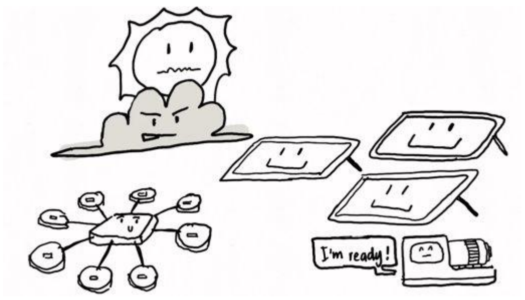
Cloud Motion Vector System to Monitor and Predict Output Power of a Photovoltaic System in Real Time



A Major Qualifying Project

Submitted

by

Michael Carpinello,

Jacob McManus,

Matthew Moreira, and

Kyle Pacheco

Advised by:

Dr. Maqsood Ali Mughal

A Major Qualifying Project Report Submitted to the faculty of the Electrical and Computer Engineering Department at Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in cooperation with the Eversource Energy.

Date: March 18th, 2021

Abstract

When a PV system is covered by a cloud shadow, the energy supply becomes less than the demand, and therefore, the backup generator needs to turn on to bring the supply back up. To ensure that the demand is always met, it is important for utility companies to provide alternate backup energy sources, such as backup generators, to supplement the remaining demand. Since the switch from solar to backup power is not instantaneous, it is important to predict the PV system output power. We propose a cloud motion vector system (CMVS) model that will operate in real-time at a frequency of 10 kHz and utilizes a cluster of ambient light sensors to detect a cloud rolling over a PV system and to determine size, speed, and direction of the moving cloud. This data is fed into a PV system model to compute power reduction due to change in irradiance from the cloud cover and turn on the backup power supply. In turn, this endeavor aims to create a sustainable electrical grid network with the majority of power coming from renewable energy resources. Furthermore, this model will prevent frequent switching of voltage-controlled devices, which may otherwise shorten their life expectancy and ultimately increase operating costs.

Acknowledgements

This project would not have been possible without the aid and guidance of several individuals and organizations; hence, we wish to express our heartfelt gratitude to all of those who helped us realize this MQP.

First, we would like to thank Worcester Polytechnic Institute for their continued support during a global pandemic as well as providing us with lab space and an on-campus site to implement our design and perform experiments. We would also like to thank graduate students Hongjie Zhang and Song Cui for working alongside us, sharing their knowledge, and guiding us through difficult times. Most of all, we would like to thank Dr. Maqsood Ali Mughal for advising our project, facilitating different components of the project, and providing us with resources to complete the project. His proper guidance and encouragement each week was beneficial.

In addition, we would like to thank Eversource Energy for sharing key information, providing feedback on our design, and offering on-site access and resources to implement and test the design of our project. We would also like to thank HyperChiicken for sharing key MATLAB algorithms, Bash scripts, and Python scripts with us. Finally, we would like to thank William Appleyard, the Electronics Technician in the Electrical and Computer Engineering department at Worcester Polytechnic Institute for the speedy acquisition of materials that were required to complete the hardware part of the design.

Table of Contents

List of Figures

1. Introduction

Over the last decade or so, the shift towards green energy has seen technologies like photovoltaics become a substantial component in our nation's power grid. However, due to the intermittent nature of PV technology, it inherently produces power only during daylight hours. In addition, clouds rolling over a PV system could potentially cause significant decrease in solar irradiance, hence, reduction in power production from the PV system. This instantaneous drop in power disrupts the conventional methods for planning the daily operation of the electrical grid network. Power fluctuating over multiple time horizons, ultimately forces the grid operator to adjust their real-time operating procedures. As a result, utility companies struggle to find a balance between electricity supply and demand that needs to be maintained at all times to avoid a blackout or other equipment-related issues. It has become apparent that PV power is not without its own challenges.

Cloud coverage can significantly reduce the power output of PV arrays, and this sudden loss of power can cause flicker. When PV panel output drops, either due to weather or the time of day, this loss of output is compensated for by on-demand generators. This solution is sufficient for regular events such as the setting of the sun; however, weather has proved to be more difficult to predict. Therefore, compensating for a loss of power output with generators becomes a reactive measure to reduce or eliminate flicker, rather than a preventative one. The delay between cloud coverage and generator compensation, and the power instability that it causes, is unacceptable for the power grid of our modern era.

Eversource - a publicly traded energy delivery company providing electrical, natural gas, and water services to customers across New England - reached out to WPI for help in resolving this flickering issue. Overall, there are certain standards that energy delivery companies must adhere to when providing their services. For their electrical energy delivery, Eversource aims to keep the voltage levels within +5% / -5% of the nominal grid voltages. [5] When working with combustion these boundaries are normally feasible, but with many forms of renewable energy it can be difficult to maintain such a consistent output. When it comes to solar energy in particular, the voltage levels can fluctuate due to changes in solar irradiance at the photovoltaic site. With many clouds in the sky, these fluctuations are more drastic, and as photovoltaic penetration continues to increase, staying within these regulations becomes increasingly more difficult.

We propose a Cloud Motion Vector System (CMVS) that will improve the stability of photovoltaic power and make it a more viable option specifically when cloud coverage becomes a concern. In particular, the system will reduce or eliminate variations in power output from cloud coverage, by predicting the decrease in irradiance and proactively compensating for it with an alternative power source. We developed a light sensor based CMVS model to detect a cloud rolling over a PV system. Our system measures cloud parameters including size, speed, and direction, all with high accuracy and a resolution of 100 ms. A CMVS system spread throughout a large area (such as a city) has potential to bring us closer to more reliable PV power, a step closer to smart cities, and to incorporating more efficient microgrid systems into our society.

2. Background

As stated earlier, the purpose of this project is to develop a real-time CVMS that collects cloud motion simulator parameters, predicts output power, calculates flicker severity index, and provides real-time energy back-up. In this section, we will study previous research, flicker severity index, and Eversource Energy's mission and purpose.

2.1 Previous Research

Before the development of a real-time cloud motion vector system, it is important to first examine previously completed research. In the academic years of 2019 and 2020, a group of four students at Worcester Polytechnic Institute (WPI) - sponsored by Eversource Energy - developed a model to study the impacts of cloud coverage upon PV systems. This project is a continuation of that work. The previous MQP group calculated six Cloud Motion Simulator Parameters: The number of clouds, the cloud speed, the width of clouds, the number of clouds, the time interval between two successive clouds, and the direction of cloud motion using data from a single pyranometer. However, irradiance (IR) data was only available at a fifteen-minute interval, from an external pyranometer. Our model, on the other hand, works in real time with its own array of sensors, immensely improving the granularity of IR data. In addition, our system uses multiple relatively cost-efficient light sensors, as opposed to a pyranometer, and achieves real-time results with a similar level of precision.

When clouds pass over a PV array, they block sunlight from being absorbed by the PV system, weakening the electric field. As a result of this, the electrical output "varies in reference to the initial output". From here, there is a ripple effect that is then felt across the entire grid, ultimately, increasing operating costs while causing "electrical instability and asset degradation". To determine a method as to how to predict and better manage the flicker's effects on the electrical grid, the team of students at WPI used irradiance data provided by Eversource Energy to develop a model interfaced with Synergi - a simulation software- that provided a user with a predicted timeframe as to when the flicker moment occurred; therefore, allowing the user to "compensate accordingly". [1]

While their research and results helped to establish a model that can be used by Eversource Energy in the future, our model differs in that we will be providing them with a system that collects and analyzes data in real-time. By having the ability to adequately predict cloud motion in real-time, Eversource Energy can provide a real-time energy backup system to its customers in the event of impending cloud coverage.

2.2 Eversource Energy

As the world turns to renewable energy, so does Eversource. They are committed to switching to renewable energy and becoming a carbon neutral company by the year 2030. [7] This means that their customers will be encouraged to make the switch as well, resulting in more solar energy on the grid. As of February 5, 2020, Eversource can power more than 11,000 homes with their solar portfolio. [8]

When calculating voltage flicker, Eversource engineers run simulations that assume that the rate of change of power that occurs is instant and goes from 100% to 5% and then back to 100%. [5] This is a simulation and only tests the extremes of voltage flicker. It does not account for real cloud motion and all of the different possibilities of how voltage flicker could fluctuate.

2.3 Flicker Severity Index

Flicker, voltage flicker, or lamp flicker is defined as noticeable illumination changes from lighting equipment caused by voltage fluctuations on electric power systems. [4] These changes occur below the 50 to 60 Hz frequency of the supply, so they are not corrected by the systems; however, they occur frequently enough to an extent that causes irritation to those observing even if it’s not easily visible.

Due to the irritation caused by this flicker, companies began adopting a standard to maintain their flicker levels. Eversource’s standard is to keep their levels between +2% / -2%. This standard came about when General Electric published the “GE flicker curve” in 1925, which was created from a collection of previous flicker studies. The GE flicker curve utilized two different curves. One curve would show the borderline of light visibility and the other would show the borderline of irritation. This set the stage for flicker standards and although this method has drawbacks for more complex voltage modulation, it is still used today and there are also more accurate methods for calculating voltage flicker. [4]

There exist various methods for determining a flicker severity index. First, using statistical methods, a cumulative probability can be created to estimate the chance that the signal level falls out of acceptable bounds during a given time period. [6]

$$p(l) = \frac{\textit{total time where signal level} \geq l}{T} \tag{1}$$

Where p(l) is the cumulative probability, l is the acceptable signal level, and T is the period. To derive the short-term flicker severity index, five-gauge points are taken over a ten minute period. These five-gauge points are $P_{0.1}$, P_{1s}, P_{3s}, P_{10s}, and P_{50s}, representing the probabilities that flicker exceeds acceptable bounds 0.1% of the time, 1% of the time, 3% of the time, et cetera. The equation to calculate short-term flicker severity index, henceforth known as P_{st}, is as follows:

$$P_{st} = 0.1\sqrt{3.14P_{0.1} + 5.25P_{1s} + 6.57P_{3s} + 28P_{10s} + 8P_{50s}} \tag{2}$$

In and of themselves, the above gauge points are calculated as follows:

$$P_{1s} = \frac{P_{0.7}+P_{1}+P_{1.5}}{3} \tag{3}$$

$$P_{3s} = \frac{P_{2.2}+P_{3}+P_{4}}{3} \tag{4}$$

$$P_{10s} = \frac{P_{6}+P_{8}+P_{10}P_{13}+P_{17}}{5} \tag{5}$$

$$P_{50s} = \frac{P_{30}+P_{50}+P_{80}}{3} \tag{6}$$

With these calculations, the short-term flicker severity index over a ten minute period can be estimated; hence, providing us with a metric by which to empirically measure flicker.

3. Methodology

From the start, there were several major factors that we considered, including: the scope of the project, the general system overview and timeline of events, the collection of equipment, and the setup, design, and implementation of hardware and software. This chapter outlines the approaches that we have taken to achieve each of these objectives in order to further our development of a real-time cloud motion vector system that collects cloud motion simulator parameters, predicts output power, calculates flicker severity index, and provides real-time energy back-up.

3.1 System Diagram and Timelines

At the beginning of each term we created timelines, shown in Figures 2-4 below, to guide us throughout the term. These helped us stay on track and complete each task in an organized manner that allowed for the final product to be completed by the end of the final term.

A system diagram, shown in Figure 1 below, was also created to show an outline of how the CMVS system was structured. In order for nine TSL2591 light sensors to be used, eight of them were connected to one TCA9548A multiplexer with the ninth one being connected to another TCA9548A multiplexer. Both of these multiplexers were connected to the Arduino Uno using one i2c address. The Arduino Uno was then connected to the BeagleBone Black via USB cable. This BeagleBone was then connected to an ethernet port and to the PC via USB cable. This allowed the data to be stored in a .csv file on the computer and sent to the Thingspeak server.

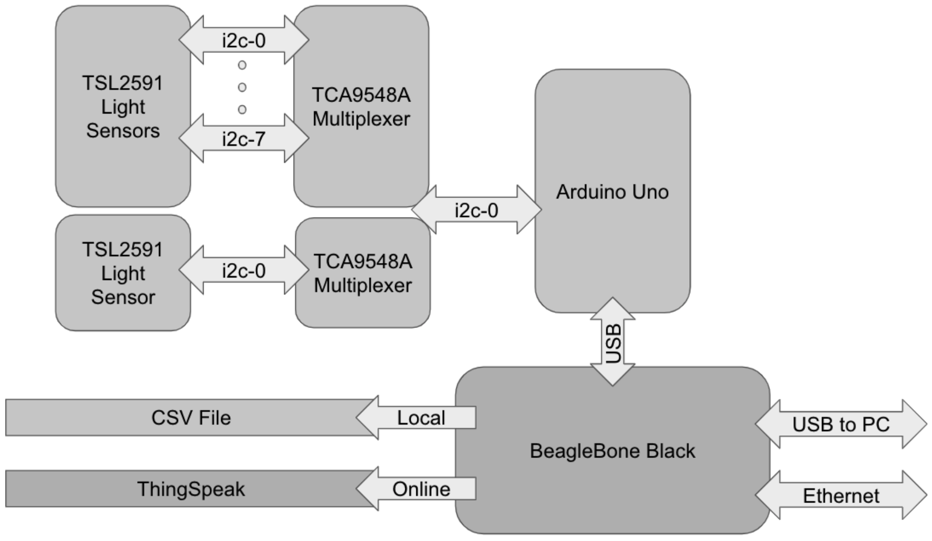
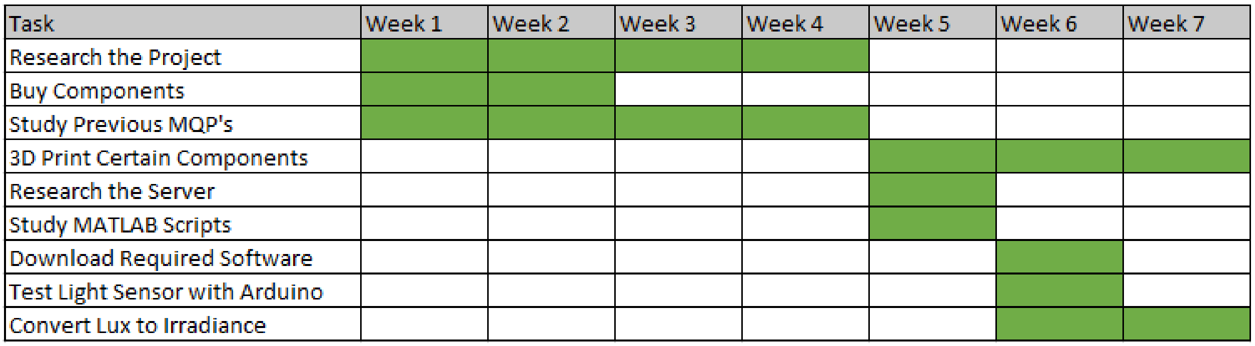
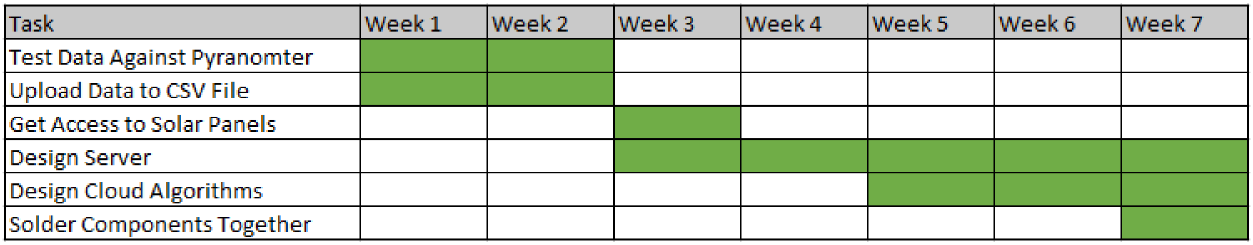


Figure 1: CMVS System Diagram.



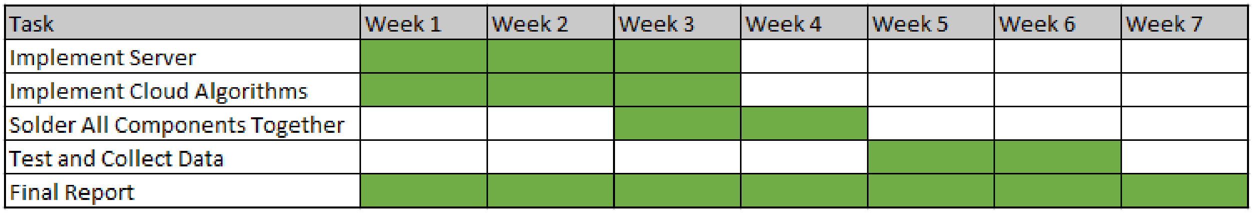
Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Research the Project	🟩	🟩	🟩	🟩			
Buy Components	🟩	🟩					
Study Previous MQP's	🟩	🟩	🟩	🟩			
3D Print Certain Components					🟩	🟩	🟩
Research the Server					🟩		
Study MATLAB Scripts					🟩		
Download Required Software						🟩	
Test Light Sensor with Arduino						🟩	
Convert Lux to Irradiance						🟩	🟩

Figure 2: A-Term Timeline.



Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Test Data Against Pyranomter	🟩	🟩					
Upload Data to CSV File	🟩	🟩					
Get Access to Solar Panels			🟩				
Design Server			🟩	🟩	🟩	🟩	🟩
Design Cloud Algorithms					🟩	🟩	🟩
Solder Components Together							🟩

Figure 3: B-Term Timeline.



Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Implement Server	🟩	🟩	🟩				
Implement Cloud Algorithms	🟩	🟩	🟩				
Solder All Components Together			🟩	🟩			
Test and Collect Data					🟩	🟩	
Final Report	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Figure 4: C-Term Timeline.

3.2 Collecting Irradiance Data

To collect irradiance data, we needed to first obtain an Arduino Uno, nine TSL2591 Light Sensors, and two TCA9548A Multiplexers. Since the Arduino only had two i2C channels and the multiplexers only had eight channels, we had to use two multiplexers to take in and forward the nine inputs out through a single channel. To begin, we wired the light sensors and multiplexers to the arduino as such: Vin to the power supply, GND to the common power/data ground, SCL pin of the multiplexers to the I2C clock SCL pin on the Arduino ("A5"), and the SDA pin of the multiplexers to the I2C data SDA pin on the Arduino ("A4"). In addition to this, we had to connect each individual sensor's SCL and SDA pins to a corresponding pin on the multiplexers. For instance, the first sensor's SCL pin was connected to the SC0 pin on one of the multiplexers and it's SDA pin was connected to the SD0 pin on the same multiplexer. The full wiring schematic is illustrated in Figure 5 below.

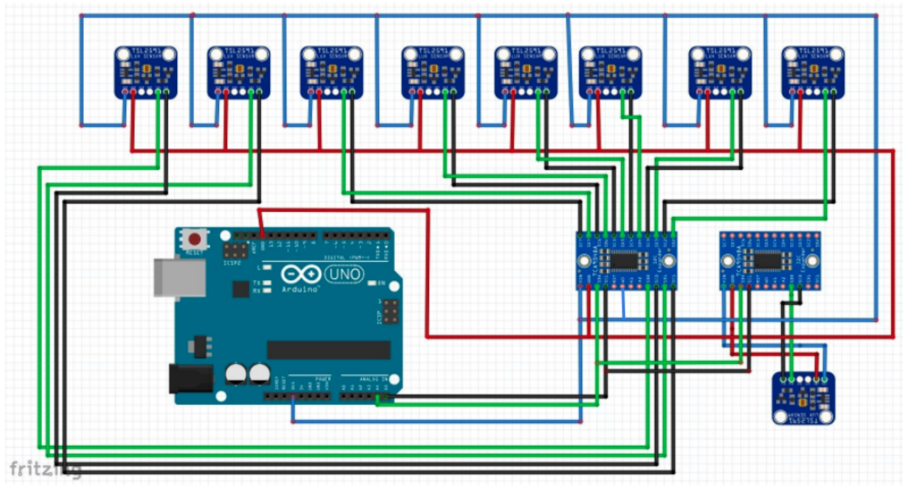
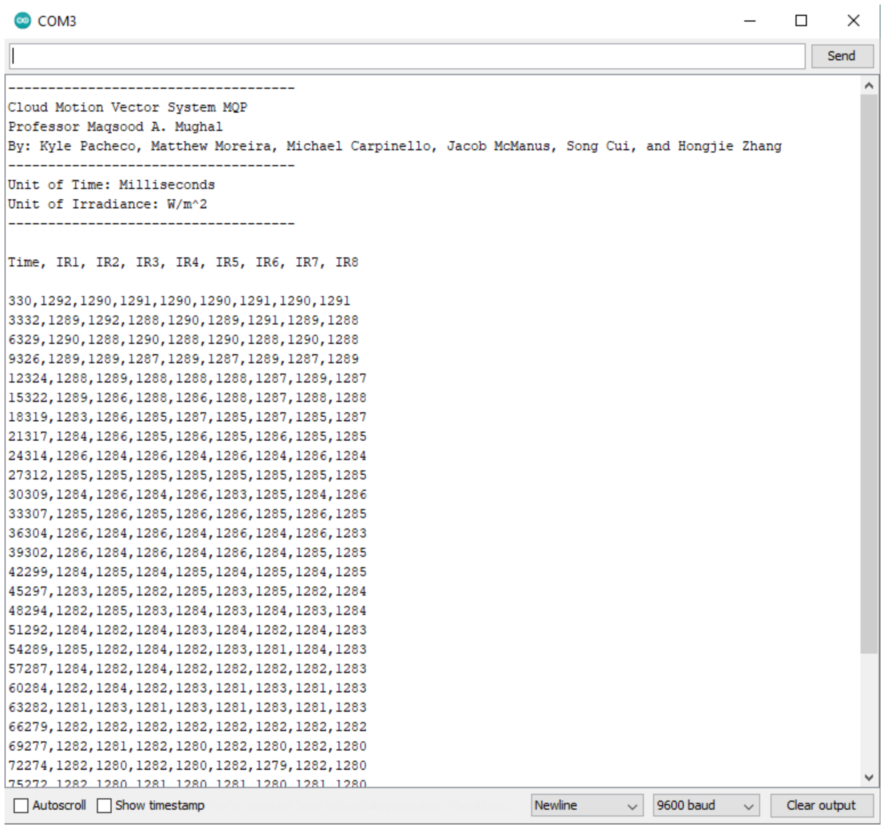


Figure 5: Arduino Uno Wiring Schematic.

After connecting the sensor components, we then downloaded an Arduino IDE in order to program the microcontroller. Following the installation, we opened up the program and installed the Adafruit_TSL2591 library in order to begin reading sensor data. Similarly, we then had to install the sensor itself, named "Adafruit Unified Sensor", in the library manager. After installing the most up to date libraries and components, we then had to modify the "tsl2591" demo file so that it could read the irradiance values for all nine sensors that we were using. To do so, we simply introduced a for loop in our main function that iterated through nine values before resetting itself. By doing this, the program would iterate through the nine sensors connected to the multiplexer and record the individual readings. In addition to this change, we also updated some of the "printf" functions to personalize what our output would look like. After successfully verifying and uploading the code to the Arduino, we opened the serial terminal window in the Arduino IDE and obtained the following values illustrated in Figure 6 below that updated every 300 ms in real time.



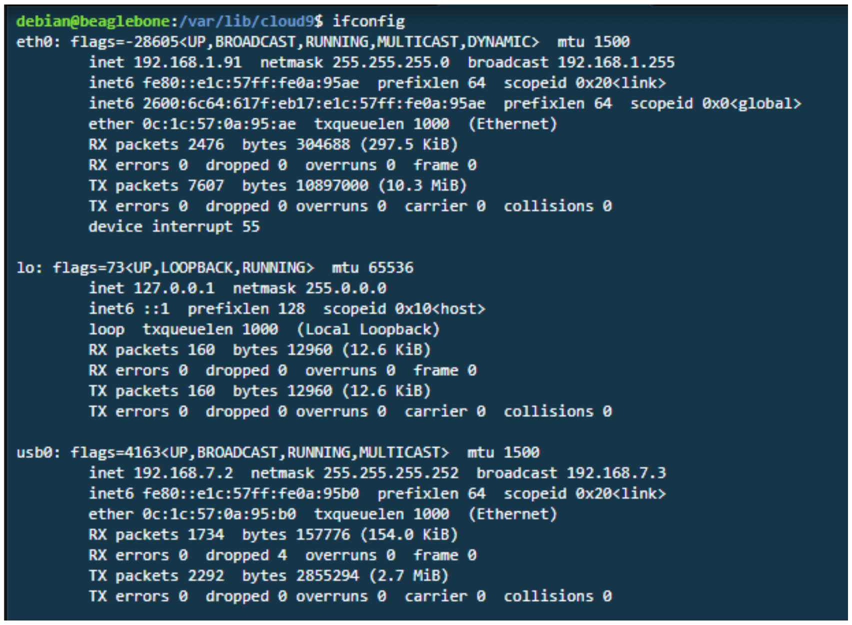
```
------------------------------------
Cloud Motion Vector System MQP
Professor Maqsood A. Mughal
By: Kyle Pacheco, Matthew Moreira, Michael Carpinello, Jacob McManus, Song Cui, and Hongjie Zhang
------------------------------------
Unit of Time: Milliseconds
Unit of Irradiance: W/m^2
------------------------------------

Time, IR1, IR2, IR3, IR4, IR5, IR6, IR7, IR8

330,1292,1290,1291,1290,1290,1291,1290,1291
3332,1289,1292,1288,1290,1289,1291,1289,1288
6329,1290,1288,1290,1288,1290,1288,1290,1288
9326,1289,1289,1287,1289,1287,1289,1287,1289
12324,1288,1289,1288,1288,1288,1287,1289,1287
15322,1289,1286,1288,1286,1288,1287,1288,1288
18319,1283,1286,1285,1287,1285,1287,1285,1287
21317,1284,1286,1285,1286,1285,1286,1285,1285
24314,1286,1284,1286,1284,1286,1284,1286,1284
27312,1285,1285,1285,1285,1285,1285,1285,1285
30309,1284,1286,1284,1286,1283,1285,1284,1286
33307,1285,1286,1285,1286,1286,1285,1286,1285
36304,1286,1284,1286,1284,1286,1284,1286,1283
39302,1286,1284,1286,1284,1286,1284,1285,1285
42299,1284,1285,1284,1285,1284,1285,1284,1285
45297,1283,1285,1282,1285,1283,1285,1282,1284
48294,1282,1285,1283,1284,1283,1284,1283,1284
51292,1284,1282,1284,1283,1284,1282,1284,1283
54289,1285,1282,1284,1282,1283,1281,1284,1283
57287,1284,1282,1284,1282,1282,1282,1282,1283
60284,1282,1284,1282,1283,1281,1283,1281,1283
63282,1281,1283,1281,1283,1281,1283,1281,1283
66279,1282,1282,1282,1282,1282,1282,1282,1282
69277,1282,1281,1282,1280,1282,1280,1282,1280
72274,1282,1280,1282,1280,1282,1279,1282,1280
75272,1282,1280,1281,1280,1281,1280,1281,1280
```

Figure 6: Irradiance Readings.

Now having a test file with irradiance values, our next step was to convert the file into .csv format. To do so, we began by unplugging the Arduino Uno from the laptop and connecting it to the BeagleBone Black instead, through a USB connection. We then connected the Beagle to the laptop via a USB connection, as well as to a local ethernet port. Once connected, the Beagle was now running a DHCP server that could provide the laptop with an IP address. To locate the ethernet IP address of the Beagle, we then connected to Cloud9, a cloud-based integrated development environment that lets us write, run, and debug code over a web browser. To access this, we simply browsed to the web server on the board - “192.168.7.2” - and then selected “Cloud9 IDE”. Once in the IDE, we opened up a new terminal window and ran the “ifconfig” command to display the current network connections on the board. The output of this command is illustrated in Figure 7 below.



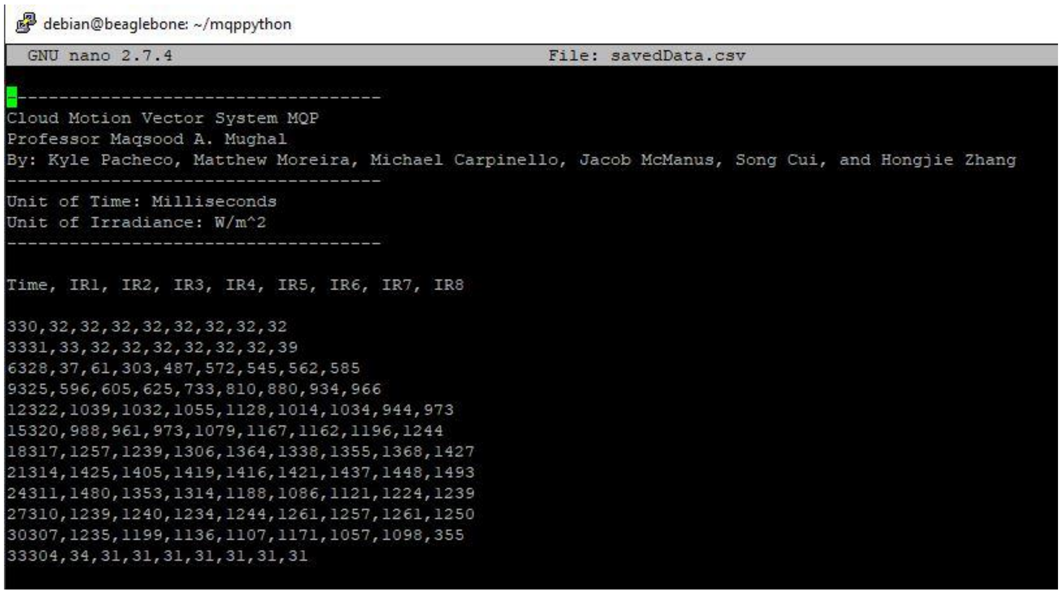
```
debian@beaglebone:/var/lib/cloud9$ ifconfig
eth0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet 192.168.1.91  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::e1c:57ff:fe0a:95ae  prefixlen 64  scopeid 0x20<link>
        inet6 2600:6c64:617f:eb17:e1c:57ff:fe0a:95ae  prefixlen 64  scopeid 0x0<global>
        ether 0c:1c:57:0a:95:ae  txqueuelen 1000  (Ethernet)
        RX packets 2476  bytes 304688 (297.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7607  bytes 10897000 (10.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 55

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 160  bytes 12960 (12.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 160  bytes 12960 (12.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.7.2  netmask 255.255.255.252  broadcast 192.168.7.3
        inet6 fe80::e1c:57ff:fe0a:95b0  prefixlen 64  scopeid 0x20<link>
        ether 0c:1c:57:0a:95:b0  txqueuelen 1000  (Ethernet)
        RX packets 1734  bytes 157776 (154.0 KiB)
        RX errors 0  dropped 4  overruns 0  frame 0
        TX packets 2292  bytes 2855294 (2.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 7: BeagleBone Black Network Connections.

From here, we then downloaded and ran PuTTY - an SSH client- to remotely connect to the Beagle using its ethernet IP address. Once in the bash shell, we updated to python3.8 and downloaded the pyserial library. We then created a python file called "csv.py" and inserted the code to save the data from the Arduino to a .csv file. Once we ran the python script, the data was saved as "savedData.csv" as shown below in Figure 8.



```
debian@beaglebone: ~/mqppython
  GNU nano 2.7.4                                          File: savedData.csv

-----------------------------------
Cloud Motion Vector System MQP
Professor Maqsood A. Mughal
By: Kyle Pacheco, Matthew Moreira, Michael Carpinello, Jacob McManus, Song Cui, and Hongjie Zhang
-----------------------------------
Unit of Time: Milliseconds
Unit of Irradiance: W/m^2
-----------------------------------

Time, IR1, IR2, IR3, IR4, IR5, IR6, IR7, IR8

330,32,32,32,32,32,32,32,32
3331,33,32,32,32,32,32,32,39
6328,37,61,303,487,572,545,562,585
9325,596,605,625,733,810,880,934,966
12322,1039,1032,1055,1128,1014,1034,944,973
15320,988,961,973,1079,1167,1162,1196,1244
18317,1257,1239,1306,1364,1338,1355,1368,1427
21314,1425,1405,1419,1416,1421,1437,1448,1493
24311,1480,1353,1314,1188,1086,1121,1224,1239
27310,1239,1240,1234,1244,1261,1257,1261,1250
30307,1235,1199,1136,1107,1171,1057,1098,355
33304,34,31,31,31,31,31,31,31
```

Figure 8: CSV File of Irradiance Data.

After collecting irradiance data, we decided to enclose each of the components in 3D printed cases. We found .stl files for both the BeagleBone Black and Arduino Uno on Thingiverse, a website committed to the sharing of user created digital-design files. Figure 9 below shows the Beagle and Arduino fit into their enclosures.

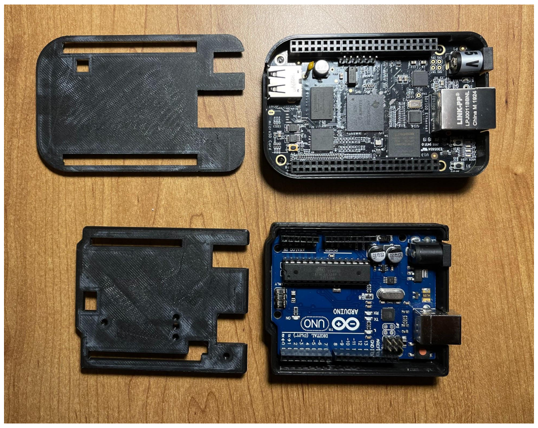


Figure 9: BeagleBone Black and Arduino Uno Enclosures.

In addition to these two enclosures, we also printed nine base plates to hold the sensors. Each base plate consisted of two screw holes and one rectangular cutout where the external wires could be soldered to the sensors. Once the sensors were set up on the base plates, we also printed top and bottom round enclosures to protect the sensors and two multiplexers from any inclement weather. Figure 10 below illustrates one enclosure in testing.

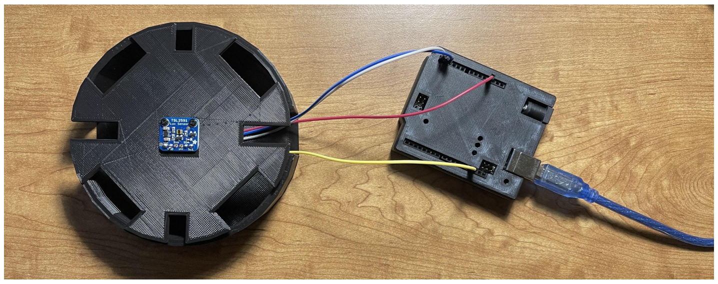


Figure 10: 3D Printed Sensor Setup.

3.3 Photovoltaic System

Part of the project was to also create a photovoltaic system that utilizes microprocessor data - collected from light sensors and predicted cloud movement- to know when to switch between a group of solar panels and a battery or generator when needed. As illustrated in Figure 11 below, the external devices we have contemplated using are lightbulbs.

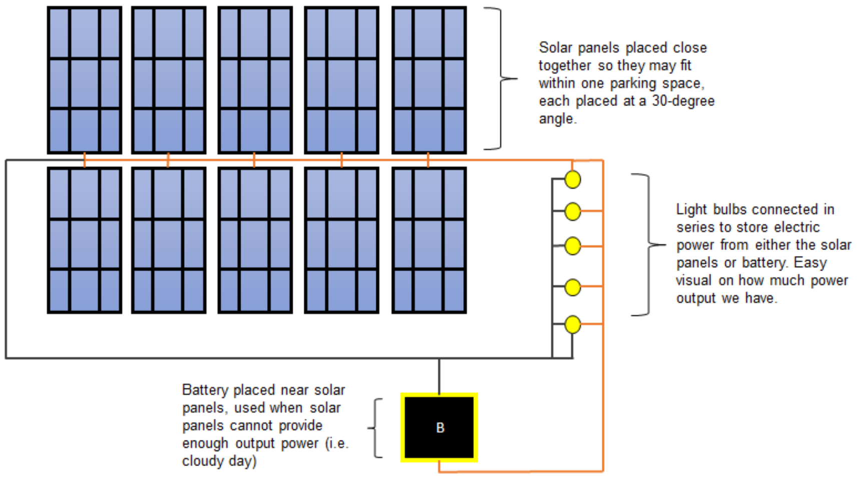

Solar panels placed close together so they may fit within one parking space, each placed at a 30-degree angle.
Light bulbs connected in series to store electric power from either the solar panels or battery. Easy visual on how much power output we have.
Battery placed near solar panels, used when solar panels cannot provide enough output power (i.e. cloudy day)
B

Figure 11: System Diagram Setup for Photovoltaic System.

We began by collecting ten solar panels that were used in a previous project, and implemented them into our project as a result of their seemingly great condition. Before setting up the solar system on top of East Hall Garage, we had to first thoroughly clean each panel. After being cleaned, we then needed to test each one to ensure that they registered a voltage output. We did this by individually connecting each solar panel's positive and negative ends to a multimeter. Fortunately, all of the solar panels registered some voltage output both when exposed and removed from light in the laboratory.

Once each solar panel had been tested and proven to work correctly, we had to find a way to raise the solar panels 30 degrees and to have them sit on a flat surface. After researching several different methods, we decided to order ten metal solar panel mounts from Lowe's that we would be able to put together ourselves. This was a cheaper option and allowed us to build the mounts specifically geared towards our solar panel size. A fully built solar panel mount connected to a solar panel raised to 30 degrees can be seen in Figure 12 below.

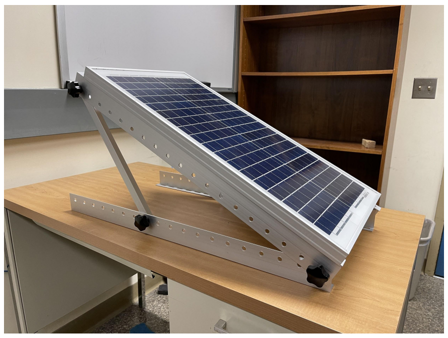


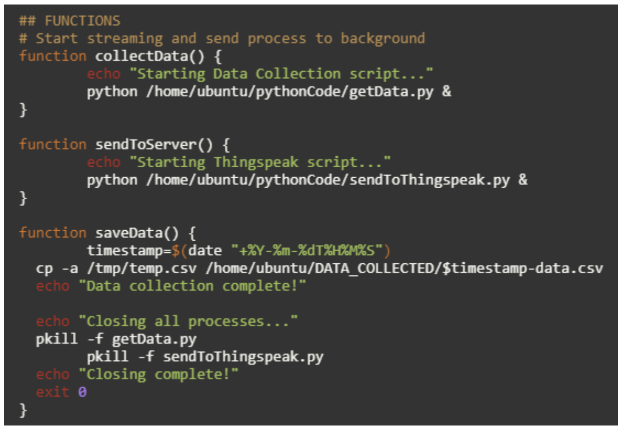
Figure 12: Solar Panel with Mount.

Moving forward, we set up the solar panels on the roof of East Hall Parking Garage. This was a spot on campus that allowed us to use a substantial amount of space to set up both the photovoltaic system described above as well as the sensor array. When combined, the solar panel set-up and the sensor array takes up around four parking spaces.

3.4 Thingspeak Server

When the irradiance data was collected from the sensors and compiled into a .csv file by the BeagleBone, we were then able to send this data to an external server. This external server was done using Thingspeak. Thingspeak is an open-source IoT (Internet of Things) analytics application and API used to store and receive data from devices using the HTTP and MQTT protocol over the Internet or Local Area Network (LAN). The Beaglebone Black used a LAN connection to send data to the Thingspeak server.

The benefit of using an external server such as Thingspeak is that it provided the ability to update data in real-time. This was done by utilizing a python script to send the .csv file collected from the BeagleBone to the server at a specific time interval. The python script, shown in Figure 13 below, shows two embedded python scripts: “getData.py” and “sendToThingspeak.py”. Both of these functions were necessary in sending real-time data to the server; however, this method did not utilize the import functionality built into our channel. Instead, we had to use our channel’s specific API key, shown in Figure 14 below. By using the API key, we were able to send data directly to the server without having to log into Thingspeak itself. Using the API key also ensured continuous updated information at the time intervals specified in the python scripts.



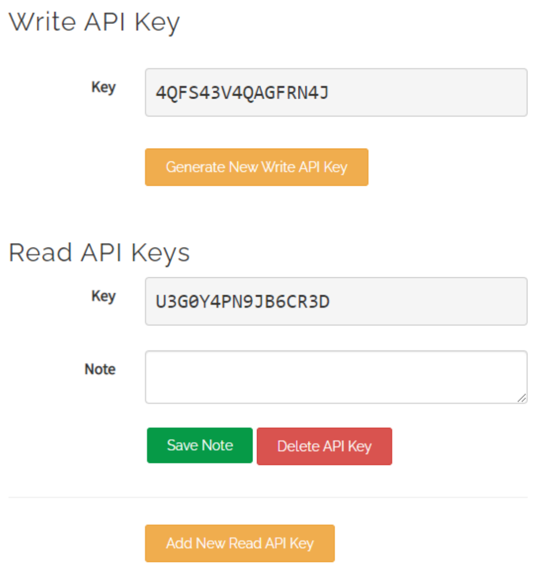
```
## FUNCTIONS
# Start streaming and send process to background
function collectData() {
        echo "Starting Data Collection script..."
        python /home/ubuntu/pythonCode/getData.py &
}

function sendToServer() {
        echo "Starting Thingspeak script..."
        python /home/ubuntu/pythonCode/sendToThingspeak.py &
}

function saveData() {
        timestamp=$(date "+%Y-%m-%dT%H%M%S")
  cp -a /tmp/temp.csv /home/ubuntu/DATA_COLLECTED/$timestamp-data.csv
  echo "Data collection complete!"

  echo "Closing all processes..."
  pkill -f getData.py
        pkill -f sendToThingspeak.py
  echo "Closing complete!"
  exit 0
}
```

Figure 13: Python Script to Send Data to Server.



Write API Key

Key: 4QFS43V4QAGFRN4J

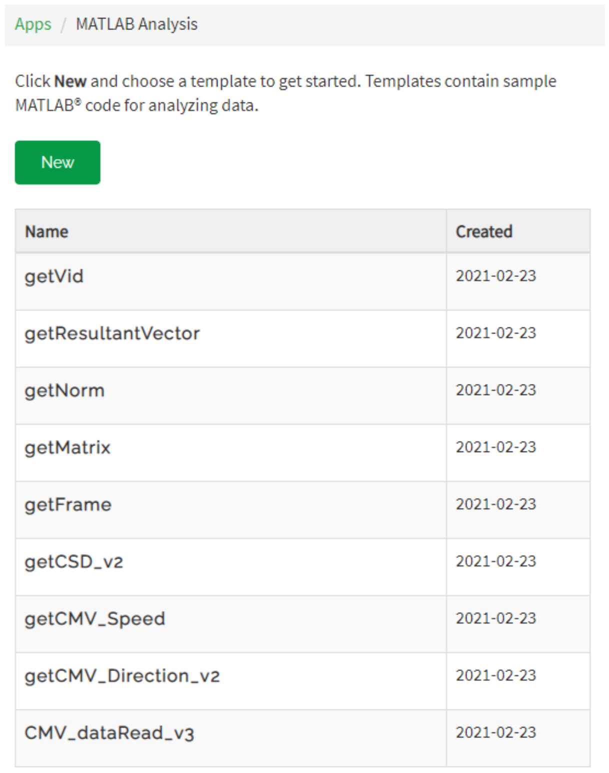
Read API Keys

Key: U3G0Y4PN9JB6CR3D

Note:

Figure 14: Thingspeak Server API Keys.

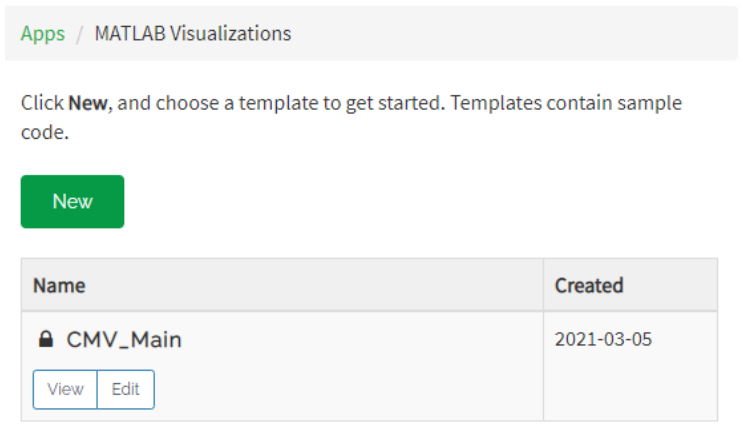
With Thingspeak being an IoT analytics server, it not only stored the data from the .csv file, but also analyzed it. Thingspeak in particular has access to MATLAB analytics. All of the MATLAB scripts and algorithms used to analyze the behavior of the clouds were compiled together in one section under the "MATLAB Analysis" tab of Thingspeak shown in Figure 15 below. Once the data was analyzed with these scripts, Thingspeak was also capable of creating visualizations of the data. Under the "MATLAB Visualizations" tab shown in Figure 16 below, the MATLAB script "CMV_Main" took all of the cloud behavior analytics and plotted them.


Apps / MATLAB Analysis

Click **New** and choose a template to get started. Templates contain sample MATLAB® code for analyzing data.

Name	Created
getVid	2021-02-23
getResultantVector	2021-02-23
getNorm	2021-02-23
getMatrix	2021-02-23
getFrame	2021-02-23
getCSD_v2	2021-02-23
getCMV_Speed	2021-02-23
getCMV_Direction_v2	2021-02-23
CMV_dataRead_v3	2021-02-23

Figure 15: Thingspeak MATLAB Analysis Tab.


Apps / MATLAB Visualizations

Click **New**, and choose a template to get started. Templates contain sample code.

Name	Created
CMV_Main	2021-03-05

Figure 16: Thingspeak MATLAB Visualizations Tab.

With Thingspeak providing all of this functionality in one place, the data was not only being sent to the server in real-time, but it was also being analyzed and plotted by the MATLAB scripts in real time.

3.5 Cloud Motion Simulation Parameters

To predict cloud motion, specifically speed and direction, there are several different algorithms that can be used, including Linear Cloud Edge Method, Gradient Matrix Method, and the Peak Matching Method.

First, we have the Linear Cloud Edge Method. [2] This method requires a triplet of light sensors (at least) to determine the cloud speed and direction. This method also requires two cloud events to take place in a relatively short period of time. The light sensor data is then collected and expanded upon using the following mathematical equations:

To calculate the cloud shadow speed:

$$v = \frac{D}{t_{cB}\sin\alpha + t_{cA}\cos\alpha} \tag{7}$$

To calculate the angle alpha (a) above:

$$\alpha = \tan^{-1}\left[-\frac{d_{cB}\sin\theta\,(t_{cA2} - t_{cA1})}{d_{cA}(t_{cB2} - t_{cB1}) - d_{cB}\cos\theta\,(t_{cA2} - t_{cA1})}\right] \tag{8}$$

When using sensors that are perpendicular to one another, the above angle equation can be rewritten and simplified to:

$$\alpha = \tan^{-1}\left[-\frac{d_{cB}(t_{cA2} - t_{cA1})}{d_{cA}(t_{cB2} - t_{cB1})}\right] \tag{9}$$

The next method to calculate cloud direction and speed is the Gradient Matrix Method. This method is based more on cloud edge detection. In this method, the data collected from the light sensors is transformed into a 3x3 matrix and is processed much the same as image analysis. Then, once the 3x3 matrix is made, it is convoluted with the Sobel Filter. [3] The Sobel Filter, or Sobel operator, is a matrix used in image processing and digital vision, specifically with edge detection algorithms. Once these two matrices are convolved, then the result is two magnitudes labeled Gx and Gy. Now taking these two magnitudes and using the equation below, we can now solve for the angle, theta.

$$\theta = \tan^{-1}[\frac{G_y}{G_x}] \tag{10}$$

The last method is the Peak Matching Method, which also integrates the angle calculated from the Gradient Matrix Method. This method simply uses the correlation between speed, distance, and time in the following equation:

$$v_{AB} = \frac{D}{t_{AB}} \tag{11}$$

For this algorithm, it utilizes data from a pair of sensors, calculating the time lag in between the cloud coverage of one sensor to the next -accounting for t_AB in the equation- and is then used to calculate the speed of the cloud for that pair. [3] Now, taking data from multiple pairs of sensors utilizing the same equation, along with the angle from the Gradient Matrix Method, it will yield the estimated cloud shadow speed.

We then proceeded to use the provided MATLAB code containing the algorithms located in Appendix B to calculate cloud speed and direction. We first tested the MATLAB code on our PCs to ensure that .csv data could be collected and displayed in graphical format. We used the nine sensor array with the algorithms to collect irradiance data, the .csv is then opened in MATLAB and we were able to receive graphical representation of cloud speed and direction.

Once this was completed, we then moved toward sending the .csv file to Thingspeak utilizing the specific API key designated for our server. On Thingspeak, all of the MATLAB files are placed in the 'MATLAB Analysis' section aside from the main file, which is placed in the 'MATLAB Visualization' since it controls creating the graphs. The following graphs represent the data collected from running the .csv on a PC:

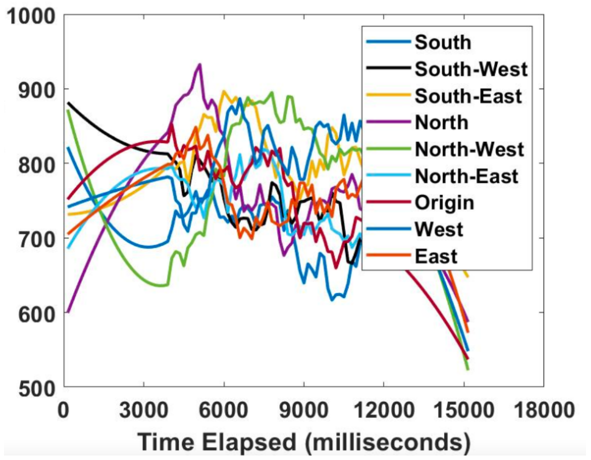


Figure 17: Graphical representation of irradiance data over time collected from the nine sensor array. The nine sensors are labeled based on location around an origin in a circular scheme.

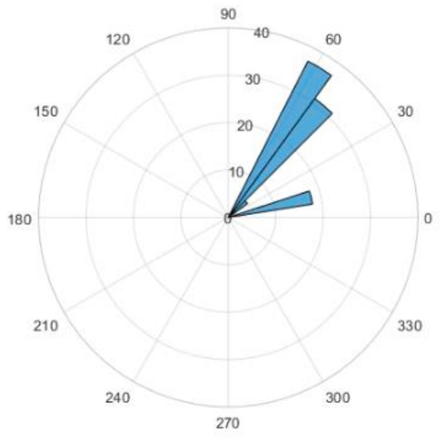


Figure 18: Graphical representation of the direction of the cloud in conjunction with the locations of the sensors.

4. Results

Once we collected irradiance data through the Arduino, we were able to upload the MATLAB code to the Thingspeak server. We then built our system and began testing. As shown in Figure 19 below, we gained access to the rooftop of East Hall Garage to set-up our sensor array. As evidenced below, we had the 8 sensors placed at varying degrees around the 9th sensor at the origin. Each sensor was also equipped with their own baseplate. We then also had the multiplexers and Arduino at the origin.

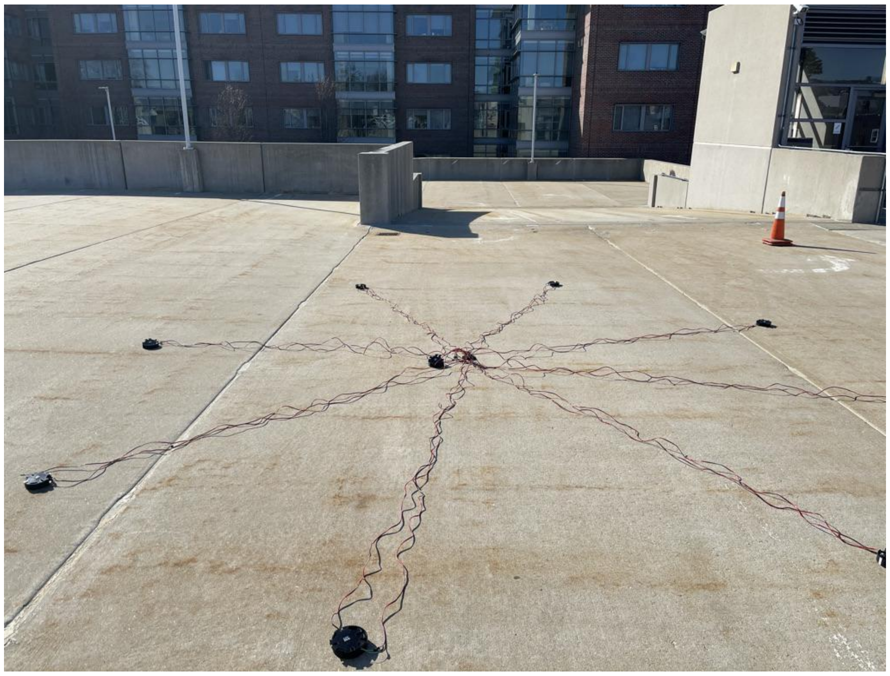


Figure 19: Sensor Array set-up on top of East Hall garage. Includes nine sensors, two multiplexers, and an Arduino.

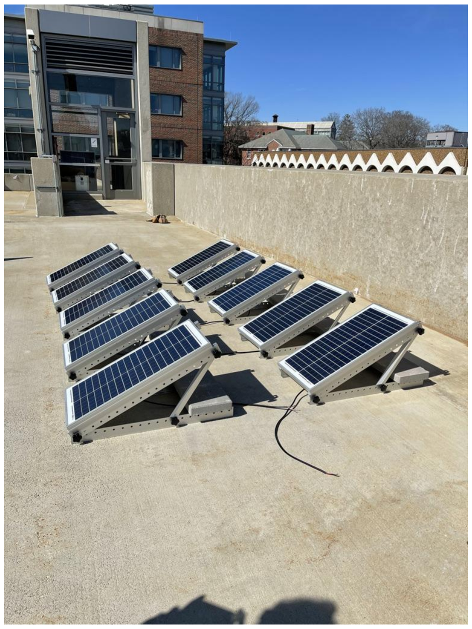


Figure 20: Solar panel set-up on top of the East Hall garage - panels are set up in two rows of five and are weighed down by cement blocks.

In addition to the set-up of the sensor array on the rooftop of East Hall garage, we also installed an array of solar panels, as shown in Figure 20 above. These solar panels are set at a 30 degree angle - utilizing metal stands to hold them up - and are each weighed down by a single cement block to protect from strong winds.

From our array system of sensors, we then connected the Arduino within the array to our laptop. From here, we then were able to run the Arduino scripts to collect irradiance data under direct sunlight. We then proceeded to move around the array and cover different sensors with our shadows to imitate cloud movement. Once we collected a sufficient amount of irradiance data, we were then able to take the saved .csv file, and upload this to the Thingspeak MATLAB scripts.

Once we were able to run the MATLAB scripts on the .csv file, we were able to produce the first set of graphs in Figure 21 and 22 below, which shows the individual irradiance for each sensor, as well as the graphs in Figures 23. Figure 23 below shows the irradiance data with noise taken into account in the top left, however, filtering this out we then arrive at the graph in the top right. This graph represents the irradiance (W/m^2) data collected without normalization. It is also filtered without the noise that was present in the previous graph. Once we normalized the data, the Y-axis was scaled down to one instead of thousands, as represented in the bottom graph.

We believe the data collected is much higher than it should be, ranging around the 10,000 W/m^2 mark instead of 1,000 W/m^2. We believe this may have been caused by an error in the Arduino software where the calculation for irradiance is located. To test the system we covered a sensor in shade for a period of time then moved onto the next sensor and repeated this process, resulting in the fluctuation of irradiance in the graphs. Every sensor is color coded in terms of data represented and is also labeled as directions on the compass and the origin.

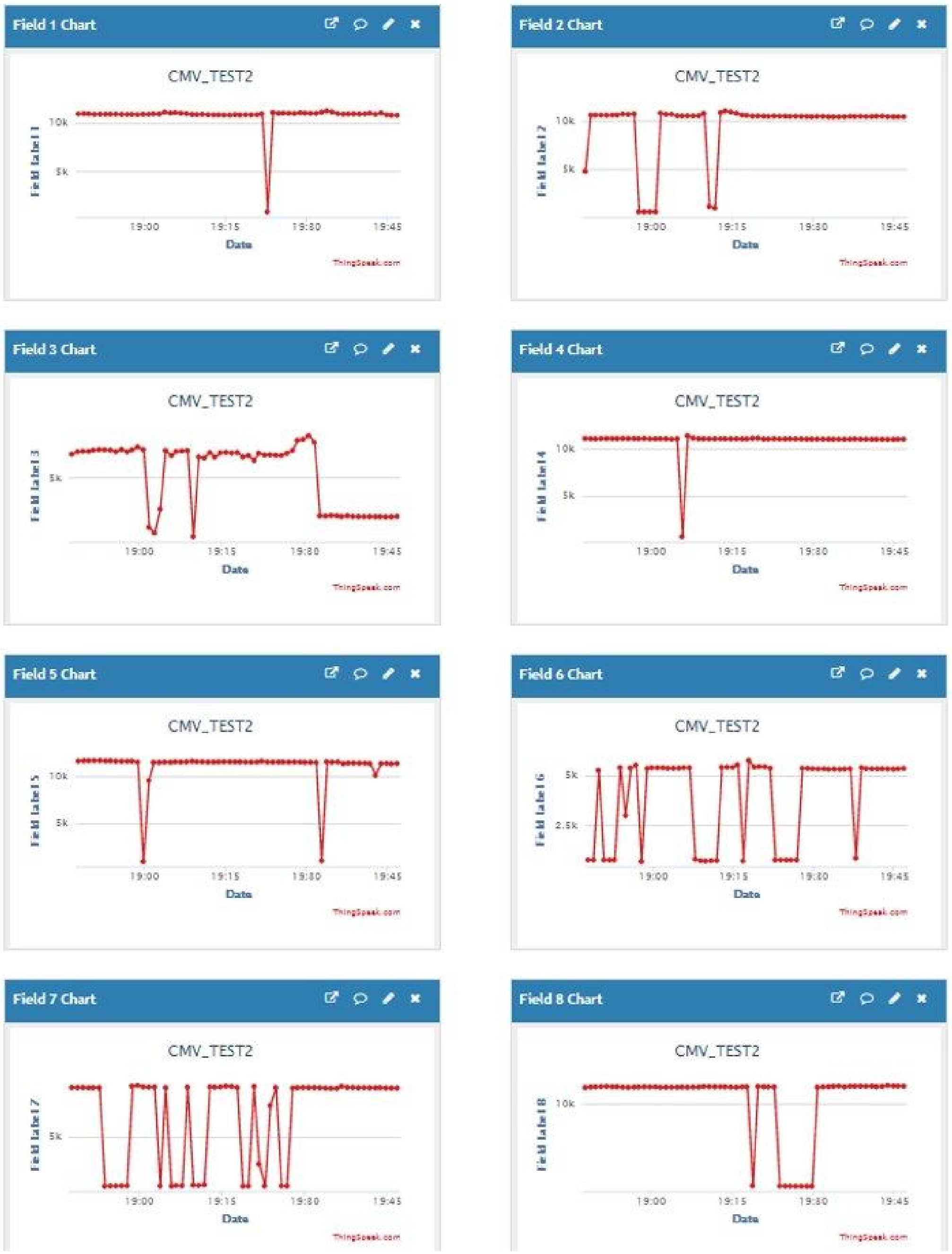


Figure 21: Individual Irradiance data for each sensor - eight totals sensors in channel 1.

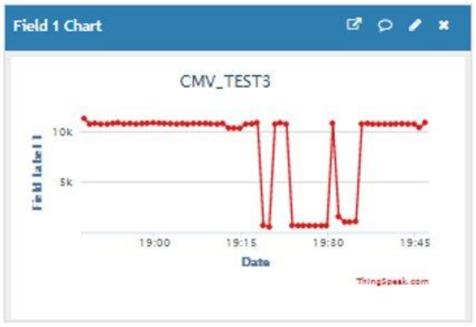


Figure 22: Individual Irradiance data for the ninth sensor located in channel 2.

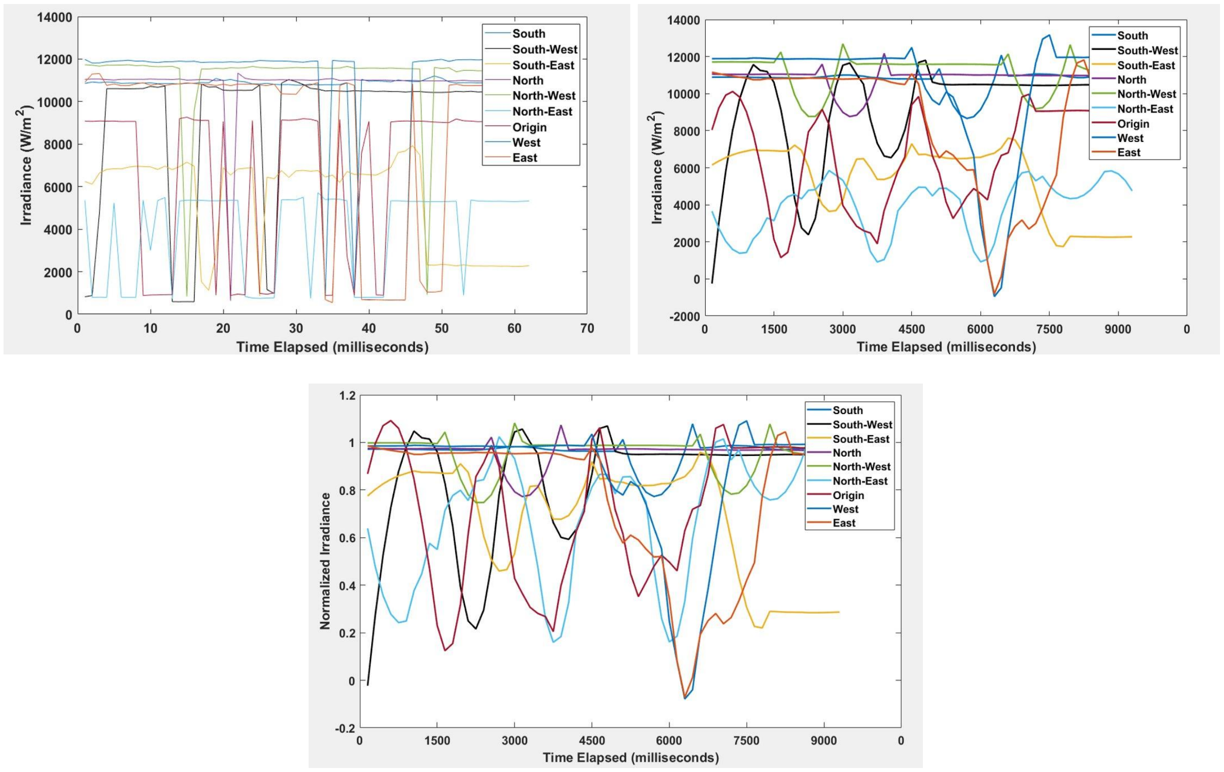


Figure 23 : Irradiance Data collected represented with/without noise filtering, and normalized irradiance.

When we received the irradiance data, we were also able to construct polar histograms utilizing the degree of the sensors as well as the irradiance data. These polar histograms have the corresponding degree to each sensor located around the graph, so the shaded blue areas of the graph represent the direction in which the clouds or shadows are moving. For example, in Figure 24 on the left, the shadow is moving directly West at 180 degrees. However, these do not show the speed of the cloud. Using the time the cloud takes to pass from sensor to sensor and also the distance found by getting the direction, we are able to calculate the speed of the cloud using the simple speed = distance/time equation, as shown in Figure 25. Since we know the speed and the direction of the cloud as well as the distance to our CMVS design. We then are able to calculate the time of arrival of the cloud over the CMVS design by using equation (12), where 'a' is the calculated angle:

$$\text{Time of Arrival} = (\text{Distance}/\text{Speed}) * \text{Cos}(a) \tag{12}$$

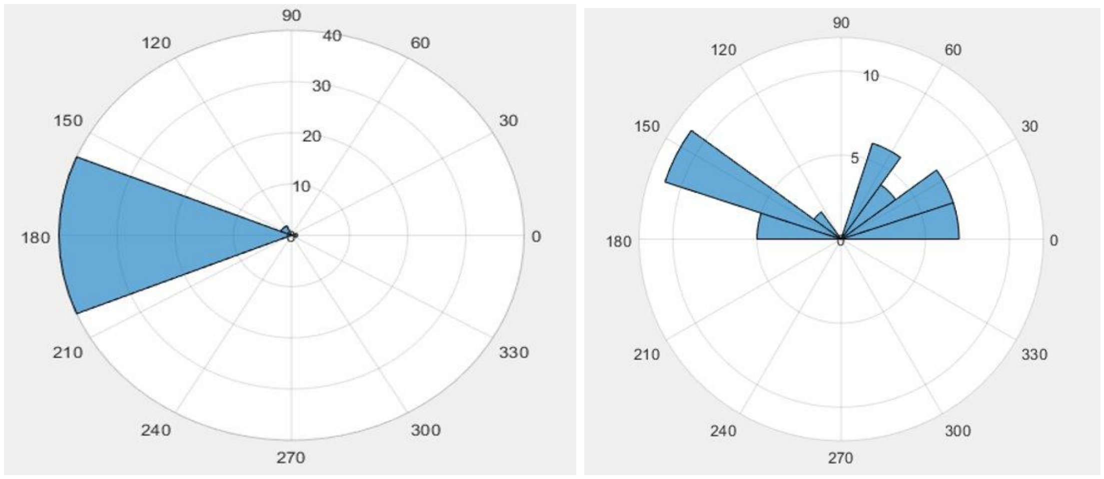
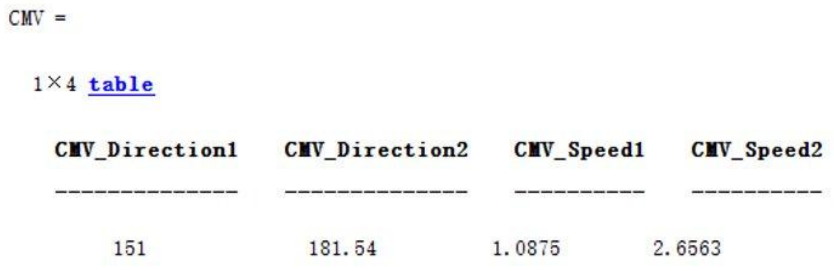


Figure 24: Polar Histograms created from sample testing data.



```
CMV =

  1×4 table

    CMV_Direction1    CMV_Direction2    CMV_Speed1    CMV_Speed2
    ______________    ______________    __________    __________

         151              181.54          1.0875        2.6563
```

Figure 25: Speed (CMV_Speed1, CMV_Speed2) calculated from two sample directions of shadows (CMV_Direction1, CMV_Direction2).

5. Conclusion

In an attempt to shift towards implementing renewable energy resources into communities across the globe, having a system that can detect clouds before they pass over a PV site and determine what impact those clouds will have on the PV site, is compelling to have. This paper introduced a Cloud Motion Vector System that detects cloud motion simulation parameters. The system calculates the speed and direction of the cloud, as well as how much that cloud affects solar irradiance. Knowing this change in solar irradiance, as well as the maximum output power of the PV site that this system is predicting, could also allow the system to predict the change in output power of the PV site because of cloud coverage. If the predicted power falls below the power needed from that PV site, a backup power supply could be switched on to account for the lack of power. The results show that there is a large impact on the magnitude of solar irradiance when a cloud passes over the array of sensors; therefore, potentially having a similar effect on the output power of a PV site.

If the CMVS design is implemented close to a PV site, it can enable grid operators to better understand and mitigate the effects of PV power variability on grid planning and operations - making this model extremely robust. This is important towards preventing voltage fluctuation, flicker, and potential equipment malfunction.

6. Future Work

In addition to the work that we have completed, there are several additional measures that can be taken to improve the system. The first step is to integrate the CMS model from the 2019-2020 MQP project into this system in order to accomplish tasks such as calculating short term flicker severity index and generating irradiance versus time plots. Furthermore, theft prevention and maintenance due warnings could be implemented into the system to further protect the system from being stolen or malfunctioning as a result of a faulty component. These warnings could either come up as messages in the software side of the system or as actual sound once theft or malfunction is detected. Next, estimating cloud thickness is an additional calculation that can be included in the system. This will be crucial in determining the light intensity of direct versus indirect sunlight. Lastly, overlapping sensor clusters could increase the accuracy of predicting cloud movement. As a result of our project only having one cluster of light sensors besides a PV site, it was quite difficult to measure the time of arrival of a cloud at the site. This is due to the fact that if the cluster of light sensors were to be placed on the right side of the PV site for instance, if the clouds were to come in from the left side of the PV site, by the time the sensor cluster were to detect the clouds, it may already be over the PV site; ultimately, not allocating enough time to switch on the backup power source.

7. Bibliography

[1] Aslanian Jr, Barry, et al. *Study On Impacts of Varying Weather Patterns Upon Circuits with Photovoltaic Penetration.* : Worcester Polytechnic Institute, 2020.

[2] Bosch, J. L., and J. Kleissl. "Cloud Motion Vectors from a Network of Ground Sensors in a Solar Power Plant." *Maeresearch.ucsd.edu*, MAE Research, maeresearch.ucsd.edu/kleissl/pubs/BoschKleisslSE2013_CloudSpeed.pdf.

[3] HyperChiicken. "Cloud Motion Vector System for Solar Power Forecasting." *Hackster.io*, 14 June 2019, www.hackster.io/HyperChiicken/cloud-motion-vector-system-for-solar-power-forecasting-5c4a86#toc-clouds-are-complicated-6.

[4] IEEE Recommended Practice for the Analysis of Fluctuating Installations on Power Systems. , 2015, doi:10.1109/IEEESTD.2015.7317469.

[5] "Information and Technical Requirements for the Interconnection of Distributed Energy Resources (DER).", Jan 21, 2020, https://www.eversource.com/content/docs/default-source/builders-contractors/der-information-technical-requirements.pdf?sfvrsn=ab2bfc62_10

[6] Thomas Keppler, Neville Watson, and Jos Arrillaga. *Computation of the Short-Term Flicker Severity Index*, IEEE, 2000.

[7] "Welcome." *Eversource*, www.eversource.com/content/general/about/about-us/about-us/our-future-is-clean-energy/commitment-to-clean-energy-carbon-neutrality.

[8] "Welcome." *Eversource*, www.eversource.com/content/general/about/about-us/about-us/our-future-is-clean-energy/renewable-generation.

8. Appendices

Arduino Code

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_TSL2591.h"

Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591);

void TCA9548A1(uint8_t bus){
  Wire.beginTransmission(0x71);  // TCA9548A address is 0x70
  Wire.write(1 << bus);          // send byte to select bus
  Wire.endTransmission();
}

void TCA9548A2(uint8_t bus){
  Wire.beginTransmission(0x70);  // 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) {
  Serial.begin(9600);
  configureSensor();
}

void advancedRead(void){
  uint32_t lum = tsl.getFullLuminosity();
  uint16_t ir, full;
  ir = lum >> 16;
  full = lum & 0xFFFF;
  Serial.print(ir);
}

void loop(void) {
  for (int i = 0; i < 8; i++)
  {
    TCA9548A1(i);
    advancedRead();
    Serial.print(F(","));
  }

  for (int i = 0; i < 8; i++) {
    TCA9548A2(i);
    if (i==0){
      advancedRead();
    }
  }
  Serial.println();
  delay(100);
}
```

Bash Script

```
#!/usr/bin/env bash
# This bash script runs 2 python scripts to stream the data from the TM4C123 (TIVA-C)
# then logs the data into an individual CSV at a given hour

## VARIABLES
# end time at 5am UTC = 10pm PST
end_hour=5

## FUNCTIONS
# Start streaming and send process to background

function collectData() {
		echo "Starting Data Collection script..."
		python /home/ubuntu/pythonCode/getData.py &
}

function sendToServer() {
		echo "Starting Thingspeak script..."
		python /home/ubuntu/pythonCode/sendToThingspeak.py &
}

function saveData() {
		timestamp=$(date "+%Y-%m-%dT%H%M%S")
 cp -a /tmp/temp.csv /home/ubuntu/DATA_COLLECTED/$timestamp-data.csv
 echo "Data collection complete!"

 echo "Closing all processes..."
 pkill -f getData.py
		pkill -f sendToThingspeak.py
 echo "Closing complete!"
 exit 0
}

## MAIN
echo Starting Cloud Motion Vector sensor...
# Delete file if it exists
if [[ -e /tmp/temp.csv ]]; then
		rm /tmp/temp.csv
		echo "Renewing temp file complete!"
fi
```

```
collectData
sleep 3
sendToServer

while [ 1 ]; do
        current_hour=$(date +%k)
        if (( current_hour==end_hour )); then
                saveData
        else
                sleep 60
        fi
done
```

Python Scripts

getData.py

```
#!/usr/bin/env python
from datetime import datetime
import serial, io, time, shutil

outFile = '/tmp/temp.csv'
ser = serial.Serial(
        port='/dev/ttyACM0',
        baudrate=115200,
)

sio = io.TextIOWrapper(
        io.BufferedRWPair(ser, ser, 1),
        encoding='ascii', newline='\r\n'
)

# Give time to initialize port
time.sleep(1)

# "1" here is the start command for the TIVA-C to start collecting data
ser.write('1\n')

with open(outFile,'a') as f: #appends to existing file
        while ser.isOpen():
                datastring = sio.readline()
                f.write(datetime.utcnow().isoformat() + ',' + datastring)
                f.flush() #included to force the system to write to disk
#ser.close()
```

sendToThingspeak.py

```
#!/usr/bin/env python
from collections import deque
import csv, time, schedule
import os
import sys
import urllib            # URL functions
import urllib2           # URL functions

filename="/tmp/temp.csv"

THINGSPEAKKEY = 'INSERTYOURTHINGSPEAKKEYHERE'
THINGSPEAKURL = 'https://api.thingspeak.com/update'

def getLast(csv_filename):
   with open(csv_filename, 'r') as f:
       try:
           lastrow = deque(csv.reader(f), 1)[0]
       except IndexError:  # empty file
           lastrow = None
       return lastrow

def getLast_v2(csv_filename):
   with open(csv_filename, 'r') as f:
       lastrow = None
       for lastrow in csv.reader(f): pass
               return lastrow

def sendData(url,key,field1,field2,field3,field4,field5,field6,field7,field8,
                        data1,data2,data3,data4,data5,data6,data7,data8):
 """
 Send event to internet site
 """
 values = {
                        'api_key' : key,
                        'field1' : data1,
                        'field2' : data2,
                        'field3' : data3,
                        'field4' : data4,
                        'field5' : data5,
                        'field6' : data6,
                                'field7' : data7,
                                'field8' : data8
}

 postdata = urllib.urlencode(values)
 req = urllib2.Request(url, postdata)

 log = time.strftime("%d-%m-%Y,%H:%M:%S") + ","
 log = log + data1 + ","
 log = log + data2 + ","
 log = log + data3 + ","
 log = log + data4 + ","
 log = log + data5 + ","
 log = log + data6 + ","
 log = log + data7 + ","
 log = log + data8 + ","

 try:
   # Send data to Thingspeak
   response = urllib2.urlopen(req, None, 5)
   html_string = response.read()
   response.close()
   log = log + 'Update ' + html_string

 except urllib2.HTTPError, e:
   log = log + 'Server could not fulfill the request. Error code: ' + e.code
 except urllib2.URLError, e:
   log = log + 'Failed to reach server. Reason: ' + e.reason
 except:
   log = log + 'Unknown error'

 print log
if __name__ == "__main__":

        while 1:
                try:
                        temp_data=getLast(filename)
                        data={
                                "SOUTHWEST":temp_data[1],
                                "SOUTH":temp_data[2],
                                "SOUTHEAST":temp_data[3],
                                "WEST":temp_data[4],
                                "ORIGIN":temp_data[5],
                                "EAST":temp_data[6],
                                "NORTHWEST":temp_data[7],
                                "NORTH":temp_data[8],
                                "NORTHEAST":temp_data[9]
                        }

                #print data
                        sendData(THINGSPEAKURL,THINGSPEAKKEY,'field1','field2','field3',
                                'field4','field5','field6','field7','field8',
                                data['SOUTHWEST'],data['SOUTH'],data['SOUTHEAST'],data['WEST'],
                                data['EAST'],data['NORTHWEST'],data['NORTH'],data['NORTHEAST'])
                        time.sleep(60)
                except:
                        pass
```

MATLAB Algorithms

CMVS_Main.m

```
clc;clear;close all;
filepath = 'C:\Users\dell\Desktop\CMV_Research\MATLAB\MATLAB_scripts\data.csv';     %set folder location to save figures
save_filepath = 'C:\Users\dell\Desktop\CMV_Research\MATLAB\MATLAB_scripts\';

data1 = thingSpeakRead(1248525,'Fields',[1,2,3,4,5,6,7,8],'NumPoints',200,'ReadKey','EQ4MUCOL8YTU4EBS');
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','EQ4MUCOL8YTU4EBS');
   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 = 100;
   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);

%% Plot and save datasets
   figure
   plot1 = plot(data);
   ylim([0 inf]);
   set(plot1(2),'DisplayName','South-West','LineWidth',2,'Color','k');
   set(plot1(1),'DisplayName','South','LineWidth',2);
   set(plot1(3),'DisplayName','South-East','LineWidth',2);
   set(plot1(8),'DisplayName','West','LineWidth',2);
   set(plot1(7),'DisplayName','Origin','LineWidth',2);
   set(plot1(9),'DisplayName','East','LineWidth',2);
   set(plot1(5),'DisplayName','North-West','LineWidth',2);
   set(plot1(4),'DisplayName','North','LineWidth',2);
   set(plot1(6),'DisplayName','North-East','LineWidth',2);
   set(gca,'FontSize',15,'fontweight','bold');
   % Enlarge figure to full screen.
   set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.04, 1, 0.96]);
   xlabel('Time Elapsed (milliseconds)')
   ylabel('Irradiance (W/m^2)')
   % modify labels for tick marks
```

```
xticks = get(gca,'xtick');
scaling  = 150;
newlabels = arrayfun(@(x) sprintf('%d', scaling * x), xticks, 'un', 0);
set(gca,'xticklabel',newlabels);
legend('show');
% Save figure and image in folder
saveas(gca, fullfile(save_filepath,'Dataset'),'fig');
saveas(gca, fullfile(save_filepath,'Dataset'),'png');
close

% Plot framed dataset
figure
plot1 = plot(data_sample);
set(plot1(2),'DisplayName','South-West','Color','k');
set(plot1(1),'DisplayName','South');
set(plot1(3),'DisplayName','South-East');
set(plot1(8),'DisplayName','West');
set(plot1(7),'DisplayName','Origin');
set(plot1(9),'DisplayName','East');
set(plot1(5),'DisplayName','North-West');
set(plot1(4),'DisplayName','North');
set(plot1(6),'DisplayName','North-East');
set(gca,'FontSize',15,'fontweight','bold')
xlabel('Time Elapsed (milliseconds)')
ylabel('Irradiance (W/m^2)')

% Plot smoothed framed dataset
figure
plot1 = plot(cmv_sample);
set(plot1(2),'DisplayName','South-West','LineWidth',2,'Color','k');
set(plot1(1),'DisplayName','South','LineWidth',2);
set(plot1(3),'DisplayName','South-East','LineWidth',2);
set(plot1(8),'DisplayName','West','LineWidth',2);
set(plot1(7),'DisplayName','Origin','LineWidth',2);
set(plot1(9),'DisplayName','East','LineWidth',2);
set(plot1(5),'DisplayName','North-West','LineWidth',2);
set(plot1(4),'DisplayName','North','LineWidth',2);
set(plot1(6),'DisplayName','North-East','LineWidth',2);
set(gca,'FontSize',15,'fontweight','bold')
xlabel('Time Elapsed (milliseconds)')
ylabel('Irradiance (W/m^2)')
legend('show');
% modify labels for tick marks
xticks = get(gca,'xtick');
scaling  = 150;
newlabels = arrayfun(@(x) sprintf('%d', scaling * x), xticks, 'un', 0);
set(gca,'xticklabel',newlabels)
% Save figure and image in folder
saveas(gca, fullfile(save_filepath,'CMV_Sample'),'fig');
saveas(gca, fullfile(save_filepath,'CMV_Sample'),'png');
close

figure
plot1 = plot(cmv_sample_norm);
```

```
    set(plot1(2),'DisplayName','South-West','LineWidth',2,'Color','k');
    set(plot1(1),'DisplayName','South','LineWidth',2);
    set(plot1(3),'DisplayName','South-East','LineWidth',2);
    set(plot1(8),'DisplayName','West','LineWidth',2);
    set(plot1(7),'DisplayName','Origin','LineWidth',2);
    set(plot1(9),'DisplayName','East','LineWidth',2);
    set(plot1(5),'DisplayName','North-West','LineWidth',2);
    set(plot1(4),'DisplayName','North','LineWidth',2);
    set(plot1(6),'DisplayName','North-East','LineWidth',2);
    set(gca,'FontSize',15,'fontweight','bold')
    xlabel('Time Elapsed (milliseconds)')
    ylabel('Normalized Irradiance')
    legend('show');
    % modify labels for tick marks
    xticks = get(gca,'xtick');
    scaling  = 150;
    newlabels = arrayfun(@(x) sprintf('%d', scaling * x), xticks, 'un', 0);
    set(gca,'xticklabel',newlabels)
    % Save figure and image in folder
    saveas(gca, fullfile(save_filepath,'CMV_Sample_Norm'),'fig');
    saveas(gca, fullfile(save_filepath,'CMV_Sample_Norm'),'png');
    close

%% 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
    figure
    for sensor_idx = 1:
        sensor_inv = 1./cmv_sample_norm(:,sensor_idx);
        [dip,dlocation] =
        findpeaks(sensor_inv,'MinPeakProminence',peak_prominence,'MinPeakDistance',peak_distance,'NPeaks',1);

        if isempty(dlocation)
            dlocation_arr(sensor_idx) = 0;
        else
            dlocation_arr(sensor_idx) = dlocation;
        end

        hold on
        plot2(sensor_idx) = plot(t,cmv_sample_norm(:,sensor_idx),'DisplayName','Origin Sensor','LineWidth',2);
        plot(t(dlocation),1./dip,'rs','markersize',10);
    end
    hold off
    set(plot2(2),'DisplayName','South-West','LineWidth',2,'Color','k');
    set(plot2(1),'DisplayName','South','LineWidth',2);
    set(plot2(3),'DisplayName','South-East','LineWidth',2);
    set(plot2(8),'DisplayName','West','LineWidth',2);
    set(plot2(7),'DisplayName','Origin','LineWidth',2);
    set(plot2(9),'DisplayName','East','LineWidth',2);
    set(plot2(5),'DisplayName','North-West','LineWidth',2);
    set(plot2(4),'DisplayName','North','LineWidth',2);
    set(plot2(6),'DisplayName','North-East','LineWidth',2);
    set(gca,'FontSize',15,'fontweight','bold')
    xlabel('Time Elapsed (milliseconds)')
    ylabel('Normalized Irradiance')
    % legend('show');
    % Modify labels for tick marks
    xticks = get(gca,'xtick');
    scaling  = 150;
    newlabels = arrayfun(@(x) sprintf('%d', scaling * x), xticks, 'un', 0);
    set(gca,'xticklabel',newlabels)
    % Save figure and image in folder
    saveas(gca, fullfile(save_filepath,'CMV_Sample_Norm'),'fig');
    saveas(gca, fullfile(save_filepath,'CMV_Sample_Norm'),'png');

%% 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;
        figure;quiver(Gx,-Gy) %invert to correct visual vector orientation
        GM(idx) = getframe(gcf);
        close
    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
    figure
    hist = polarhistogram(angle_rad,10);
    set(gca,'FontSize',10)
    saveas(gca, fullfile(save_filepath,'GM-1'),'fig');                  %save figure
    saveas(gca, fullfile(save_filepath,'GM-1'),'png');                        %save image

    % Algorithm 2.2
    [M, Phase_rad] = getResultantVector(magnitude,theta);
    Phase_deg = rad2deg(Phase_rad);

    % Plot polar histogram
    figure
    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]);
    set(gca,'FontSize',10)
    saveas(gca, fullfile(save_filepath,'GM-2'),'fig');                        %save figure
    saveas(gca, fullfile(save_filepath,'GM-2'),'png');                        %save image

%% 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];

    % Save CMV array to a text file
    fileID = fopen(strcat(save_filepath,'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);

end
```

getCMVS_Direction.m

```
%% This function gets the CMV direction using algorithm (1) without 2*pi wraparound or (2) with 2*pi wraparound
function CMV_Direction = getCMV_Direction_v2(angle_rad,hist_plot,algorithm)

[max_row,~] = size(angle_rad);
[~,edge_idx] = max(hist_plot.Values);
edgeValues = hist_plot.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
```

getCMVS_Speed.m

```
% This function receives the CMV direction and calculates the cloud shadow
% speed from the local minima locations
function CMV_Speed = getCMV_Speed(CMV_Direction,dlocation_arr)

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
        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
```

getResultantVector.m

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

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
```

getCSD_v2.m

```
%% This function gets the raw magnitude and theta converts to a corrected array
function outputArray = getCSD_v2(magnitude,theta,threshold)

[~,~,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
```

getMatrix.m

```
%% This function maps lux data into a selected matrix type
function outputArray = getMatrix(cmv_sample,window,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
```

getNorm.m

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

getFrame.m

```
function data_sample = getFrame(data,x_start,x_end)
%% This function gets a specified frame from x_start to x_end of the dataset.
data_length = x_end - x_start + 1;
data_sample = zeros(data_length,1,9);
for i = 1:9
   actual_temp = data(x_start:x_end,i);
   data_sample(:,:,i) = actual_temp;
end
data_sample = reshape(data_sample,data_length,9);
```

getQuadrant.m

```
function angle_label = getQUADRANT(array)

angle_label = cell(1);

for idx = 1:numel(array)
   if array(idx)>=0 && array(idx) < pi/2
      angle_label(idx, 1) = {'Quadrant1'};
   elseif array(idx)>=pi/2 && array(idx) < pi
      angle_label(idx, 1) = {'Quadrant2'};
   elseif array(idx)>=pi && array(idx) < 3*pi/2
      angle_label(idx, 1) = {'Quadrant3'};
   elseif array(idx)>=3*pi/2 && array(idx) < 2*pi
      angle_label(idx, 1) = {'Quadrant4'};
   else
      angle_label(idx, 1) = {NaN};
  end end end
```

CMV_dataRead_v3.m

```
function data_raw = CMV_dataRead_v3(filepath)
%% This function grabs the filepath of the CMV sensor data and ouputs a table version of the data
%   data = output table version of original data
%   filepath = directory of the input file

fd = fopen(filepath, 'rt');
formatSpec = '%f %f %f %f %f %f %f %f %f';

% Transfer csv file data into data_raw array
data_raw = textscan(fd, formatSpec, 'Delimiter',{',','\n'}, 'CollectOutput', 1, 'EndOfLine', '\n');
fclose(fd); %close csv file
```