulators or switched capacitor banks can alleviate slow-moving fluctuations, but these devices need to operate more frequently than usual when PV generation fluctuates due to cloud cover. For example, the output of PV systems can drop from 100% to 20% in a matter of seconds and return back to 100% within the same time frame. Utility companies fear that frequent operation will impact the life expectancy of voltage control devices [1, 2]. In order to fully understand and address these problems, utility companies like Eversource are seeking solutions to this problem.

This work is a continuation of the work done by a group in the previous year. The goal of the project was to design and construct a light sensor array to detect and analyze cloud cover approaching a PV site. This device would enable utility companies that make extensive usage of PV generation in their grid networks to have a more reliable and consistent output as they would be able to predict future power generation given forecasted cloud cover. The last team set the foundations of the project including code, circuit diagrams, and knowledge that allowed us to quickly determine what needed to be improved. This system utilizes an array of nine separate light sensors to forecast the direction a cloud is traveling, the time it would arrive at the PV site, and predict the power generated by the PV grid. Eight of the sensors are placed around the system with the ninth sensor being placed in the center of the device, which forms a Cloud Motion Vector Sensor (CMVS) system. Our goal this year was to revise the previous year's sensor system design, improve the code, and test at a PV site.

We have designed and implemented several improvements across all parts of the CMVS system. The structure of the physical array was modified to improve its sturdiness. The weatherproofing of the system was improved using tangle and water-resistant wire sheaths to protect the wires. These were connected to a central enclosure that protected the Arduino microcontroller and central light sensor, as well as the eight radial sensor enclosures. Additionally, snap-fit enclosures were designed in Solidworks and 3D-printed to allow for rapid assembly.

Although the foundation of the electrical system remains relatively unchanged from the design used in the past, it has been improved and iterated upon. We have gone through several stages of prototyping the electronics. This ranged from basic breadboarding to experimenting with several potential PCB layouts and configurations in order to find an ideal medium for this project. This process has enabled us to create a recommended PCB for further revisions and improvements to the project.

The software element of the model also received significant changes. Initially, the MATLAB algorithm had errors within it. These errors were debugged, and repetitive, bulky elements of the code were reduced and simplified to improve the overall efficiency of the model. We also added several quality-of-life changes with features such as an added progress bar and

documentation of all code being incorporated. The newly added computations to obtain additional cloud parameters were first incorporated into MATLAB and then these changes were added into ThingSpeak: the online IoT platform that we use to process our collected irradiance data. The newly implemented parameters were estimated time of arrival, cloud size, and depth. These parameters, in addition to the date and time that the histograms were computed, was all done in real-time.

Determining the predicted power output required the use of additional software systems. The initial method to determine the predicted power output relied on calculating the alpha values during the day across the span of year to create a database of alpha values at known irradiance values. This would enable easy determination of predicted power by referencing the alpha values. We shifted to a different method involving Simulink because product delays made the initial plan implausible within our timeframe. This Simulink model has two inputs, irradiance and temperature, and outputs the predicted power. For this model, a pyranometer and temperature sensor were required at the PV site and were connected to a separate microcontroller from the one used in our CMVS system. These sensors did not support direct interfacing with the Simulink model, so Python was used as an intermediate interface between the two. Currently, the predicted power projection model needs to be further refined.

While many improvements were made over the course of this project, there still is ample room for future refinement of the system. The system is in need of better weatherproofing to enable it to function more effectively across temperature ranges. While the electronics are well protected from weather conditions such as rain and snow, the wire connections did not maintain their integrity in sub-freezing temperatures. Additionally, a new more compact and spatially efficient central PCB design is recommended. Finally, there are select segments of the code that could be made more efficient or refined. The Simulink model needs to be updated to be able to integrate into the rest of the system, and the efficiency and annotation in the MATLAB code should be reflected in that on ThingSpeak. It should be noted that the cloud size and depth have not been retroactively implemented into the MATLAB code.

Table of Contents

Acknowledgements: 1

Abstract: 2

Executive Summary: 3

1 Introduction 7

1.1 Problem 7

1.3 Current Solutions and Technology 7

1.4 Our Solution and Technology 8

2 System Design and Modifications 9

2.1 Mechanical Design 10

2.2 Electronics Design 14

2.3 Software Design 19

3 Results 23

3.1 CMVS Model 23

3.2 I-Corps Program 26

4 Conclusions 28

4.1 Recommendations 28

4.1.1 Hardware Recommendations 28

4.1.2 Electronics Recommendations 29

4.1.3 Software Recommendations 30

Bibliography: 31

Appendices: 32

Project Code: 32

The Revised MATLAB Algorithm: 33

Python Scripts for Power Output Prediction: 40

MATLAB Script for Power Output Prediction: 41

Arduino Script for Power Output Prediction: 42

Revised ThingSpeak Algorithm: 43

Recommended PCB Schematic: 52

Additional Figures: 53

Table of Figures:

Figure 1: 9
Figure 2: 10
Figure 3: 11
Figure 4 13
Figure 5: 14
Figure 6: 15
Figure 7: 16
Figure 8: 17
Figure 9: 18
Figure 10: 19
Figure 11: 22
Figure 12: 24
Figure 13: 25
Figure 14: 26
Figure 15: 29
Figure 16: 30
Figure 17: 52
Figure 18: 53
Figure 19: 53
Figure 20: 54
Figure 21: 54

1 Introduction

As the environmental drawbacks of fossil fuel reliant forms of power generation become more apparent, alternatives such as renewable energies become more attractive as long-term investments. Photovoltaic systems are limited by weather, such as cloud cover, PV irradiance, and humidity. These can impede the maximum power output of a PV array. Thereby potentially causing the PV array to fall below the load demand of the local grid. Our work is a continuation of the work done by a group in the previous year. The goal was to design and construct a light sensor array to detect and analyze cloud cover approaching a PV site. This array would forecast the direction a cloud is traveling, the time it would arrive at the PV site, and predict the power generated by the system. The previous group produced a baseline sensor array and code to run it. Our goals for the project were to improve the array design, the code, and the accuracy of the model. Additionally, the model needed to be tested at a PV site.

1.1 Problem

Around the world, distributed PV power generation systems are being deployed at a rapid pace. This is causing technical problems across the utility sector as reverse power flows and voltage fluctuation occurs more in distribution feeders. There is also a rise in real and reactive power transients that affect the operation of the bulk transmission system. Traditional voltage control devices such as line voltage regulators or switched capacitor banks can alleviate slow-moving fluctuations, but these devices need to operate more frequently than usual when PV generation fluctuates due to cloud cover. For example, the output of PV systems can drop from 100% to 20% in seconds and return back to 100% in the same time frame [1]. The utility sector fears that such frequent operation will impact the life expectancy of these voltage control devices. In order to fully understand and address these problems, utility companies like Eversource are seeking solutions to this problem. Such a solution would require extensive computer simulation and data analytics studies to properly address the issue [2].

For further perspective, Massachusetts alone has had a large increase in its integrated PV power within the last decade. The state possessed roughly 1000 MW of PV power generation in 2015 and had the goal of reaching 1600 MW by 2020. As of 2019, the state has had a cumulative capacity of 2752 MW, which is more than double what it had only four years before [3]. This further displays the need for systems that can improve the resiliency of PV systems. As the grid becomes more dependent on PV arrays, loss of power due to cloud cover becomes more impactful and adds more stress to the system.

1.3 Current Solutions and Technology

Throughout our research we have encountered several contemporary solutions to the problem identified. All of them in some way serve to forecast weather or monitor outputs from a

PV system. One possible solution is the Total Sky Imager (TSI) product, which periodically takes photos of the sky to measure how much cloud cover there is. Preliminary testing has been done on its application as a tool for short-term solar irradiance forecasting [2]. This could in the future prove useful for managing PV systems, but TSIs are not widely installed, with only a few sites established around the world. Another solution, the Smart Monitoring Device (SMD), utilizes several algorithms to connect PV arrays into an Internet of Things (IoT) system [3]. This product also uses irradiance sensors combined with image capture to forecast cloud coverage. This product is the most similar to our own that we have seen so far. One final solution we investigated is the Cloud Shadow Positioning System (CSPS). This system uses a stationary camera to take pictures of passing clouds. These pictures are then analyzed by their cloud detecting system, called SIFT, by selecting various points on the clouds and comparing their position to the next image captured five minutes later. The estimated arrival time is then computed based on the clouds' speed [4].

1.4 Our Solution and Technology

The major drawback of all of these current solutions is the cost of implementation. Additionally, some of these contemporary solutions are only used in a small number of locations or are still in the testing phase. In contrast, our CMVS system will be easier and cheaper to integrate into pre-existing arrays and is capable of capturing irradiance data using an array of light sensors at high resolution and measuring solar irradiance with high accuracy. The model can also produce high accuracy irradiance values due to its close proximity to the PV array and is favorable for short-term solar forecasting. These measurements are conducted much faster than what is traditionally possible, only taking two seconds at its fastest, compared to approximately 15 minutes. Eventually, our light sensor array will be marketed as an easily installable DIY-kit, which removes the need for a third party or technician to install. The CMVS Model can also be integrated into Synergi, a distributed feeder simulation software used by the majority of Utility companies. The addition of shape, size, and speed of clouds into the system will be useful in making real-time decisions to prevent frequent switching of voltage-controlled devices. As mentioned previously this frequent switching may shorten the life expectancy of these devices and ultimately increase operating costs.

2 System Design and Modifications

The CMVS system can be divided into three sections: the hardware section, the electrical section, and the software section. The hardware section concerns the enclosure that protects the electronics from the elements as well as the other steps we took to properly weatherproof our design. The electrical section details the components used. While the Software section details the improvements that were made to the algorithm. The diagram representing the CMVS can be seen below. The physical construction of the array constitutes the section on the far left, while the software side of the project is the three sections on the right. The real-time data collection and conversion consists of the Arduino code uploading data to ThingSpeak. The weather data analysis section processes the collected data and outputs the generated parameters. The visualization and energy management section involves the process of automatically updating the histograms on ThingSpeak. It also accounts for the future work of connecting up a generator to the system, should PV generation fall beneath load demands.

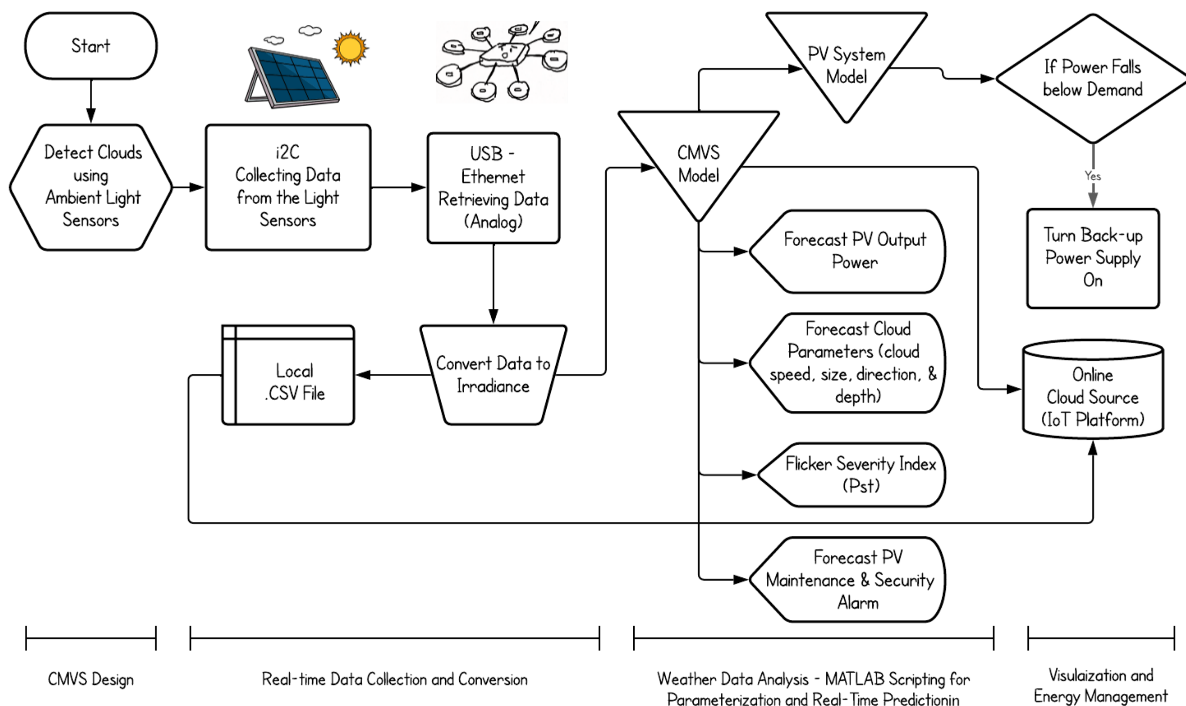


Figure 1. High-level block diagram of CMVS system concept.

The figure below displays the PV array as it is connected to the LED signboard as a load, as well as the hardware configuration for the pyranometer used in predicted output power computation. These electronics were configured at the site, on top of the East Hall Parking Garage. The LED signboard was configured and programmed within the lab to display various short animations, then was transported to the site. The Pyranometer was used to obtain the irradiance values at the PV panels, these values were then inputted into the Simulink model used to determine the predicted power output.

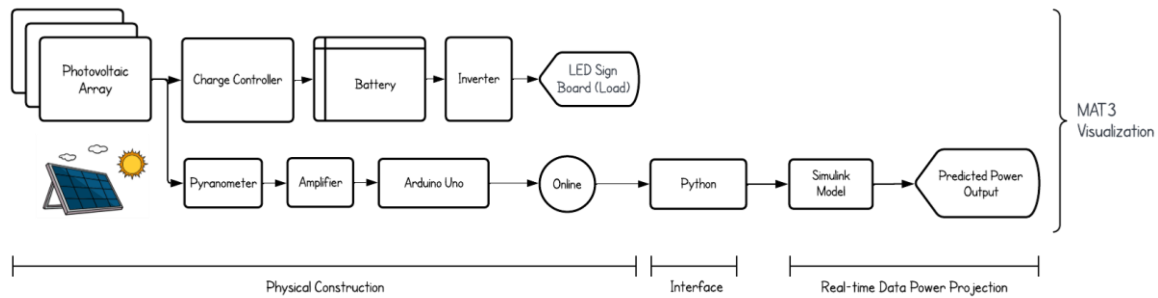


Figure 2. Block diagram of the PV system and power prediction model.

2.1 Mechanical Design

As this project is a continuation from a previous year, most of the physical layout of the system had been designed in advance. The inherited design relies on a total of nine irradiance sensors and a central microcontroller and is elaborated upon in the electrical subsection. While the basic design remains largely unchanged from previous iterations, heavy emphasis was placed this year on designing and constructing a more rugged system that could withstand non-ideal weather conditions, improper care, and repeated usage.

2.1.1 Wire Connectors & Connections

One major issue that we encountered throughout working on the various prototypes of the CMVS system were the wire connectors. These were the connectors between the wires of the irradiance sensors and those from the central PCB. In the prototyping phase, prior to designing a PCB, the array was constructed on a breadboard. This was done to assure that the connected sensors functioned properly, because of this connection pins were mounted onto the stripped ends of the wires. In later prototypes, Molex wire-to-wire connectors were implemented into the design to aid in the construction, deconstruction, and transport of the array. These connectors are waterproof, insulated, and weather resistant snap joints between wires. These were configured at the central PCB such that it could be easily isolated from the sensors for transport. These Molex junctions connected the wires that were mounted to the PCB to the 12-foot-long cables to the sensors. Between all wire-to-wire connections involving a change in wire gauge, heat-shrinking was used to provide additional security. The tips of the adjoining wires were stripped then joined within the heat-shrink sleeve.

As mentioned, the connection that provided the most difficulty was the wire connectors between the PCB and the Molex connectors. These connectors were also the one that changed the most across different iterations of prototypes. Initially, the wires from the Molex were soldered

directly to the port holes on the PCB. In a later prototype, the design was changed to incorporate screw terminal connectors onto the PCB. These screw terminals were soldered to the board and operated by inserting the adjoining wire into the terminal, then fastening the wire in place with the screw. These terminals were challenging to work with because wires kept slipping out, snapping, or refusing to connect at all. This was especially frustrating and cumbersome when testing in below freezing weather conditions. This could be attributed to the lack of dexterity and fine control of the wires and terminals. This led to the desire for an improved wire connector such as the type discussed in “Section 5.1 – Recommendations”.

2.1.2 System Weatherproofing

The system’s weather resistance was improved with the addition of tangle resistant wire sheaths and quartz glass panes. This weather proofing is critical to ensure that the electronics do not become damaged due to inclement rain or snow. That said, on days where the conditions were this poor, we still packed up the array and terminated the testing. Each sensor required a pair of I2C cables to properly transmit the data and receive power.

Figure 3. Light sensor in 3D printed enclosure connected to wires with ribbed sheathing.

The introduction of the black tangle resistant wire sheaths provided two benefits. One of which was that the wires connecting the sensor enclosures to the central enclosure became better protected against rainfall, and the tangle resistant quality prevented the array from becoming an unusable nest of wires when stored away.

The transparent quartz panes were obtained to protect the formerly exposed sensors from potential non-ideal weather conditions. The PCB enclosure was designed such that the panes could be mounted over the light sensor to weatherproof it.

2.1.3 PCB Enclosures

3D-printed enclosures were designed to protect the sensitive electronics in conjunction with quartz glass. Like the previous year, enclosures for the central microcontroller and the radial sensors were designed in Solidworks. However, we significantly improved these designs relative to the previous year. Naturally, there were a few prototypes prior to the finalized designs shown below. In designs, unnecessary material in these enclosures were eliminated, they became snap-fit enclosures, and stabilizing arms with screw holes were implemented.

Separate designs were created for sensors that are positioned at the circumference of the array compared to the central enclosure holding the PCB and the ninth sensor. As mentioned, the shared characteristics of these models include a snap-fit closure between the cover and baseplate, wire and pin slots in the cover, as well as screw holes located on the baseplate. As seen on the figure above, there is a slot in the walls of the enclosure that allows the wires to connect to the sensor. These screw holes at the end of radial arms allow the enclosure to be securely mounted to a piece of wood, elevating it off of the ground. This then provides additional weatherproofing against rain and snow. The following figures show the 3-D printed enclosures that were designed:

Figures 4a, 4b, 4c, 4d. Solidworks models designed for the CMVS system, with dimensions in millimeters.
Figure 4a (top left) - The cover of the plastic sensor enclosures.
Figure 4b (top right) - The spacer that holds glass above the sensor on enclosure cover.
4c (bottom left) - Base of plastic sensor enclosures.
4d (bottom right) - Fully assembled model of the light sensor enclosure.

2.2 Electronics Design

The electrical system allows the CMVS system to collect the irradiance values needed to predict cloud cover. A diagram of the electrical system can be seen below in figure 5.

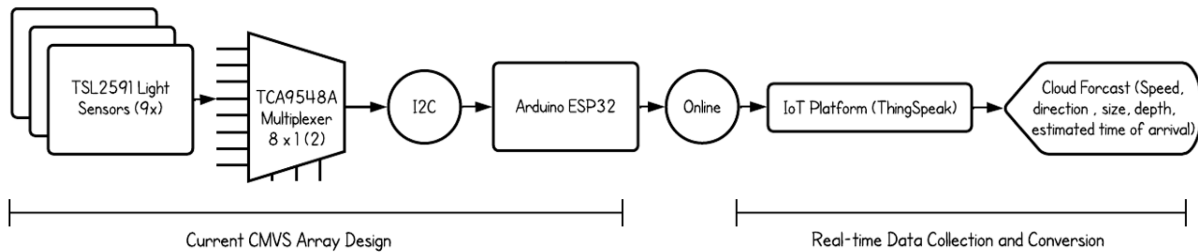


Figure 5. The block diagram of the CMVS electronics.

Nine individual TSL2591 light sensors are used to collect irradiance values at different points around the CMVS system. One light sensor is placed in the center, while the remaining eight light sensors are placed twelve feet away from the center in 45º increments from one another radially. These light sensors are connected to a central Arduino Huzzah32-ESP32 Feather microcontroller with the I2C protocol. As their I2C address is unalterable, a TCA9548A I2C multiplexer is used to assign the light sensors different addresses. In the past, two separate multiplexers were used to change the addresses of all nine sensors. We discovered it is possible to use a single multiplexer to change the address of eight sensors and maintain the default I2C address for the ninth.

The TSL2591 light sensor was specifically chosen as it is a direct improvement from the now-outdated TSL2561 that was utilized in the design made by the previous year's team. Whereas the TSL2561 had a dynamic range of 0.1 to 40k Lux, the TSL2591 has a dynamic range of 188u to 88k Lux. This radical improvement in sensitivity for sensing lux values allows for the CMVS system to collect more accurate values that can improve the quality of predictions made.

The Arduino Huzzah32-ESP32 Feather microcontroller was specifically chosen for its WiFi connectivity and battery charging capabilities. The lux values collected by the Arduino are sent to an IoT platform, i.e., ThingSpeak, where they are processed to predict incoming cloud cover. This step requires the system to have an internet connection, which the ESP32 Feather can support. Additionally, this board has built-in battery charging capabilities that allows it to be powered off of a LiPoly battery which improves the portability and reliability of the system. This allows for testing to not have to depend on proximity to an external power source as it can be powered by an included battery. The microcontroller used by the previous group was the BeagleBone Black, which introduced unnecessary additional complexity and lacked these additional features.

2.2.1 PCB Prototypes

Initial testing of the CMVS system was done on a breadboard. While the previous year's team left behind an intact model for our usage, it proved too fragile and entangled for us to use for testing. The following figure shows the schematic of last year's CMVS system that was followed in assembling that initial breadboard:

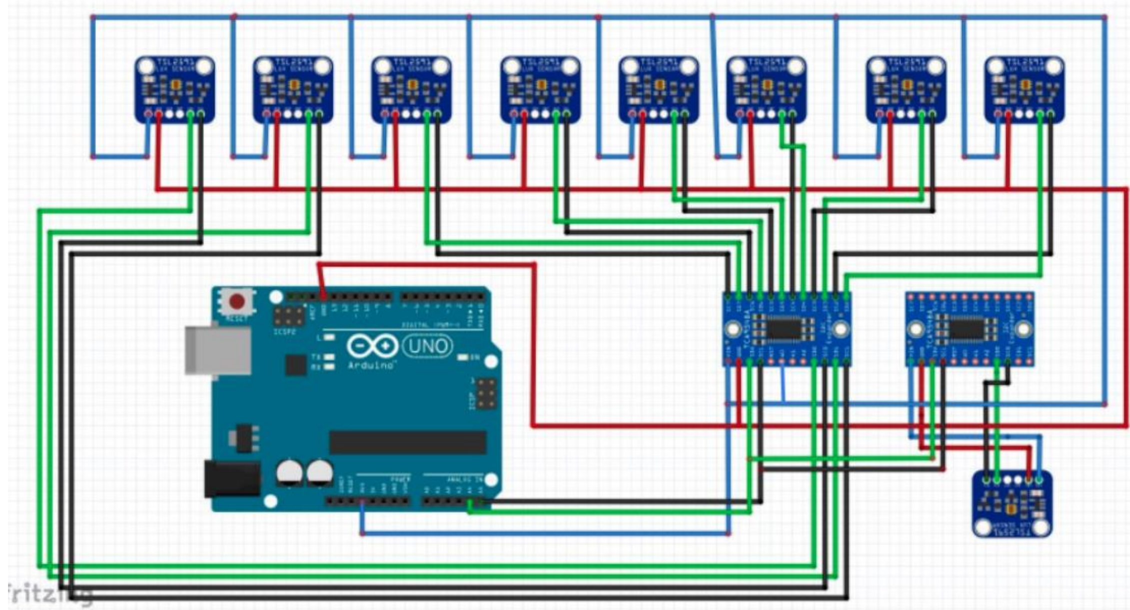
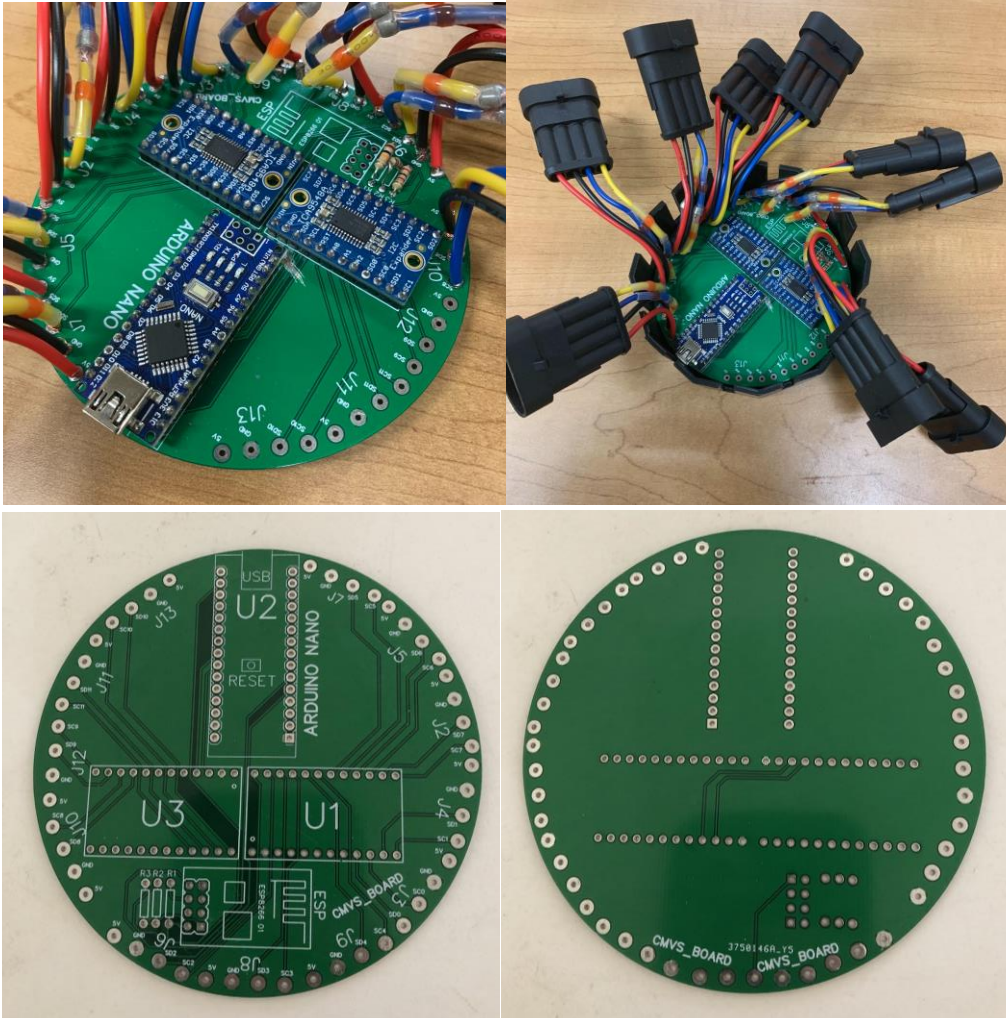


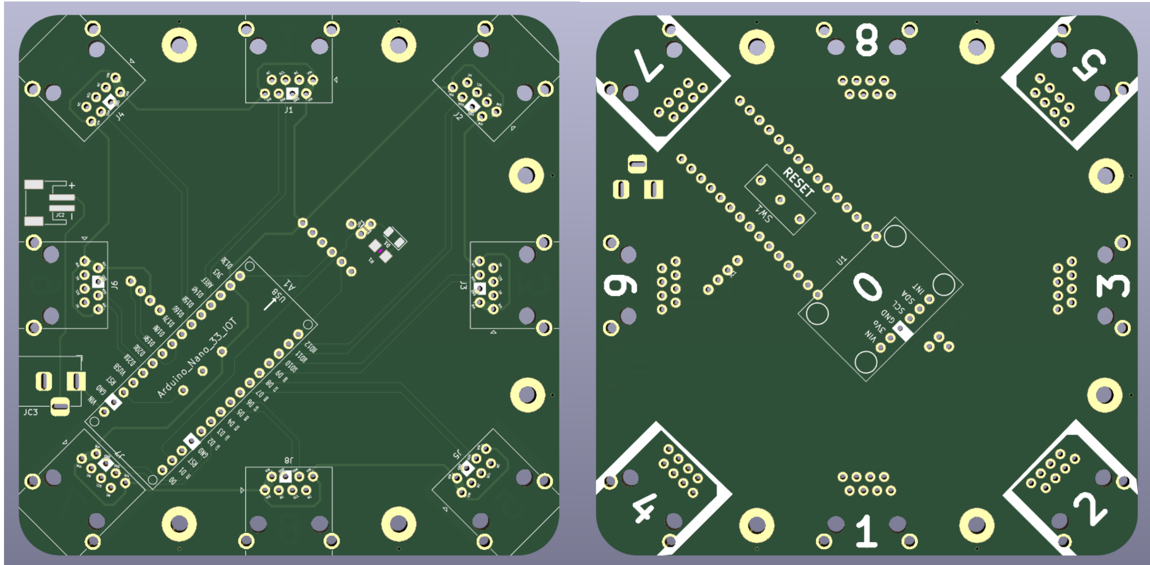
Figure 6. The circuit schematic for Arduino Uno with nine light sensors and two multiplexers.

The first breadboarded system proved to function well and allowed us to collect cloud cover data. While this breadboarded prototype worked well initially, it proved cumbersome and frustrating for long term usage. In order to resolve this, a graduate student that was working with us on the project designed a PCB for us to use instead. It is functionally identical to the breadboarded system as it follows the electrical schematic shown in Figure 6. This PCB is shown in the following figure:



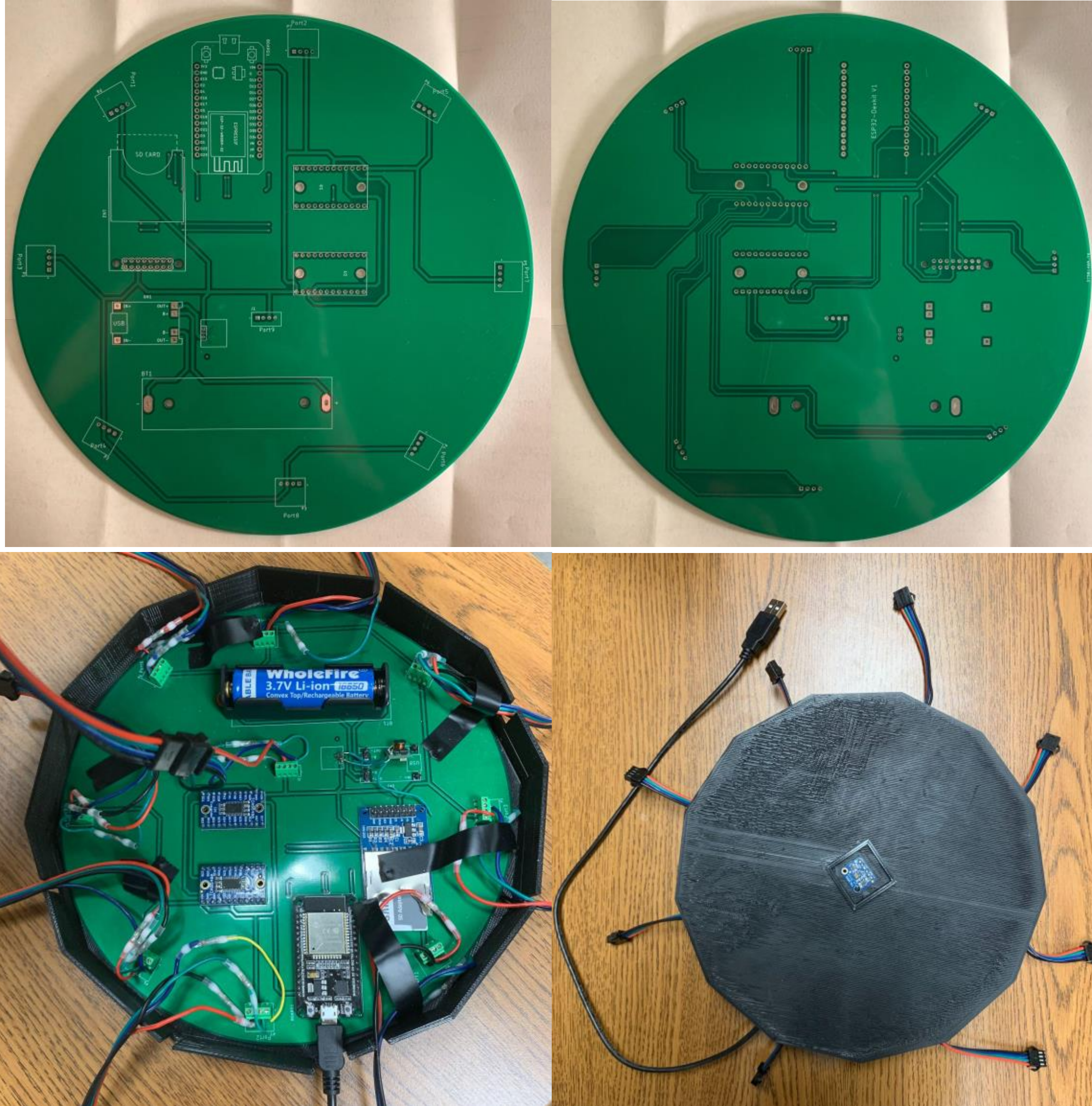
Figures 7a, 7b, 7c, 7d. The first PCB iteration that was designed.
Figures 7a and 7b (Top) – PCB with mounted components, in and out of the central enclosure.
Figures 7c and 7d (Bottom) - Front and back of the blank PCB.

After assembly, this PCB was much better suited for long-term testing as the connections were much sturdier than the previous breadboarded system. However, there were some issues with the design as the WiFi module was nonfunctional and a misplaced connection had to be manually scratched out to disconnect it. The undergraduate team designed another PCB to fix these issues. Like the other system prototypes, this new PCB is functionally identical to the original schematic given in Figure 5. This new PCB is shown in the following figure:



Figures 8a and 8b. Front and back of RJ45 PCB.

This new CMVS system PCB resolved many of the shortcomings of the previous PCB by removing the faulty connection, using a WiFi-enabled microcontroller, and using RJ45 connectors to connect the light sensors to the Arduino. The usage of RJ45 connectors allowed for the connection from the light sensors to be fully weatherproofed and easily connectable to the CMVS system. While this solution worked well at first, issues arose as the light sensors would unpredictably disconnect after periods of extended use. Due to time constraints, we did not have the time to properly debug this issue and chose to use another PCB designed by the graduate student who designed the PCB shown below:



Figures 9a, 9b, 9c, and 9d. Large PCB wire screw terminals designed by graduate student.
Figures 9a and 9b (Top) - The top and bottom of the blank PCB, respectively.
Figures 9c and 9d (Bottom) - The assembled PCB in the scaled-up enclosure.

While improving the issues faced with the PCB shown in Figure 7, this PCB had several issues which hindered its adoption. The external battery did not work, the WiFi module was still nonfunctional, and the SD card that was intended to be used for storing collected data locally ended up being unnecessary as all our collected data was being uploaded to ThingSpeak. After these multiple attempts to have a functional PCB, we decided to go back to the simple breadboarded CMVS system as it allowed us to quickly make changes if something broke or was not working as expected. The updated breadboarded CMVS system can be seen in the following figure:

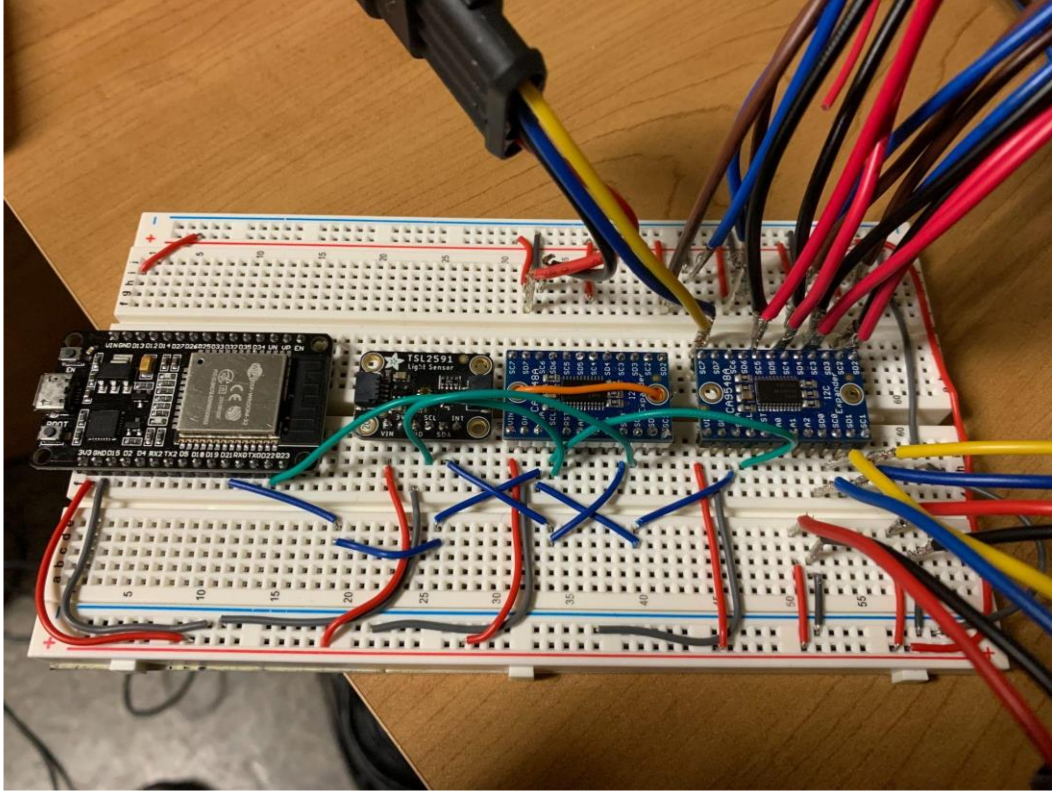


Figure 10. The breadboard design used for testing when wires could not be connected to a PCB due to cold weather.

Despite being one of our simplest designs, the updated breadboarded CMVS system performed extremely well and offered us the most flexibility with collected cloud cover data. We did not have to deal with unwieldy wire connectors, and we were able to make quick adjustments to the system as needed. Additionally, we made the switch over to the Arduino Huzzah32-ESP32 Feather microcontroller board as it was WiFi-enabled and had built-in battery support. While lacking the convenience and ruggedness of the other PCB solutions, its performance was much more reliable and predictable.

2.3 Software Design

The software for this project can be divided into three distinct categories: data collection, data processing for visualization, and simulations. The data collection is done locally on the CMVS array that we tested. The CMVS system uses an Arduino C script to send collected light intensity values from the sensors via the microcontroller to ThingSpeak, an online analytics service that allows uploaded data to be remotely aggregated, visualized, and analyzed. Once the collected data about the current cloud cover is on ThingSpeak, it is then processed through two different separation methods to determine the forecasted parameters regarding the incoming cloud cover. The code can be locally run through MATLAB; however, it does not have the auto-update functionality that ThingSpeak has. Please note that all code used in this project can be found in the Appendices.

2.3.1 Data Collection

The Arduino code primarily served to initiate the microcontroller such that data sampling could be conducted. Initially, the primary usage of the Arduino code was to confirm the proper configuration of the sensors, which was assured from viewing the output data.

The Arduino code was not modified extensively. As mentioned, the functionality of this code was limited to collecting the light intensity values from the nine sensors and then to automatically upload this data to ThingSpeak. This was able to reliably occur in two second intervals. Because of this code, large quantities of data could be measured and operated remotely. Given the location that the array was operated in was just beyond the range of WPI's wireless network, an issue that was encountered was that the microcontroller was unable to access Wi-Fi. This was solved by creating a cellular hotspot at the site.

2.3.2 Cloud Cover Prediction

The MATLAB code was the origin for the other types of code used in the project. It was also the version that experienced the greatest change due to the revisions made. Although MATLAB was the starting platform, the project desired automatic updates which could not be provided directly, so ThingSpeak was used. This was a simple enough change since ThingSpeak is also owned by MathWorks therefore the MATLAB script could easily be integrated. MATLAB and ThingSpeak are used to process the light intensity values such that the cloud cover parameters and predicted output power can be calculated.

The MATLAB code was thoroughly examined for a better understanding of how the cloud motion vector speed and direction were being calculated. This was necessary, as it initially produced errors when run. Furthermore, the MATLAB scripts were optimized and refactored to make it more maintainable, easier to build additional features, and to debug any glitches that may occur. One feature that was added was a progress bar. The inclusion of the progress bar enables the operator to be able to monitor the status of the scripts while the gradient matrix video is generated. In this step, the length of time to complete the computations scales proportionally with the number of data samples used.

Improvements were also made to the generation, labeling, and saving of the histograms produced by the code. The most prominent modification to the code was to include titles and labels to the axes to the histograms, as these would provide more ease in understanding the generated outputs. Additionally, the histograms were saved in separate image files and were understood to be representative of two separate ways of calculating the probability of shadow direction. The following revisions to the graphing of the polar histograms were made to aid in the understanding and comparison of these two models: both methods were graphed on the same histogram, each method was graphed individually on a smaller histogram the side for better scaling should there be a large discrepancy between the two models, and the three histograms described above were placed on tiled layout on the same image. It should be noted that both methods of the MATLAB

script use a Savitzky-Golay filter to smooth the data samples and use a sliding window to parse through the entire data set. The data set is then normalized with a method based on probability, placed into respective bins, and then graphed on the histograms. The differences in the methods are as follows. Method One specifies the maximum number of bins to be 10, while not specifying the angle of these projections. In other words, each bin can be considered a “slice of a pie chart” on the histogram generated. While Method Two provides the fixed edges to these bins. Meaning a fixed width and orientation for a given sector area should it be populated with a cloud's direction. The generation of the histograms were intended to show the probability that the cloud’s shadow is moving in any given direction. The histograms have been changed to have the normalization method based on probability. In this case, the radial axis is scaled from zero to one. This greatly increases the clarity of the information that was conveyed and is the case for the MATLAB script. When integrated onto ThingSpeak, the graphs are scaled proportional to the maximum values within the data set, rather than the fixed graphical range on MATLAB. Additionally, the calculated direction, speed, and time of arrival of the cloud to the PV array is printed at the bottom of the tiled histogram. For easy reference, like the calculated values, the date and time that the algorithm was computed at is also printed on the histogram.

2.3.3 Power Output Prediction

As the goal of the CMVS system was to use predicted cloud cover as a way to predict the generated power output of a PV array, we also need to find a way to predict that output power. One of the graduate students who was working with us created a Simulink model for our usage that could take the current irradiance and temperature data collected from the solar PV site and process them to get the predicted power of the PV array as an output. The figure below shows what the model looks like:

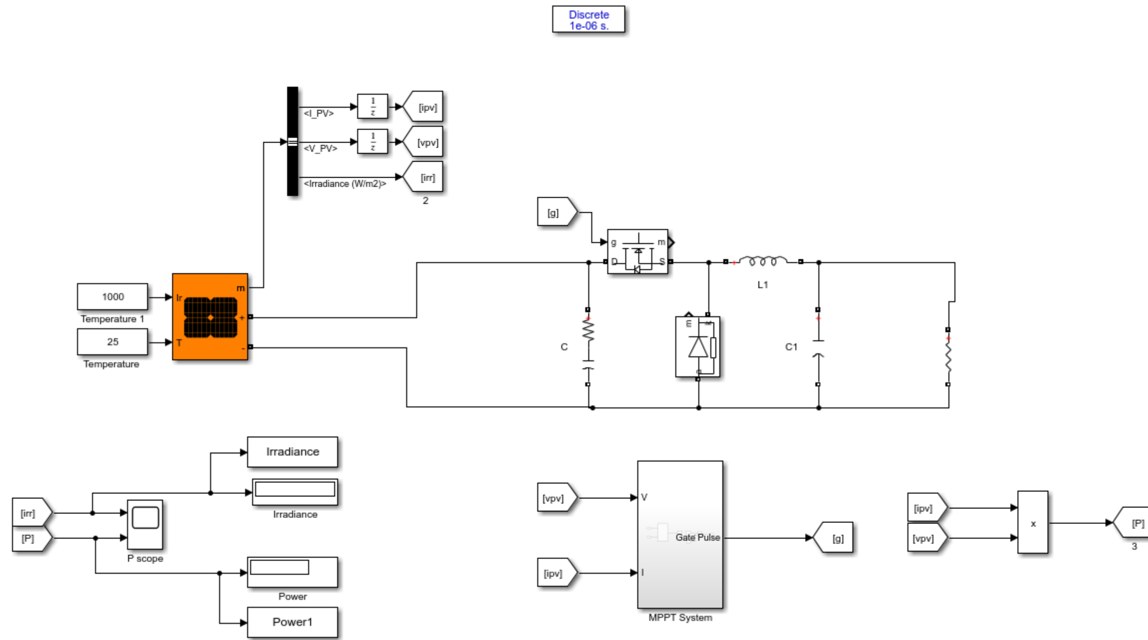


Figure 11. A sample of the Simulink model used to calculate predicted output power. The two blocks on the left connected to the orange block are the inputs. Both blocks continuously report a constant value, while the top block provides irradiance at 1000 W/m^2 and the bottom provides temperature at 25 °C.

The PV system simulation is developed in Simulink using MATLAB functions. The PV system consists of a PV module, DC-DC converter, maximum power point tracking (MPPT) algorithm, and load. The power is supplied by a PV model using irradiance and ambient temperature measured at the PV array. We used a pyranometer LI-200R for irradiance input and Temperature Sensor LM 35 for Temperature input. The function of the MPPT algorithm is to calculate maximum power at a given PV irradiance and temperature. We have used the Perturb & Observe MPPT algorithm for maximum power tracking. A Buck Converter is used as a DC-DC converter. A pulse width modulator set to 5 kHz was used to generate the duty cycle the model depends on. The Buck Converter is used to step down the PV voltage to the battery or load. This simulation model has been tested and verified in literature [7,8].

In order to have inputs that reflect real-world conditions, the undergraduate team built a second system that utilized an Arduino Uno along with specific types of sensors to collect these values. After analysis into which sensors would work best for our specific application, we settled on the Adafruit DHT22 Temperature-Humidity Sensor and a LI-COR 2420-BNC Light Sensor Amplifier. The DHT22 uses two internal sensors, a capacitive humidity sensor and a thermistor, to measure the surrounding air and then provide a digital signal detailing measured values. This specific sensor was chosen due to the amount of documentation on its usage, as well as its widespread adoption in hobbyist projects. As for irradiance sensing, we chose the 2420-BNC due to its compatibility with the LI-200R Pyranometer that we use at the testing site to measure global

solar radiation. The 2420-BNC converts the current (μA) signal from the solar radiation sensor into a voltage that can be measured by a data logger.

Our first attempt to get live data from the Arduino relied on the “Simulink Support Package for Arduino” library to try to interface with the Arduino directly. Despite carefully going through the installation process, we were unable to get the DHT22 and 2420-BNC sensor to work consistently within Simulink. In order to overcome this setback, we chose an alternative workflow that involved logging data with the Arduino, tracking the data going to the COM output port with Python, and then running the Python script within a MATLAB script that Simulink could interface with. While this approach worked initially and we were able to get collected data from the Arduino system all the way to MATLAB, we were not able to make the final step into Simulink due to the script relying on a function that Simulink did not support. While we have addressed this in more depth in Section 5.1.3 Software Recommendations, this is an issue that needs to be resolved in the future.

3 Results

3.1 CMVS Model

As the CMVS system is meant to operate throughout weather conditions, we tested in a variety of weather environments ranging from temperate fall days with minimal cloud cover to cold winter days with little sun. Regardless of the weather conditions, the steps needed to test the CMVS system model were the same. The system had to be laid out in a cleared region near the PV array installed on the top of the East Hall parking garage. This involved shoveling out snow in the winter and working around puddles in the fall and spring to not compromise the integrity of our device or jeopardize the quality of collected data. The eight radial sensors were subsequently placed around the central enclosure containing the PCB and the ninth sensor. Due to the university’s Wi-Fi not reaching this location, the system depended on a cellular hotspot connection such that the Arduino could upload collected data. As the system was able to upload data remotely, we were able to monitor performance and generate histograms while away from the testing site. Although the sensors could reliably send measured data at a two second interval, the team collectively decided to maintain a five-minute refresh rate for the histograms. This was done initially due to a separate hardware constraint, but later maintained as it was estimated cloud cover would not change significantly over the chosen time period.

By conducting the testing stated above we were able to garner some results. These results primarily consist of the histograms generated from the MATLAB and ThingSpeak scripts. The data used to generate these histograms were collected over several days in January and February. This data was collected over the course of three hours from 11:00 AM to 2:00 PM each day during a five-day work week. Figure 16 shows an example of a histogram generated directly in ThingSpeak. The orange and blue sectors show the direction the cloud is traveling based on the two methods explained previously. Additionally, the text at the bottom of these histograms

displays the calculated cloud parameters: the time of arrival, speed, direction in terms of theta, the length and depth of the cloud, and the date and time that this data was computed.

Figure 12. A histogram generated from collected data using code on ThingSpeak.

We downloaded the data and ran it through the same scripts on MATLAB to verify that the code on ThingSpeak was producing histograms correctly. An example of a histogram generated with MATLAB can be seen below in Figure 12. There is a slight difference in the detail and positioning of the text due to minor differences in the functions between MATLAB and ThingSpeak. Additionally, the calculations to compute the cloud size and depth were not retroactively added to MATLAB. The two examples of histograms displayed were not computed using the same data set; however, if the histograms were generated using matching data sets, equivalent projections would be generated.

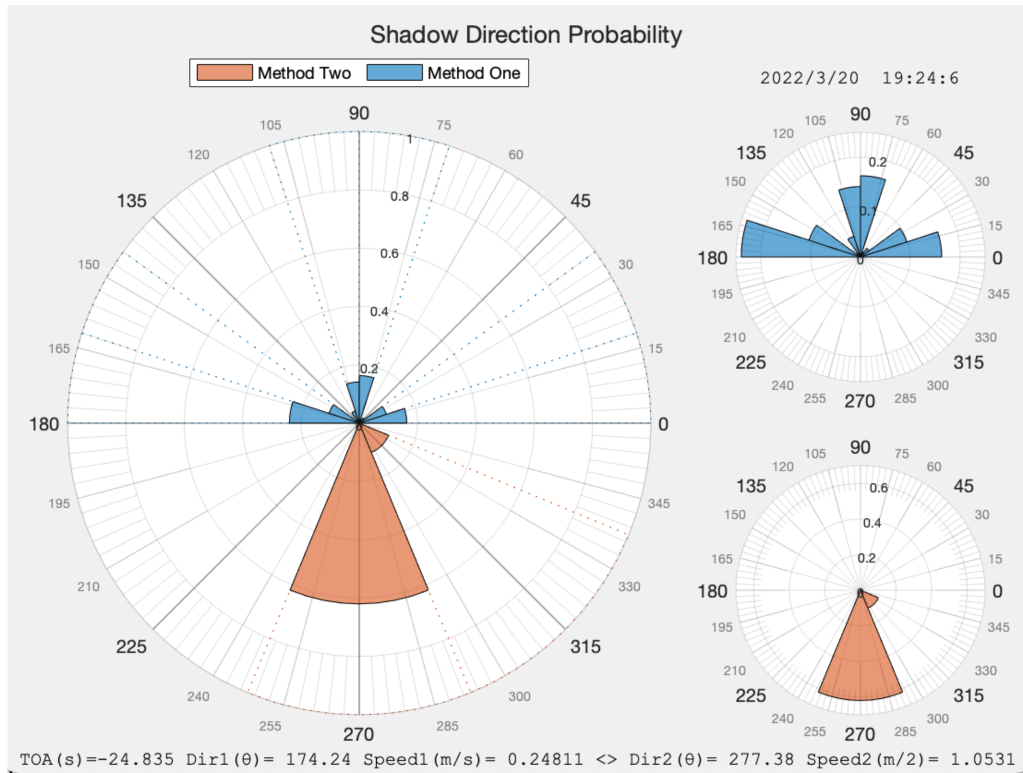
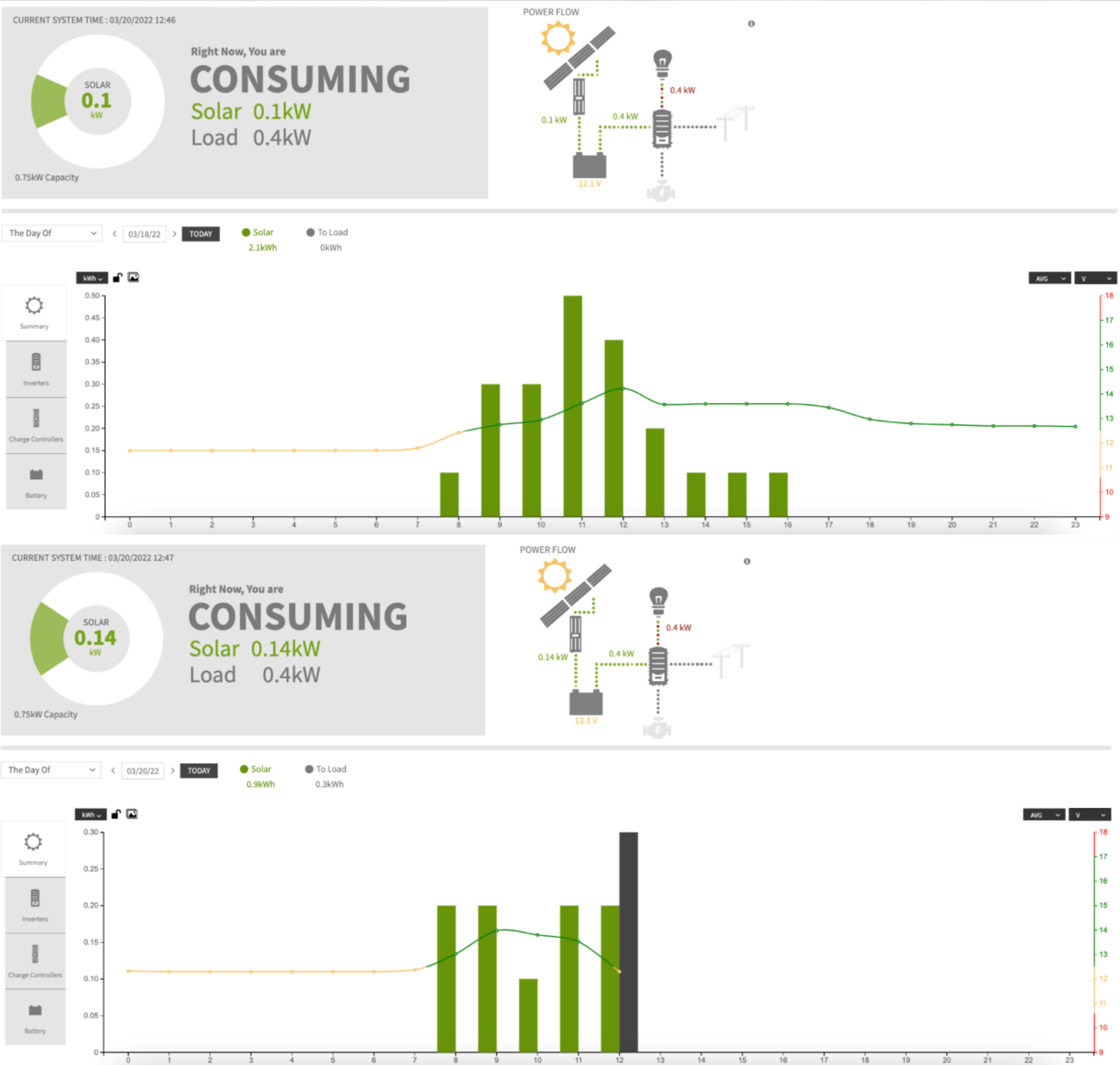


Figure 13. Example of a histogram generated by importing collected data to MATLAB.

When initially verifying if the histograms produced accurate projections, we manually covered select sensors to simulate cloud clover in a particular direction. For example, we would cover the three sensors on the eastern side of the array for 30 seconds, followed by the three in the center column 30 seconds later, then cover three sensors that make up the column on the west side 30 seconds after that. We repeated the process when removing the cinder blocks that were used to cast the shade. This process allowed us to know which direction the "cloud" should appear to be moving in the histograms, and to manually calculate the cloud's speed. Using different variations of "cloud" direction and time intervals we were able to verify that the scripts were correctly performing calculations. It should be noted that a large volume of data was still collected from genuine cloud cover passing above the array.

At the completion of this undergraduate project, the Simulink model used to predict the output power of the area had not been fully functional. However, when this model is completed in the future the resulting calculations can be verified by comparing them to system data shown below. This data is accessible online because of MATE3, a system controller connected to the battery of the PV system. The following two figures display the power generated and consumed on two different days, as well as the current power consumed by the load at the time the image was recorded:



Figures 14a and 14b. Bar graph displaying power generated by the PV system on different dates.
Figure 14a (Top) - Power generated on March 18th.
Figure 14b (Bottom) - Power generated and consumed on March 20th.

3.2 I-Corps Program

One other major result of this project was the work done in I-Corps. Our group opted to participate in the WPI I-Corps program to build upon this previous foundation. The full program takes place over several months and consists of two parts. The first part is a collaboration between WPI and the Massachusetts Institute of Technology (MIT). Two lectures were given by MIT staff and participants were asked to complete 12 interviews over four weeks. Participants also attended two office hours with advisors from MIT to assist with the interview process. This part of the program focused primarily on customer discovery and establishing value propositions. After this

part the remainder of the program was conducted with exclusively WPI associates and sought to reinforce the customer segments found in the previous part. By the end of the program, it was expected teams would complete 30 total interviews and have refined their product idea to the minimum viable product.

We applied to the program with the intention of identifying who our target market would be should our project be commercialized in the future. We also sought to further confirm the need for a product such as ours and gain insight into features necessary to transform our prototype into a marketable product. As of the submission of this project the program is still ongoing, however the interviews conducted up to this point still provide further insight into the issues we wanted to explore.

Early in the program, we identified our primary customer segment to be utility companies that make use of photovoltaic systems. Through speaking to companies ranging from major utilities, residential solar providers operating in the Northeastern United States, and research laboratories that focus on renewable energy and renewable integration, we were able to confirm that utility companies would be our main market.

Throughout conducting interviews, we were able to reach several conclusions and confirm previous research. First, we found that passing clouds can cause issues in delivering power, but not nearly as frequently as we expected. This issue is called Voltage Flicker and is the term for when there is a noticeable change in illumination in equipment caused by a fluctuation in voltage within power systems. While Voltage Flickers do cause problems on the grid, other things need to be considered as well, such as the ramp rate of energy storage. Energy storage was another major topic that came up repeatedly. Having local energy storage at sites provides much faster switching times than some traditional generators. Because of this, we were informed that a lot of research is currently being invested in this area. We also discovered that our project would not be financially viable if marketed towards residential PV arrays. In particular since the sensor array would be an additional cost on top of that of the PV arrays, where the primary function would be to provide a fluid switch to supplemental power. Additionally, we were told any monetary losses caused by cloud cover for home solar were inconsequential compared to large-scale commercial farms.

4 Conclusions

As this project is a continuation of previous work, we had several goals to achieve based on the work previously done. It sought to improve the system design and code, as well as improve the accuracy of the model. The physical design of the sensor array has been upgraded with newly designed, weatherproof sensor enclosures, anti-tangle wire sheaths, and a new PCB design for the central processing hub. The code was cleaned up, so it is easier to read, then added to such that it provides more information about the characteristics of detected clouds. It also updates generated histograms in real time. As for improving the accuracy of the model, more testing and research would need to be done in this area to confirm. More detail regarding recommendations for future work can be found in the following sections.

4.1 Recommendations

Given the many challenges we have faced and overcome throughout work on the CMVS system, we have several recommendations for continuing work on this project.

4.1.1 Hardware Recommendations

One major issue this year's team encountered was how to properly weatherproof the device against the cold. Protecting against weather conditions, such as rain and snow, was easy as individual quartz glass covers were used to shield the light sensors. Additionally, 3D printed enclosures and plastic cord covers safely housed the sensitive electronics and wires. The real issues with weatherproofing were encountered in trying to work around the subfreezing temperatures in the winter.

The large circular PCB that was used for collecting cloud cover parameters depended on screw terminals to connect the light sensors to the Arduino. As the light sensors had to be placed 12 feet away from the Arduino, wires had to be carefully stripped, soldered, and heat shrunk to the light sensors to prepare them for usage. The end that connects to the Arduino was left bare, so that the exposed wire could be connected to the screw terminal. Unfortunately, the extreme winter cold frequently jammed the screw from tightening to the point where a solid electrical connection could not be made. This caused testing to slow down considerably as the team had to find an alternate way to connect the light sensors to Arduino.

We recommend investing in proper connectors that are rated to work in subfreezing temperatures to avoid this from happening in future designs. The female-end of these connectors should be used on the PCB, whereas the male-end should be used in the light sensor. This will ensure system reliability in testing across various weather conditions and temperatures. This will also speed up the time needed to set up the system as the light sensors can be quickly connected to the CMVS PCB.

4.1.2 Electronics Recommendations

The biggest issue that this year's team encountered was confusion on which components to use. Between using multiple multiplexers and working with several different single-board microcontrollers, trying to keep track of all the different components was difficult and frustrating. In order to avoid the same frustrations and complications that we faced, we recommend carefully evaluating the current components available to you and other alternatives that you may use to determine which is best suited for the project. After using many different single-board microcontrollers, we found the Arduino Huzzah32-ESP32 Feather microcontroller to be the best-suited device for the specific application of the project.

This year's team also designed a PCB that incorporates all the improvements we have made to the initial design as a starting point for future hardware development. It uses the suggested Arduino Huzzah32-ESP32 Feather microcontroller, 6-pin weatherproof JST connectors, and a reset button to restore default device operation. The recommended PCB can be seen below and the schematics can be seen in the Appendix.

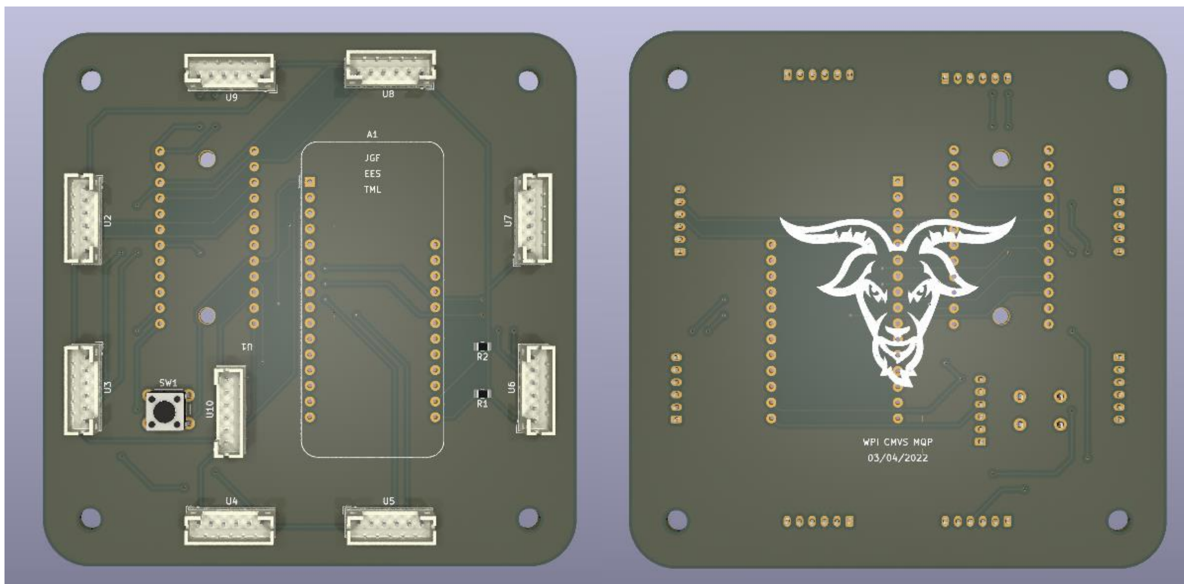


Figure 15. The front and back of the proposed PCB design

Additionally, an electrical block diagram of this proposed PCB can be seen in the following figure:

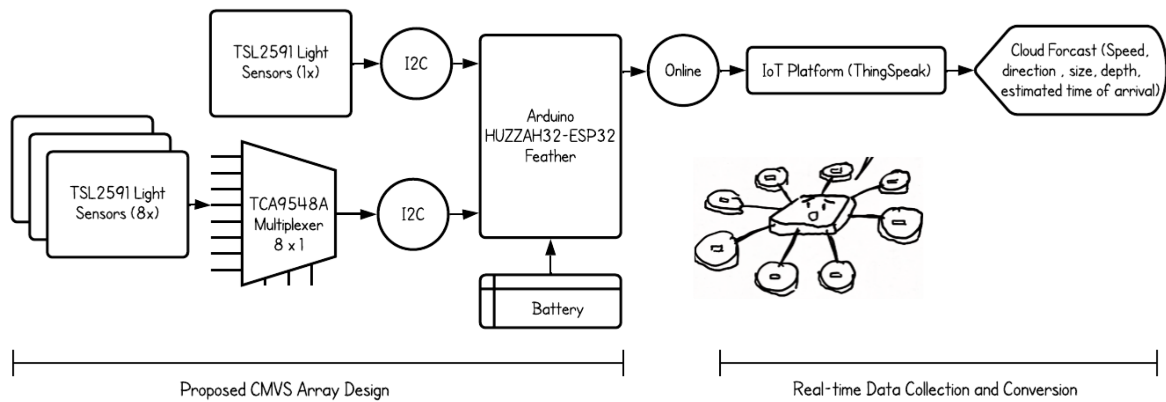


Figure 16. The electrical block diagram of the proposed PCB.

The main difference between this finalized electrical block diagram and the initial one in Figure 5 is the usage of only one multiplexer and the inclusion an external battery. As previously mentioned, the second multiplexer was removed after we realized it's unnecessary and the added battery was used due to compatibility with the new ESP32 Feather.

4.1.3 Software Recommendations

The most critical aspect of the code that needs improvement is the Simulink model. In particular, the segment of the code that is responsible for obtaining live sensor data and implementing this into the predictive power model on Simulink. The workflow that we developed this year did not work completely as intended and we believe that handling everything within Simulink will lead to better results.

Additionally, we propose that the MATLAB code be adapted slightly so that it can be utilized on ThingSpeak, instead of the current version. This is because the MATLAB code is better documented and refined. This would require retroactively implementing the cloud size and depth parameters to the MATLAB code. One additional recommendation for the code is to account for the case when the histograms occasionally display "NaN" for various cloud parameters. This likely is a result of a cloud not passing above the central sensor.

Bibliography:

[1] Rahimi, Kaveh, et al. "Computation of Voltage Flicker with Cloud Motion Simulator." IEEE Transactions on Industry Applications, vol. 54, no. 3, IEEE, May 2018, pp. 2628–36, doi:10.1109/TIA.2017.2787621.

[2] Chengrui Cai, and Aliprantis, Dionysios C. "Cumulus Cloud Shadow Model for Analysis of Power Systems With Photovoltaics." IEEE Transactions on Power Systems, vol. 28, no. 4, IEEE, Nov. 2013, pp. 4496–506, doi:10.1109/TPWRS.2013.2278685.

[3] " Renewable Energy Snapshot: Amount of solar, wind and combined heat and power (CHP) installed in Massachusetts," *Mass.gov*. [Online]. Available: https://www.mass.gov/info-details/renewable-energy-snapshot.

[4] A. Ryu, M. Ito, H. Ishii and Y. Hayashi, "Preliminary Analysis of Short-term Solar Irradiance Forecasting by using Total-sky Imager and Convolutional Neural Network," *2019 IEEE PES GTD Grand International Conference and Exposition Asia (GTD Asia)*, 2019, pp. 627-631, doi: 10.1109/GTDAsia.2019.8715984.

[5] A. S. Spanias, "Solar energy management as an Internet of Things (IoT) application," 2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA), 2017, pp. 1-4, doi: 10.1109/IISA.2017.8316460.

[6] Y. Najera, D. R. Reed and W. M. Grady, "Image processing methods for predicting the time of cloud shadow arrivals to photovoltaic systems," 2011 37th IEEE Photovoltaic Specialists Conference, 2011, pp. 000188-000191, doi: 10.1109/PVSC.2011.6185877.

[7] Motsoeneng, P., Bamukunde, J. and Chowdhury, S., 2019, March. "Comparison of Perturb & Observe and Hill Climbing MPPT schemes for pv plant under cloud cover and varying load". In 2019 10th International Renewable Energy Congress (IREC) (pp. 1-6). IEEE.

[8] M. H. Uddin, M. A. Baig and M. Ali, "Comparison of 'perturb & observe' and 'incremental conductance', maximum power point tracking algorithms on real environmental conditions," 2016 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube), 2016, pp. 313-317, doi: 10.1109/ICECUBE.2016.7495244.lar plant site.

Appendices:

Project Code:

All the code that was used in this project can be accessed on a GitHub repository linked below. The code on this GitHub includes the code from the previous year that has been revised as well as that which was newly rewritten. Within GitHub is the MATLAB, Simulink, ThingSpeak, Python, and the Arduino codes and algorithms.

The code can be downloaded from the following link: https://github.com/Natste/CMVS

```
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <stdio.h>

#include "Adafruit_TSL2591.h"

#define MUX1_ADDR 0x70
#define MUX2_ADDR 0X71
#define NUM_SENSORS 2
#define NUM_CH_PER_MUX 8
#define NUM_MUXES 2
#define RD_WIDTH 16
#define RD_DLY 100

Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591);

void i2cMultiplexSignal(uint8_t addr, uint8_t bus) {
  Wire.beginTransmission(addr);  // TCA9548A address is 0x70
  Wire.write(1 << bus);          // send byte to select bus
  Wire.endTransmission();
}

void configureSensor(void) {
  tsl.setGain(TSL2591_GAIN_MED);  // 25x gain
  tsl.setTiming(TSL2591_INTEGRATIONTIME_300MS);
}

void setup(void) {
  configureSensor();
  Serial.begin(9600);
  delay(RD_DLY);
}

void readSensors(void) {
  char     irstr[NUM_SENSORS * RD_WIDTH];
  uint16_t ir;

  for (uint8_t i = 0; i < NUM_CH_PER_MUX * NUM_MUXES; ++i) {
    if (i == NUM_SENSORS) break;
```

```
    delay(RD_DLY);
    if (i < NUM_CH_PER_MUX) {
      Wire.beginTransmission(MUX1_ADDR);
      Wire.write(1 << i);
    } else {
      Wire.beginTransmission(MUX2_ADDR);
      Wire.write(1 << (i - NUM_CH_PER_MUX));
    }
    Wire.endTransmission();
    ir = tsl.getLuminosity(TSL2591_INFRARED);
    sprintf(irstr, "%6u, ", ir);
    Serial.print(irstr);
  }

}

void loop(void) {
  readSensors();
  Serial.println();

}
```

The Revised MATLAB Algorithm:

```
clear frames paddedData;
clear; clc;
DATA_FILE = '../Data/11-9-sensors-only.csv';
OUTPUT_DIR = '11-9-output';
% OUTPUT_DIR = 'output';
MATRIX_TYPE = 'normalized';
THRESHOLD = 0.03;
PEAK_DISTANCE = 50;
PEAK_PROMINENCE = 0.15;%0.016;
PEAK_WIDTH = 15;
SENSOR_ORDER  = [5 4 6 8 7 9 2 1 3]; % Northwest to Southeast
DATA_ORDER  =   [9 4 8 3 5 1 7 2 6]; % Northwest to Southeast
FILL_ORDER  =   [4 1 8 9 6 2 5 3 7]; % N S W E NE SW NW SE O
CREATE_VIDEO = false;
CREATE_PLOTS = true;
% SENSOR_ORDER = FILL_ORDER;
if ispc % Check to see if operating system is Windows
 DELIMITER = '\';
else % otherwise, use unix-style path delimiters.
 DELIMITER = '/';
end
figure_setup;
load figure_setup SENSOR_STRINGS FIGURE_STRINGS FMT BIN_EDGES SCALE TILE;
if ~isfile(DATA_FILE)
 [DATA_FILE, DATA_FILE_PATH] = uigetfile('*.csv;*.txt;*.dat',...
   'Select Input CSV Data', 'data.csv');
 DATA_FILE = [DATA_FILE_PATH, DATA_FILE];
```

```
end
if ~isfolder(OUTPUT_DIR)
 OUTPUT_DIR = uigetdir('.', 'Select Output Directory');
end
% Read and transfer raw lux data to new array
data = readmatrix(DATA_FILE, OutputType='string');
if ~isempty(regexp(data, '[a-fx]', 'once')) % Check if data is hex
 data = hex2dec(data); % If hex, convert to decimal
end
data = uint32(data);
%%%%%% THINGSPEAK
% i = 0;
% lastMat = 0;
% T = thingSpeakRead(1552033, ... % Get table from TSpeak
%     Fields=1, ...
%     NumPoints=1661, ...
%     ReadKey='AD8ZB04MFD6HIYI8', ...
%     OutputFormat='table');
% dataCol = T(:, {'cmvsData'});       % time & data cols -> data col
% dataCol = rowfun(@string, dataCol); % char table -> str table
% data = dataCol{:,:};                %  str table -> str array
% data = arrayfun(@(x) uint32(str2num(x)), data, ...
%                  uniform=false);     % str array -> uint32 cell array
% data = cell2mat(data);              % uint32 cell array -> uint32 array
%%%%%%
nNotNan  = sum(~isnan(data),2); % count number of valid values in each row
nSensors = round(mean(nNotNan)); % Use mean to get number of sensors
data = data(nNotNan == nSensors, :); % Get rows with a reading for each sensor
data = rmmissing(data, 2); % Exclude any remaining columns that contain a Nan
iLoop = 1;
% iDataStart = 660;
% iDataEnd = 770;
% iDataStart = 1;
% iDataEnd = 101;
iDataStart = 840; % 10 ft/ 30 s -- whole array
iDataEnd = 940;
% iDataStart = 1030 + 24; % 10 ft / 10 s -- whole array
% iDataEnd = 1130 - 24;
iDelta = iDataEnd - iDataStart;
dataWindow = 10; % specifies sliding window length for moving sum
filterWindow = 11; %  specifies smoothing window length
%% Validate, Normalize, and Smooth Data
if dataWindow > length(data)
 dataWindow = length(data);
 dataWindowWarn = sprintf("dataWindow exceeds length of data and has been
trimmed");
else
 dataWindowWarn = sprintf('');
end
if filterWindow > length(data)
 filterWindow = length(data);
 filterWindowWarn = sprintf("filterWindow exceeds length of data and has been
trimmed");
else
```

```
 filterWindowWarn = sprintf('');
end
if iDataEnd > length(data)
 iDataEnd = length(data);
 iDataStart = max(iDataEnd - iDelta, 1);
 dataEndWarn = sprintf("iDataEnd exceeds length of data. Range parameters have been
changed");
else
 dataEndWarn = sprintf('');
end
if ~ismissing([dataWindowWarn, filterWindowWarn, dataEndWarn])
 warning('\n\t%s\n\t%s\n\t%s', dataWindowWarn, filterWindowWarn, dataEndWarn);
end
if nSensors < 9
 %   paddedData = padarray(data', 9 - width(data), nan, 'post')';
 paddedData = NaN(length(data), 9);
 if mod(nSensors, 2)
   fillOrder = [FILL_ORDER(1:nSensors) FILL_ORDER(end)];
 else
   fillOrder = FILL_ORDER;
 end
 for iSensor = 1:nSensors
   paddedData(:, fillOrder(iSensor)) = data(:, iSensor);
 end
 % paddedData(:, floor(linspace(1,9,nSensors))) = data;
 data = fillmissing(paddedData, 'movmean', max(9 - nSensors,2), 2, ...
   EndValues='nearest');
 warning("%d sensors detected. Missing data is being interpolated, and may be
inaccurate.", ...
   nSensors);
 nSensors = 9;
end
% while iDataStart + iDelta < height(dataCol)
% Tsl = Sensor;
% data               = calculate_lux(Tsl, data);
iDataEnd          = iDataStart +  iDelta;
dataSample        = get_sample_range(data, iDataStart, iDataEnd);
dataSample        = dataSample(:, DATA_ORDER);
dataSampleNorm    = get_norm(dataSample);
smoothSample      = smoothdata(dataSample, 'sgolay', filterWindow);
smoothSampleNorm = smoothdata(dataSampleNorm, 'sgolay', filterWindow);
%% Plot Sensor Data
plotSets = {
 data
 dataSample
 smoothSample
 smoothSampleNorm
 };
if CREATE_PLOTS
 figure(FMT.FIG);
 dataPlotFmt.LineWidth = 2;
 for iPlotSet = 1:length(plotSets)
   dataPlot = plot(plotSets{iPlotSet});
   for iSensor = 1:nSensors
```

```
     dataPlotFmt.DisplayName = SENSOR_STRINGS(iSensor, :);
     set(dataPlot(iSensor), dataPlotFmt);
   end % iSensor = 1:nSensors
   legend('show');
   dataAx = gca;
   xlabel('Time Elapsed (milliseconds)');
   ylabel('Irradiance (W/m^2)');
   dataAx.XTickLabel = arrayfun(@(x) sprintf('%d', SCALE * x), dataAx.XTick,...
     'un', 0);
   set(dataAx, FMT.AX);
   saveas(gca, fullfile(OUTPUT_DIR, FIGURE_STRINGS(iPlotSet, :)), 'fig');
   saveas(gca, fullfile(OUTPUT_DIR, FIGURE_STRINGS(iPlotSet, :)), 'png');
   close
 end % iPlotSet = 1:length(plotSets)
end % CREATE_PLOTS
%% Find peaks and dips
t = (iDataStart:iDataEnd); %/ Fs
peakArr = zeros(nSensors, 1);
peakLocArr = zeros(nSensors, 1);
dipArr = zeros(nSensors, 1);
dipLocArr = zeros(nSensors, 1);
% Peak and dip parameters
dipFmt.MinPeakDistance = PEAK_DISTANCE;
dipFmt.MinPeakProminence = PEAK_PROMINENCE;
dipFmt.NPeaks = PEAK_WIDTH;
% Plot local maxima and minima
if CREATE_PLOTS
 sensorPlot = repelem(0, nSensors);
 figure(FMT.FIG);
 hold on
 for iSensor = 1:nSensors
   sensorInv = 1 ./ smoothSampleNorm(:, iSensor);
   [dip, dipLoc] = findpeaks(sensorInv, dipFmt);
   if isempty(dipLoc)
     dipLocArr(iSensor) = 0;
   else
     dipLocArr(iSensor) = dipLoc(1);
   end
   sensorPlot(iSensor) = plot(t, smoothSampleNorm(:, iSensor), ...
     DisplayName='Origin Sensor', LineWidth=2);
   set(sensorPlot(iSensor), dataPlotFmt);
   plot(t(dipLoc), 1 / dip, 'rs', 'MarkerSize', 10);
 end
 hold off
 sensorAx = gca;
 set(sensorAx, FMT.AX);
 xlabel('Time Elapsed (milliseconds)');
 ylabel('Normalized Irradiance');
 sensorAx.XTickLabel = arrayfun(@(x) sprintf('%d', SCALE * x), sensorAx.XTick,
'un', 0);
 saveas(gca, fullfile(OUTPUT_DIR, 'CMV_Sample_Norm'), 'fig');
 saveas(gca, fullfile(OUTPUT_DIR, 'CMV_Sample_Norm'), 'png');
end % CREATE_PLOTS
if strcmp(MATRIX_TYPE, 'normalized')
```

```
 smoothSampleNorm2 = get_norm(smoothSample); % FIXME: Why is the normalization of
smooth sample being defined differently here?
 luxMatrix =  get_matrix(smoothSampleNorm2(:, SENSOR_ORDER), dataWindow);
else
 luxMatrix =  get_matrix(smoothSample(:, SENSOR_ORDER), dataWindow);
end
pages = length(luxMatrix);                              % find maxnumber of frames
imData =  luxMatrix(:, :, 1:pages);                     % set dataset to be analyzed
[imageRow, imageCol, ~] = size(imData);
theta = zeros(imageRow, imageCol, pages);
magnitude = zeros(imageRow, imageCol, pages);
%% Prepare Frames
clear XLim yLim
frames(pages) = struct('cdata',[],'colormap',[]);
figure(FMT.FIG);
% set(gcf, Visible = false);
progressBar = waitbar(0, '1', Name='Populating Frames');
qFigs = nan(1, pages);
for iFrame = 1:(pages)
 waitbar(iFrame/pages, progressBar, sprintf("Frame %4d / %4d\n%3d%% complete",
iFrame, pages, ceil(iFrame/pages * 100)));
 [gx, gy] = imgradientxy( imData(:, :, iFrame), 'sobel'); % Find cmv direction
using Gradient Matrix Method
 [gmag, gdir] = imgradient(gx, gy);
 theta(:, :, iFrame) = gdir;
 magnitude(:, :, iFrame) = gmag;
 if CREATE_VIDEO
   figure(FMT.FIG);
   q = quiver(gx, -gy); %invert to correct visual vector orientation
   xAbsPos = [floor(q.XData + q.UData); ceil(q.XData + q.UData)];
   [xLim(1), xLim(2)] = bounds(xAbsPos, 'all');
   yAbsPos = [floor(q.YData - q.VData); ceil(q.YData - q.VData)];
   [yLim(1), yLim(2)] = bounds(yAbsPos, 'all');
   qFigs(iFrame) = gcf;
 end % if CREATE_VIDEO
end
delete(progressBar);
%% Create Video
if CREATE_VIDEO
 vidDir = 'Gradient Matrix Animations';
 [~, ~] = mkdir([OUTPUT_DIR, DELIMITER, vidDir]);
 videoFmt = 'MPEG-4';
 videoTitle = string([OUTPUT_DIR, DELIMITER, vidDir, DELIMITER, '∇Mat',
char(datetime('now', Format='yy-MM-dd_HH-mm-ss'))]);
 v = VideoWriter(videoTitle, videoFmt);
 v.FrameRate = 30;
 open(v);
 txt = sprintf('dataWindow = %d filterWindow = %d\n', dataWindow, filterWindow);
 progressBar = waitbar(0, '1', Name='Creating Video');
 for iFrame = 1:(pages)
   waitbar(iFrame/pages, progressBar, sprintf("Frame %4d / %4d\n%3d%% complete",
iFrame, pages, ceil(iFrame/pages * 100)));
   ax = gca(qFigs(iFrame));
 %   xlim(ax, [0, xLim(2)]);
```

```
   xlim(ax, [0, 5]);
 %   ylim(ax, [0, yLim(2)]);
   ylim(ax, [0, 5]);
   textWrapper(txt, ax);
   frames(cast(iFrame, 'uint16')) = getframe(qFigs(iFrame));
   writeVideo(v, frames(iFrame));
 end % iFrame = 1:(pages)
 close(v);
 delete(progressBar);
end % if CREATE_VIDEO
%% Create Polar Histograms
mtd1.shadow = struct;
mtd2.shadow = struct;
mtd1.shadow.ang = get_csd(magnitude, theta,
THRESHOLD);                              %correct raw angles
[mtd2.shadow.mag, mtd2.shadow.ang] =  get_resultant_vec(magnitude, theta);
figure(99);
set(gcf, FMT.FIG);
tlo = tiledlayout(TILE.ROWS, TILE.COLS);
title(tlo, 'Shadow Direction Probability');
set(tlo,FMT.TLO);
nexttile(TILE.POS(1), TILE.LARGE_SPAN); % Large Left Tile BEGIN
 mtd1.phistBig = polarhistogram(mtd1.shadow.ang, 10, Normalization="probability");
 hold on;
   mtd2.phistBig = polarhistogram(mtd2.shadow.ang, BIN_EDGES,
Normalization="probability");
   legendLabels(1) = "Method One";
   legendLabels(2) = "Method Two";
   % Plot dotted projection lines
   mtd1.phistBigProj = polarhistogram(mtd1.shadow.ang, 10, ...
     Normalization="count", EdgeColor=FMT.COLORORDER(1, :), ...
     FaceColor='none', LineStyle=':');
   mtd2.phistBigProj = polarhistogram(mtd2.shadow.ang, BIN_EDGES, ...
     Normalization="count", EdgeColor=FMT.COLORORDER(2, :), ...
     FaceColor='none', LineStyle=':');
   % Find bins w/ probability >= 5% and extend to edges
   mtd1.phistBigProj.BinCounts(mtd1.phistBig.Values >= 0.05) = 1;
   mtd2.phistBigProj.BinCounts(mtd2.phistBig.Values >= 0.05) = 1;
   % Set bins w/ problility < 5% to zero
   mtd1.phistBigProj.BinCounts(mtd1.phistBig.Values  < 0.05) = 0;
   mtd2.phistBigProj.BinCounts(mtd2.phistBig.Values  < 0.05) = 0;
   bothMtds.polarAx = gca;
   set(bothMtds.polarAx, FMT.POLAX);
   FMT.RTICKSET();
   the = 0:45:315;
   rho = repmat(gca().RLim, 1, length(the));
   the = repelem(deg2rad(the), 2);
   for iTheta = 1:2:(length(the)-1)
     polarplot(the(iTheta:iTheta+1), rho(iTheta:iTheta+1), ...
       LineWidth=1, LineStyle='-', Color=[0 0 0 0.25]);
   end
 hold off;
 legendLabels(3:length(gca().Children)) = repelem("", length(gca().Children) - 2);
 legend(bothMtds.polarAx, legendLabels, ...
```

```
   Location='northoutside', Orientation='horizontal');
  set(gca, Children=flipud(gca().Children));
% Large Left Tile END
nexttile(TILE.POS(2)); % Upper Right Tile BEGIN
 mtd1.phist = polarhistogram(mtd1.shadow.ang, 10, Normalization="probability");
 mtd1.polarAx = gca;
 mtd1.phist.FaceColor = FMT.COLORORDER(1,:);
 set(mtd1.polarAx, FMT.POLAX);
 FMT.RTICKSET();
% Upper Right Tile END
nexttile(TILE.POS(3)); % Lower Right Tile BEGIN
 mtd2.phist = polarhistogram(mtd2.shadow.ang, BIN_EDGES,
Normalization="probability");
 mtd2.polarAx = gca;
 mtd2.phist.FaceColor = FMT.COLORORDER(2,:);
 set(mtd2.polarAx, FMT.POLAX);
 FMT.RTICKSET();
% Upper Left Tile END
cmvDirection1 = get_cmv_direction(mtd1.shadow.ang, mtd1.phist, 1);
cmvDirection2 = get_cmv_direction(mtd2.shadow.ang, mtd2.phist, 2);
cmvSpeed1     = get_cmv_speed(cmvDirection1, dipLocArr);
cmvSpeed2     = get_cmv_speed(cmvDirection2, dipLocArr);
cmvSpeed1     = fillmissing(cmvSpeed1, "nearest", EndValues='nearest');
cmvSpeed2     = fillmissing(cmvSpeed2, "nearest", EndValues='nearest');
pvSite_dist   = 5; % Distance the sensor cluster is from the PV Site. Units??
pvSite_phi    = 30; %The angle from the site?
TOA           = TimeOfArrival(pvSite_dist,pvSite_phi,cmvSpeed1,cmvDirection1);
cmv           = [TOA cmvDirection1 cmvSpeed1 cmvDirection2 cmvSpeed2];
clk_raw       = clock; %Outputs the [Year Month Day Hour Min Sec]
clk           = fix(clk_raw); %Rounds each entry in clock matrix, only impacts
seconds
% tlo.OuterPosition = tlo.OuterPosition .* [1 1 1 1 + 0.125];
% tlo.InnerPosition = tlo.InnerPosition .* [1 1 1 1 + 0.125];
clk_txt = sprintf('%g/%g/%g  %g:%g:%g', clk);
txt = sprintf('TOA(s)=%6.5g Dir1(θ)=% 6.5g Speed1(m/s)=% 6.5g <> Dir2(θ)=% 6.5g
Speed2(m/2)=% 6.5g', cmv);
textWrapper(txt, gca, [1.13 -0.18]);
textWrapper(clk_txt, gca, [0.90 2.55]); %Displays the time in the top right
figure(gcf);
% saveas(gcf, fullfile(OUTPUT_DIR, 'cmv_histogram'),
'fig');                            %save figure
saveas(gcf, fullfile(OUTPUT_DIR, 'cmv_histogram'),
'png');                            %save image
iLoop = iLoop + 1;
iDataStart = iDataEnd;
% end % while iDataStart + iLoop * iDelta < height(dataCol)
%% Find the optical flow
% OpF =  get_optical_flow(imData);
%  get_vid(OpF, strcat(OUTPUT_DIR, 'OpticalFlow'));                        %save
OpF run as .AVI file
fileID = fopen(strcat(OUTPUT_DIR, 'cmv.txt'), 'w');
fprintf(fileID, '%6s %6s %6s %6s\n', 'CMV_Direction1', 'CMV_Direction2',
'CMV_Speed1', 'CMV_Speed2');
fprintf(fileID, '%0.2f %0.2f %0.2f %0.2f\n', cmv);
```

```
fclose(fileID);
%% Calculate time of arrival
function TOA = TimeOfArrival(d,phi,v,theta)
       TOA = d/(v*cos(deg2rad(phi-theta)));
end
```

Python Scripts for Power Output Prediction:

```
## readSerial-dependent.py
import serial.tools.list_ports
import time


def readCOMport():
    ports = serial.tools.list_ports.comports()
    serialInst = serial.Serial()

    portList = []

    for onePort in ports:
        portList.append(str(onePort))
        print(str(onePort))

    portVar = "COM6"  # CHANGE THIS VALUE TO THE COM PORT LISTED IN THE ARDUINO
EDITOR

    serialInst.baudrate = 9600
    serialInst.port = portVar
    serialInst.open()

    print("---")

    time.sleep(5)
    packet = serialInst.readline()
    sensorReading = packet.decode("utf").rstrip("\n")
    print(sensorReading)

    return sensorReading

output = readCOMport()
```

```
%% readSerial-independent.py
import serial.tools.list_ports
import time

def readCOMport():
    ports = serial.tools.list_ports.comports()
    serialInst = serial.Serial()

    portList = []
```

```
    for onePort in ports:
        portList.append(str(onePort))
        print(str(onePort))

    val = input("Select Port: COM")

    for x in range(0,len(portList)):
        if portList[x].startswith("COM" + str(val)):
            portVar = "COM" + str(val)
            print(portList[x])

    serialInst.baudrate = 9600
    serialInst.port = portVar
    serialInst.open()

    print("---")

    while True:
        time.sleep(1)
        print(serialInst.in_waiting)
        if serialInst.in_waiting:
            packet = serialInst.readline()
            print(packet.decode('utf').rstrip('\n'))

readCOMport()
```

MATLAB Script for Power Output Prediction:

```
%% Clear
clear all
clc
%% Check to see if python environment is detected
% pyenv

%% Run Python script
% Get raw python string
output = pyrunfile("readSerial-dependent.py","output");

% Convert to a regular string
output = string(output);
splitOutput = split(output);

% Pull temperature and irradiance values from output
temperature = splitOutput(3)
irradiance = splitOutput(9)
```

Arduino Script for Power Output Prediction:

```
// Sketch for reading temperature and irradiance values
// Written by Jonathan Ferreira (ECE 2022), with help from Tim Lewis
(ECE 2022)

// REQUIRES the following Arduino libraries:
// - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-
library
// - Adafruit Unified Sensor Lib:
https://github.com/adafruit/Adafruit_Sensor

#include "DHT.h"

#define DHTPIN 2     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();  // Required to use the DHT22 sensor
  int gain = 0.325;
}

void loop() {
  // Wait a few seconds between measurements.
  delay(1000);

  // DHT SENSOR READING
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  Serial.print("Temperature (°C): ");
  Serial.print(t);

  // LICOR SENSOR READING
  int lowValue = analogRead(A0);
  int highValue = analogRead(A4);
```

```
  int irradiance = highValue * gain;

  // Print values
  Serial.print("  Low: ");
  Serial.print(lowValue);
  Serial.print("  High: ");
  Serial.print(highValue);
  Serial.print("  Irradiance: ");
  Serial.println(irradiance);
}
```

Revised ThingSpeak Algorithm:

```
% Enter your MATLAB code below
data1 =
thingSpeakRead(1248525,'Fields',[1,2,3,4,5,6,7,8],'NumPoints',200,'ReadKey','EQ4MUC
OL8YTU4EBS');
data2 =
thingSpeakRead(1307045,'Fields',1,'NumPoints',200,'ReadKey','74F6BCQ36JV3K1EF');
data = [data1,data2];

%while size(data,1)==200
data1 =
thingSpeakRead(1248525,'Fields',[1,2,3,4,5,6,7,8],'NumPoints',200,'ReadKey','EQ4MUC
OL8YTU4EBS');
data2 =
thingSpeakRead(1307045,'Fields',1,'NumPoints',200,'ReadKey','74F6BCQ36JV3K1EF');
data = [data1,data2];
data_sample = data;

%% Set test parameters
matrix_type = 'I_norm';
x_start = 1;
x_end = 200;
window = 50;
filter_window = 51;
threshold = 0.03;
%% Normalize data
data_sample_norm = getNorm(data_sample);
%% Filter and de-noise data
cmv_sample = smoothdata(data_sample,'sgolay',filter_window);
cmv_sample_norm = smoothdata(data_sample_norm,'sgolay',filter_window);
%% Find peaks and dips
t = (x_start:x_end); %/ Fs
peak_arr = zeros(9,1);
plocation_arr = zeros(9,1);
dip_arr = zeros(9,1);
dlocation_arr = zeros(9,1);
% Peak and dip parameters
peak_distance = 50;
```

```
peak_prominence = 0.15;%0.016;
peak_width = 15;
% Plot local maxima and minima
for sensor_idx = 1:9
    sensor_inv = 1./cmv_sample_norm(:,sensor_idx);
    [dip,dlocation] =
findpeaks(sensor_inv,'MinPeakProminence',peak_prominence,'MinPeakDistance',peak_dis
tance,'NPeaks',1);
    if isempty(dlocation)
        dlocation_arr(sensor_idx) = 0;
    else
        dlocation_arr(sensor_idx) = dlocation;
    end
end

%% Get lux matrix
luxMatrix = getMatrix(cmv_sample,window,matrix_type);
% Find CMV direction using Gradient Matrix Method
[~,~,pages] = size(luxMatrix); %find max number of frames
imData = luxMatrix(:,:,1:pages); %set dataset to be analyzed
[image_row, image_col, ~] = size(imData);
theta = zeros(image_row,image_col,pages);
magnitude = zeros(image_row,image_col,pages);
theta2 = zeros(pages,1);
for idx = 1:(pages-1)
    [Gx, Gy] = imgradientxy(imData(:,:,idx),'sobel');
    [Gmag, Gdir] = imgradient(Gx, Gy);
    theta(:,:,idx) = Gdir;
    magnitude(:,:,idx) = Gmag;
end
% Algorithm 2.1
angle_rad = getCSD_v2(magnitude,theta,threshold); %correct raw angles
angle_deg = rad2deg(angle_rad); %convert angles to degrees
% Plot estimated cloud shadow direction
subplot(2,3,3)
hist = polarhistogram(angle_rad,10,'FaceColor',[0 0.4470 0.7410],'FaceAlpha',0.8);
% Algorithm 2.2
[M, Phase_rad] = getResultantVector(magnitude,theta);
Phase_deg = rad2deg(Phase_rad);
% Plot polar histogram
subplot(2,3,6)
histo = polarhistogram(Phase_rad,[0.3926991 1.178097 1.9634954 2.7488936... %set
bin edges
    3.5342917 4.3196899 5.1050881 5.8904862 6.6758844],'FaceColor',[0.8500 0.3250
0.0980],'FaceAlpha',0.8);
% sgtitle("Shadow direction Probablity")

subplot(2,3,[1,2,4,5])
histo = polarhistogram(Phase_rad,[0.3926991 1.178097 1.9634954 2.7488936... %set
bin edges
    3.5342917 4.3196899 5.1050881 5.8904862 6.6758844],'FaceColor',[0.8500 0.3250
0.0980],'FaceAlpha',0.8);
hold on
hist = polarhistogram(angle_rad,10,'FaceColor',[0 0.4470 0.7410],'FaceAlpha',0.8);
```

```
legend('Mehtod Two','Method
One','Location','northoutside','Orientation','horizontal');

%% Get CMV final direction and speed
CMV_Direction1 = getCMV_Direction_v2(angle_rad,hist,1);
CMV_Direction2 = getCMV_Direction_v2(Phase_rad,histo,2);
CMV_Speed1 = getCMV_Speed(CMV_Direction1,dlocation_arr);
CMV_Speed2 = getCMV_Speed(CMV_Direction2,dlocation_arr);
CMV = [CMV_Direction1 CMV_Direction2 CMV_Speed1 CMV_Speed2];
PVSite_distance=5;
PVSite_phi=30;
TOA = TimeOfArrival(PVSite_distance,PVSite_phi,CMV_Speed1,CMV_Direction1)

ResultString1={"Dir(1): "+ (CMV_Direction1) + " Speed(1): "+ (CMV_Speed1) + "< >
Dir(2): "+ (CMV_Direction2) + " Speed(2): "+ (CMV_Speed2)};
annotation('textbox', [0.25 0.02  0.5
0.05],'String',ResultString1,'FitBoxToText','on');

ResultString1={"TOA: "+ (TOA) };
annotation('textbox', [0.08 0.02  0.5
0.05],'String',ResultString1,'FitBoxToText','on');

%end

%% Functions

function data_sample_norm = getNorm(data_sample)
%% This function normalizes data with respect to each column
[row,col] = size(data_sample);
data_norm = zeros(row,col);
for i = 1:col
    if max(abs(data_sample(:,i))) ~= 0
        data_norm(:,i) = data_sample(:,i)/max(abs(data_sample(:,i)));
    end
end
data_sample_norm = data_norm;
end

function outputArray = getMatrix(cmv_sample,window,matrix_type)
%% This function maps lux data into a selected matrix type
[data_length,~] = size(cmv_sample);
cmv_sample_norm = getNorm(cmv_sample);
pages = data_length - window + 1

% Get raw pixels
I_raw = zeros(3,3,pages);
for j = 1:data_length
    I_temp = [cmv_sample(j,5) cmv_sample(j,4) cmv_sample(j,6);
        cmv_sample(j,8) cmv_sample(j,7) cmv_sample(j,9);
        cmv_sample(j,2) cmv_sample(j,1) cmv_sample(j,3)];

    I_raw(:,:,j) = I_temp;
end
```

```
% Get pixels
switch matrix_type
    case 'I'
        I = zeros(3,3,pages);
        for j = 1:pages
            for k = 1:window
                I_temp = [cmv_sample(j-1+k,5) cmv_sample(j-1+k,4) cmv_sample(j-
1+k,6);
                    cmv_sample(j-1+k,8) cmv_sample(j-1+k,7) cmv_sample(j-1+k,9);
                    cmv_sample(j-1+k,2) cmv_sample(j-1+k,1) cmv_sample(j-1+k,3)];

                I(:,:,j) = I(:,:,j) + I_temp;
            end
            I(:,:,j) = I(:,:,j)/window;
        end
        outputArray = I;

    case 'I_norm'
        % Get normalized pixels
        I_norm = zeros(3,3,pages);
        for j = 1:pages
            for k = 1:window
                I_temp = [cmv_sample_norm(j-1+k,5) cmv_sample_norm(j-1+k,4)
cmv_sample_norm(j-1+k,6);
                    cmv_sample_norm(j-1+k,8) cmv_sample_norm(j-1+k,7)
cmv_sample_norm(j-1+k,9);
                    cmv_sample_norm(j-1+k,2) cmv_sample_norm(j-1+k,1)
cmv_sample_norm(j-1+k,3)];

                I_norm(:,:,j) = I_norm(:,:,j) + I_temp;
            end
            I_norm(:,:,j) = I_norm(:,:,j)/window;
        end
        outputArray = I_norm;

    case 'I_norm_v2'
        I_norm_v2 = zeros(3,3,pages);
        for j = 1:pages
            for k = 1:window
                I_temp = [cmv_sample_norm(j-1+k,7) cmv_sample_norm(j-1+k,8)
cmv_sample_norm(j-1+k,9);
                    cmv_sample_norm(j-1+k,4) cmv_sample_norm(j-1+k,5)
cmv_sample_norm(j-1+k,6);
                    cmv_sample_norm(j-1+k,1) cmv_sample_norm(j-1+k,2)
cmv_sample_norm(j-1+k,3)];

                I_norm_v2(:,:,j) = I_norm_v2(:,:,j) + I_temp;
            end
            I_norm_v2(:,:,j) = I_norm_v2(:,:,j)/window;
        end
        outputArray = I_norm_v2;

    case 'I_ave_norm'
        I_ave_norm = zeros(3,3,pages);
```

```
        idx = 1;
        for j = 1:data_length
            I_temp = [cmv_sample_norm(j,5) cmv_sample_norm(j,4)
cmv_sample_norm(j,6);
                cmv_sample_norm(j,8) cmv_sample_norm(j,7) cmv_sample_norm(j,9);
                cmv_sample_norm(j,2) cmv_sample_norm(j,1) cmv_sample_norm(j,3)];

            I_ave_norm(:,:,idx) = I_ave_norm(:,:,idx) + I_temp;
            if mod(j,window) == 0
                I_ave_norm(:,:,idx) = I_ave_norm(:,:,idx)/window;
                idx = idx + 1;
            end
        end
        outputArray = I_ave_norm;

    otherwise
        fprintf('ERROR')
end

end

function outputArray = getCSD_v2(magnitude,theta,threshold)
%% This function gets the raw magnitude and theta converts to a corrected array
[~,~,pages] = size(magnitude);

%% Find average angles
angle_array = zeros(pages,1);   %initialize list of angles
for idx = 1:pages
    theta_avg = 0; %initialize variable
    cnt = 0;
    for i = 1:3
        for j = 1:3
            if magnitude(i,j,idx) > threshold
                theta_avg = theta_avg + deg2rad(theta(i,j,idx)); %get a running
tally of angles
                cnt = cnt + 1;
            end
        end
    end

    theta_avg = theta_avg/cnt;                                  %get average angle
    angle_array(idx,1) = theta_avg;
end

%% Find the first non-NaN element's sign
test_array = angle_array;
cnt = 0;
for idx = 1:numel(test_array)
    if isnan(test_array(idx)) ~= 1
        cnt = cnt + 1;
        angle_array(cnt,1) = test_array(idx);
    end
end
```

```
%Find starting point
start = 1;
while start < numel(test_array) && isnan(test_array(start)) ~= 0
    start = start + 1;
end

%% Check the quadrants of the first 1/4 of the elements
angle_label = getQUADRANT(test_array);

check_array = cell(numel(angle_label,1));
check_array(1,1) = angle_label(start);

[max_row,~] = size(angle_array);
max_check = start + ceil(max_row/8);

cnt = 1;
for idx = start:max_check
    if isequal(angle_label(idx),check_array(cnt)) == 0 %check if qudrant is not the
same
        cnt = cnt + 1;  %increment
        check_array(cnt,1) = angle_label(idx);  %save new quadrant label to another
cell
    end
end

%% Correct angles (in radians) opposite that of reference quadrants
for idx = 1:numel(test_array)
    notequal = 0;

    for idx2 = 1:numel(check_array)
        if isequal(angle_label{idx},check_array{idx2}) == 1
            notequal = notequal + 1;
        end
    end

    if notequal == 0
        test_array(idx) = test_array(idx) + pi;
        angle_label{idx} = getQUADRANT(test_array(idx));
    end
end

%% Return output
outputArray = test_array;
end

function QUADRANT = getQUADRANT(theta)
%% This function receives an input angle in radiance and finds the quadrant it
belongs to
QUADRANT = cell(numel(theta),1);

for idx = 1:numel(theta)
    x_val = cos(theta(idx));
    y_val = sin(theta(idx));
```

```
    if y_val > 0 && x_val > 0
        QUADRANT{idx,1} = 'Q1';
    elseif y_val > 0 && x_val < 0
        QUADRANT{idx,1} = 'Q2';
    elseif y_val < 0 && x_val < 0
        QUADRANT{idx,1} = 'Q3';
    elseif y_val < 0 && x_val > 0
        QUADRANT{idx,1} = 'Q4';
    end
end
end

function [M, Phase] = getResultantVector(magnitude,theta)
% This function converts the magnitudes and angles into a phasor. The
% resultant vector's is then decomposed as magnitude, M, and angle, Phase.

z_total = 0;
threshold = 0.02;
[~,~,pages] = size(magnitude);
M = zeros(pages,1);
Phase = zeros(pages,1);

for idx = 1:(pages-1)
    for i = 1:3
        for j = 1:3
            R = magnitude(i,j,idx);
            rtheta = deg2rad(theta(i,j,idx));

            if R > threshold
                z = R*(cos(rtheta)+1i*sin(rtheta));    %convert into complex form
                z_total = z_total + z;                 %add complex numbers
            end
        end
    end

    M(idx) = abs(z_total);
    Phase(idx) = angle(z_total);
end
end

function CMV_Direction = getCMV_Direction_v2(angle_rad,hist_plot,algorithm)
%% This function gets the CMV direction using algorithm (1) without 2*pi wraparound
or (2) with 2*pi wraparound
[max_row,~] = size(angle_rad);
[~,edge_idx] = max(hist_plot.Values);
edgeValues = hist_plot.BinEdges;
% [~,edge_idx] = max(histo.Values);
% edgeValues = histo.BinEdges;
lower_bound = edgeValues(edge_idx);
upper_bound = edgeValues(edge_idx+1);
direction_temp = zeros(1,1);
cnt = 1;

switch algorithm
```

```
    case 1
        % Get final CMV direction if Algorithm 2.1
        for idx = 1:max_row
            if angle_rad(idx) > lower_bound && angle_rad(idx) < upper_bound
                direction_temp(cnt) = angle_rad(idx);
                cnt = cnt + 1;
            end
        end

    case 2
        % Get final CMV direction if Algorithm 2.2
        for idx = 1:max_row
            if angle_rad(idx) < 0.3926991
                angle_rad(idx) = angle_rad(idx) + 2 * pi;
            end

            if angle_rad(idx) > lower_bound && angle_rad(idx) < upper_bound
                direction_temp(cnt) = angle_rad(idx);
                cnt = cnt + 1;
            end
        end
end

CMV_Direction = rad2deg(mean(direction_temp));
if CMV_Direction < 0
    CMV_Direction = CMV_Direction + 360;
elseif CMV_Direction > 360
    CMV_Direction = CMV_Direction - 360;
end

end

function CMV_Speed = getCMV_Speed(CMV_Direction,dlocation_arr)
% This function receives the CMV direction and calculates the cloud shadow
% speed from the local minima locations
theta = deg2rad(CMV_Direction);
dlocation_arr(dlocation_arr==0) = NaN;
v = zeros(3,1);

% Initialize variables
delta_t1 = 0;
delta_t2 = 0;
delta_t3 = 0;
delta_t4 = 0;
delta_t5 = 0;

%CMV_direction = ~45 degrees
if theta >= 0.3926991 && theta < 1.178097
    delta_t1 = dlocation_arr(4) - dlocation_arr(8); %sqrt(2) m
    delta_t2 = dlocation_arr(6) - dlocation_arr(2); %2m
    delta_t3 = dlocation_arr(9) - dlocation_arr(1); %sqrt(2) m
    delta_t4 = dlocation_arr(6) - dlocation_arr(7); %1m
    delta_t5 = dlocation_arr(7) - dlocation_arr(2); %1m
    choice = 45
```

```
    %CMV_direction = ~90 degrees
elseif theta >= 1.178097 && theta < 1.9634954
    delta_t1 = dlocation_arr(5) - dlocation_arr(2); %sqrt(2) m
    delta_t2 = dlocation_arr(4) - dlocation_arr(1); %2m
    delta_t3 = dlocation_arr(6) - dlocation_arr(3); %sqrt(2) m
    choice = 90
    %CMV_direction = ~135 degrees
elseif theta >= 1.9634954 && theta < 2.7488936
    delta_t1 = dlocation_arr(4) - dlocation_arr(9); %sqrt(2) m
    delta_t2 = dlocation_arr(5) - dlocation_arr(3); %2m
    delta_t3 = dlocation_arr(8) - dlocation_arr(1); %sqrt(2) m
    delta_t4 = dlocation_arr(5) - dlocation_arr(7); %1m
    delta_t5 = dlocation_arr(7) - dlocation_arr(3); %1m
    choice = 135
    %CMV_direction = ~180 degrees
elseif theta >= 2.7488936 && theta < 3.5342917
    delta_t1 = dlocation_arr(5) - dlocation_arr(6); %sqrt(2) m
    delta_t2 = dlocation_arr(8) - dlocation_arr(9); %2m
    delta_t3 = dlocation_arr(2) - dlocation_arr(3); %sqrt(2) m
    choice = 180
    %CMV_direction = ~225 degrees
elseif theta >= 3.5342917 && theta < 4.3196899
    delta_t1 = dlocation_arr(8) - dlocation_arr(4); %sqrt(2) m
    delta_t2 = dlocation_arr(2) - dlocation_arr(6); %2m
    delta_t3 = dlocation_arr(1) - dlocation_arr(9); %sqrt(2) m
    delta_t4 = dlocation_arr(2) - dlocation_arr(7); %1m
    delta_t5 = dlocation_arr(7) - dlocation_arr(6); %1m
    choice = 225
    %CMV_direction = ~270 degrees
elseif theta >= 4.3196899 && theta < 5.1050881
    delta_t1 = dlocation_arr(2) - dlocation_arr(5); %sqrt(2) m
    delta_t2 = dlocation_arr(1) - dlocation_arr(4); %2m
    delta_t3 = dlocation_arr(3) - dlocation_arr(6); %sqrt(2) m
    choice = 270
    %CMV_direction = ~315 degrees
elseif theta >= 5.1050881 && theta < 5.8904862
    delta_t1 = dlocation_arr(9) - dlocation_arr(4); %sqrt(2) m
    delta_t2 = dlocation_arr(3) - dlocation_arr(5); %2m
    delta_t3 = dlocation_arr(1) - dlocation_arr(8); %sqrt(2) m
    delta_t4 = dlocation_arr(3) - dlocation_arr(7); %1m
    delta_t5 = dlocation_arr(7) - dlocation_arr(5); %1m
    choice = 315
    %CMV_direction = ~360 degrees
elseif theta >= 5.8904862 && theta < 6.6758844 || theta < 0.3926991
    delta_t1 = dlocation_arr(6) - dlocation_arr(5); %sqrt(2) m
    delta_t2 = dlocation_arr(9) - dlocation_arr(8); %2m
    delta_t3 = dlocation_arr(3) - dlocation_arr(2); %sqrt(2) m
    choice = 360
end

v(1) = abs(sqrt(2)/delta_t1);
v(2) = abs(2/delta_t2);
v(3) = abs(sqrt(2)/delta_t3);
v(4) = abs(1/delta_t4);
```

```
v(5) = abs(1/delta_t5);

for idx = 1:5
    if isinf(v(idx)) == 1
        v(idx) = nan;
    end
end

CMV_Speed = nanmean(v)/150*1000; %meters per second
end

%% Calculate time of arrival
function TOA = TimeOfArrival(d,phi,v,theta)
        TOA = d/(v*cos(deg2rad(phi-theta)));
end
```

Recommended PCB Schematic:

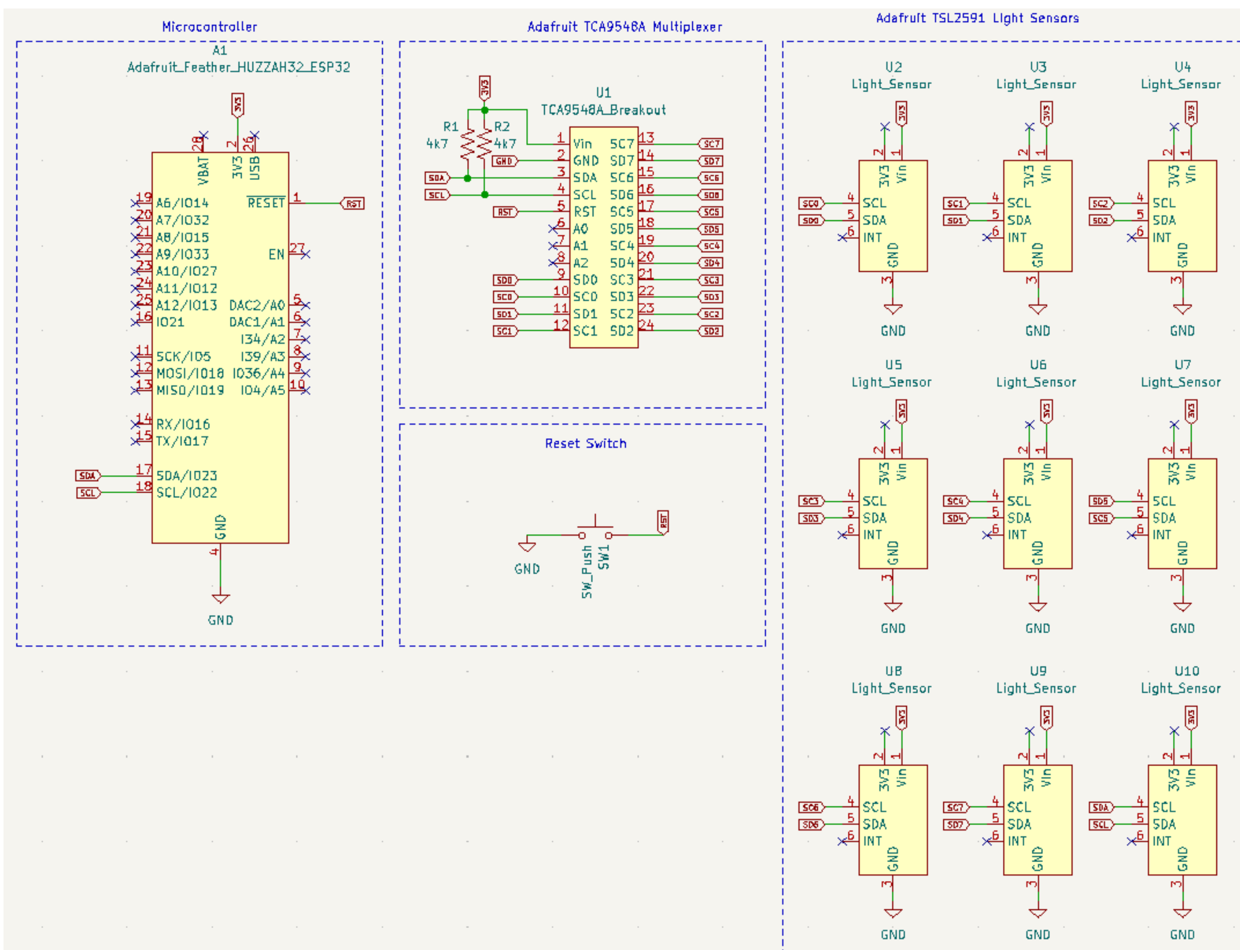


Figure 17. Schematics for major electrical components used in recommended PCB design.

Additional Figures:

Figure 18. Displaying the installation of the PV array at the site on the top of the East Hall Parking garage. This is a close up of the weather proofing box containing the battery and the electronics.

Figure 19. Displaying the installation of the PV array at the site on the top of the East Hall Parking garage. The installation expert, Tim Lewis, and Habebullah Adua are shown from left to right.

Figure 20. The PV array after the successful installation at the site.

Figure 21. Image showing full sensor array setup with PV system