

5-2019

Mobile Solar Energy Environmental Control System with Remote Accessibility

Luis S. Garay
The University of Texas Rio Grande Valley

Follow this and additional works at: <https://scholarworks.utrgv.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Garay, Luis S., "Mobile Solar Energy Environmental Control System with Remote Accessibility" (2019).
Theses and Dissertations. 468.
<https://scholarworks.utrgv.edu/etd/468>

This Thesis is brought to you for free and open access by ScholarWorks @ UTRGV. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks @ UTRGV. For more information, please contact justin.white@utrgv.edu, william.flores01@utrgv.edu.

MOBILE SOLAR ENERGY ENVIRONMENTAL CONTROL SYSTEM
WITH REMOTE ACCESSIBILITY

A Thesis

by

LUIS S. GARAY

Submitted to the Graduate College of
The University of Texas Rio Grande Valley
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2019

Major Subject: Computer Science

MOBILE SOLAR ENERGY ENVIRONMENTAL CONTROL SYSTEM
WITH REMOTE ACCESSIBILITY

A Thesis
by
LUIS S. GARAY

COMMITTEE MEMBERS

Dr. Fitratullah Khan
Chair of Committee

Dr. Mahmoud Quweider
Committee Member

Dr. Hansheng Lei
Committee Member

May 2019

Copyright 2019 Luis S. Garay
All Rights Reserved

ABSTRACT

Garay, Luis S., Mobile Solar Energy Environmental Control System with Remote Accessibility.

Master of Science (MS), May, 2019, 53 pp., 8 tables, 20 figures, references, 42 titles.

The proposed system, *Mobile Solar Energy Environmental Control System with Remote Accessibility (mSEECs w/ RA)*, is based on the utilization of an already available *Energy Storage Unit* (ESU) in a *Transport Unit System* (TUS) to harness solar energy to provide a system to enhance the environment, safety and security of a TUS in a stationary or a mobile state by monitoring and controlling the operation of its onboard systems, and provide remote accessibility to know the state of affairs at any time. To relate to a real-world application, the battery (ESU) of a vehicle (TUS) is underutilized most of the time while being stationary in a parking lot. Adding a solar panel to the vehicle with a charge controller provides a charging mechanism for the already installed battery. This allows for the operation of onboard systems, such as air circulation and surveillance, which are normally shutdown to avoid draining the battery. However, this calls for a sophisticated control and monitoring system which needs to monitor the battery to avoid draining it beyond a certain percent depending on the type of battery, control onboard systems to keep them operational at a specified level, and at the same time send alerts and provide remote access to users. Using a *Raspberry Pi (RPi)*, a small hardened computer system, with add-on sensors to gauge key environmental variables, data can be retrieved from customized inputs and thresholds to generate signals to trigger other subsystems.

DEDICATION

I would like to dedicate this thesis and the completion of my master's studies to my family, especially, to my mother, Bertha Garay, my brothers J. Daniel Garay, and Xavier Garay, who motivated me and supported me throughout my master's career. Their patience and support went a long way as I worked on my degree. I couldn't have done this without you, thank you for your love and support.

ACKNOWLEDGMENTS

I would like to thank Dr. Fitratullah Khan, chair of my thesis committee, for his advice and mentorship. I learned a lot from this research while under his guidance and I am grateful for his patience throughout the process. A thanks also goes to the thesis committee members: Dr. Mahmoud Quweider and Dr. Hansheng Lei. I wouldn't have made it to this point without their support and the opportunities they, and Dr. Khan, have granted me while pursuing my degree and working on this research.

TABLE OF CONTENTS

	Page
ABSTRACT.....	iii
DEDICATION.....	iv
ACKNOWLEDGEMENTS.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	viii
LIST OF FIGURES.....	ix
CHAPTER I. INTRODUCTION.....	1
1.1 General Setup of mSEECs w/ RA.....	1
1.2 Raspberry Pi.....	3
1.3 User Interface.....	4
1.4 Database.....	4
1.5 Other Tools.....	6
CHAPTER II. METHODOLOGY.....	7
2.1 Overview of Design and Limitations.....	7
2.2 Detailed System Flow Procedure.....	8
2.3 ESSi Sensors.....	21
2.4 ESSO Devices.....	22
2.5 Interfacing ESSi Sensors and ESSO Devices.....	22
2.6 Reading ESSi Sensors.....	25
2.7 TUS Local State Buffer and Populating JSON.....	30

2.8 CMS Processing of JSON.....	30
2.9 Website.....	31
CHAPTER III. RESULTS.....	35
3.1 Mobile and Internet of Things Implementation.....	35
3.2 Data Collection and Data Visualization.....	35
3.3 Time and Space Complexity.....	36
CHAPTER IV. CONCLUSION, CONTRIBUTIONS, AND FUTURE WORK.....	39
4.1 Contributions.....	39
4.2 Conclusion and Future Work.....	40
REFERENCES.....	41
APPENDIX.....	42
BIOGRAPHICAL SKETCH.....	53

LIST OF TABLES

	Page
Table 1.1: Raspberry Pi 3 Model B Specifications.....	3
Table 1.2: Users Table.....	4
Table 1.3: Rpi Table.....	5
Table 1.4: Access Table.....	5
Table 1.5: Status Table.....	5
Table 1.6: Log Table.....	5
Table 1.7: Types Table.....	5
Table 1.8: Status Table.....	5

LIST OF FIGURES

	Page
Figure 1.1 General setup of mSEECs w/ RA.....	2
Figure 2.1 Flowchart for the Main Function.....	10
Figure 2.2 Flowchart for the Read from Sensors Function.....	11
Figure 2.3 Flowchart for the Notification Function.....	12
Figure 2.4 Flowchart for the Update Function – Send JSON Updates.....	14
Figure 2.5 Flowchart for the Update Function – Send Motion Updates.....	15
Figure 2.6 Flowchart for the Update Function – Send Clarity Update.....	16
Figure 2.7 Flowchart for the Camera Function - Monitoring.....	17
Figure 2.8 Flowchart for the Camera Function - Clarity.....	18
Figure 2.9 Flowchart for the Motion Capture Function.....	19
Figure 2.10 Clarity Image Before and After Binarization.....	21
Figure 2.11 Overall System Design.....	23
Figure 2.12 Circuit Design of SRNs, Voltage Followers and OpAmps for SV, SC, CC, LC, & LV.....	29
Figure 2.13 Format of JSON File.....	30
Figure 2.14 Website’s Login/Registration Page.....	32
Figure 2.15 Website’s Main User Page.....	32
Figure 2.16 Website’s Control Panel Page.....	33
Figure 2.17 Website’s Statistics View and Download Page.....	33

Figure 2.18 Website’s Log View Page.....34

Figure 2.19 Website’s Image View Page.....34

CHAPTER I

INTRODUCTION

Nowadays, many devices come equipped with various sensors and actuators that read and communicate the status of a device or respond to changes in the environment of a device to maintain a specific state. Although the core of such systems is not new, the emergence of mobile devices and a large focus on connectivity to the internet, an area known as the *Internet of Things (IoT)*, opens the system to mobile and IoT implementations. Thus, the proposed system, *Mobile Solar Energy Environmental Control System with Remote Accessibility (mSEECs w/ RA)*, takes on a mobile and IoT approach to an *Environmental Control System (ECS)*.

1.1 General Setup of mSEECs w/RA

A *Raspberry Pi (RPi)* is at the heart of the proposed system where an RPi interfaces with different sensors and actuators to maintain a certain specified state of a *Transport Unit System (TUS)*. The sensors gauge *Environmental, Safety, and Security Input (ESSI)* variables such as temperature, humidity, motion and the level of an *Energy Storage Unit (ESU)*, and the actuators trigger *Environmental, Safety, and Security Output (ESSO)* subsystems such as exhaust and camera. The RPi communicates with a *Central Management System (CMS)* sending updates on the overall state of affairs such as ESSI readings and ESSO states. The RPi receives updates from the CMS regarding changes in trigger thresholds, execution of certain commands, or execution of operations. There is a local buffer in the RPi consisting of intended pending transactions with the CMS, in order to account for losing its network connection with the CMS.

ESSI sensors and ESSO devices interface with the RPi through the RPi's *General Purpose Input Output (GPIO)* bus and *Universal Serial Bus (USB)*. Interfacing circuitry massages the connectivity with ESSI sensors and ESSO devices. RPi unit's operations are coded in Python, a general-purpose programming language. The RPi unit and the CMS communicate using the *JavaScript Object Notation (JSON)* file format. Transactions of JSON files use the *File Transfer Protocol (FTP)*, rather than the recommended *Secured FTP (SFTP)* due to limitations of the platform hosting the CMS selected for prototyping the proposed system. In a production environment, one should use SFTP. mSEECs w/ RA is accessible through a website. The system is also accessible locally through the RPi in case of losing its network connection with the CMS.

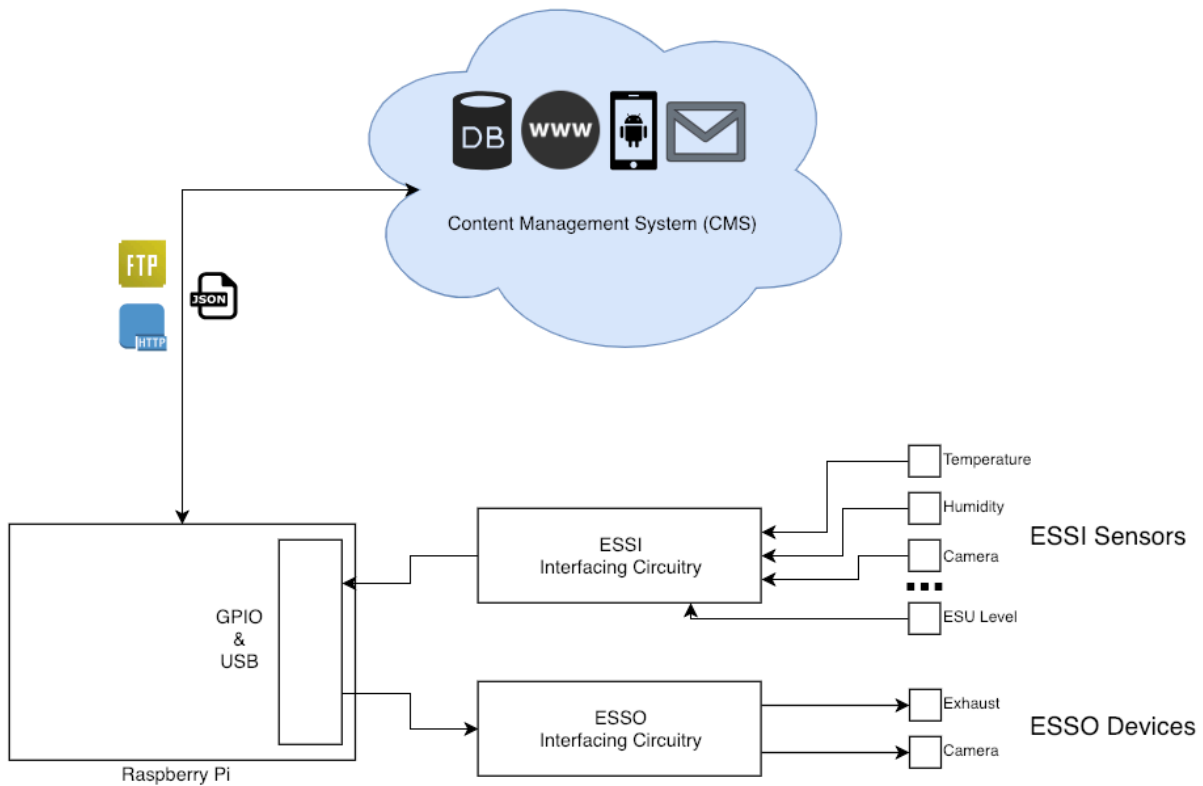


Figure 1.1 General setup of mSEECs w/ RA

The proposed system design accommodates multiple TUS instances. To relate this to a real-world example, the CMS can handle multiple vehicles. Each TUS is identified by its RPi ID

in the CMS database. A user may have multiple TUS instances registered to manage and control. Or, multiple users may be allowed to register a single TUS instance.

1.2 Raspberry Pi

As the system relies on the TUS’s ESU for power, using a desktop or laptop is not a reliable option when considering long-term environments due to their poor portability and high-power consumption. Therefore, the use of a device that requires low power, yet has the similar capabilities of a desktop/laptop, is imperative [9]. The device selected for the proposed system is a Raspberry Pi. An RPi is a small, *Single-Board Computer (SBC)* with wired and wireless connectivity that consists of several input/output buses to allow for communication between local and internet devices. This system uses the Raspberry Pi 3 Model B as its performance in processing and accessing GPIO is higher compared to its predecessors at the time of this research [10]. The relevant specifications of this RPi are Broadcom BCM2837 *System-on-Chip (SoC)*, Quadcore ARM Cortex-A53 processor, one gigabyte low-power DDR2 RAM, 2.4GHz 802.11n wireless card, 40 GPIO pin header, and four USB 2.0 ports [10].

Table 1.1 – Raspberry Pi 3 Model B Specifications

Raspberry Pi 3 Model B	
System-on-Chip	Broadcom BCM2837
CPU	Quadcore ARM Cortex-A53, 1.2GHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Networking	10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
GPIO	40-pin header
Relevant Ports	4x USB 2.0

1.3 User Interface

The CMS facilitates all communications for providing user interfaces for a website. The website's back-end is coded in *Hypertext Preprocessor (PHP)*, a server-side scripting language, while the website's front-end is coded in *Hypertext Markup Language (HTML)*, a standard markup language, and *JavaScript (JS)*, a high-level interpreted programming language. The website allows a user to view the transactions between a TUS, registered to a user, and the CMS. The transactions consist of items such as status updates, alarms (login attempts, threshold breaches, etc.), and changes made by the user. The website lets a user manage and control a TUS by changing items such as altering ESSI sensor threshold values and ESSO actions.

1.4 Database

The CMS incorporates a backend *MySQL*, an open source relational database management system, as the database system. The MySQL database consists of seven tables:

- Vitals – Keeps record of every TUS vital ESSI and respective thresholds
- Status – Keeps record of every TUS current status
- Access – Keeps record of users and corresponding TUS instances that they can access
- Users – Keeps record of every user
- Rpi – Keeps record of every RPi
- Log – Keeps record of all transactions
- Types – Keeps names of transaction types used in logs

All tables are read from by PHP scripts with embedded SQL queries running on the CMS and tables such as Vitals, Status, and Log are written into by PHP scripts on the CMS. Tables 1.2 to 1.8 show how data is stored in each table:

Table 1.2 – Users Table

uid	username	password
1	John Doe	sha1("password")
2	Jane Doe	sha1("password")

Table 1.3 – Rpi Table

rpId	rpiname	password	uid-owner
1	“RaspberryPi1”	sha1(“password”)	1

Table 1.4 – Access Table

uid-owner	uid-accessor	read-perm	write-perm
1	2	true	false

Table 1.5 – Status Table

vid	vn	v1	vu	rpId
1	InsideTemperature	25	30	1
2	ESUVoltage	12	15	1
3	GPS	None	None	1

Table 1.6 – Log Table

lid	vid	typ	uid	rpId	v1	v2	ts
1	1	ST	1	1	27	None	YYYY-MM-DD HH:MM:SS
2	2	ST	1	1	13.47	None	YYYY-MM-DD HH:MM:SS
3	3	ST	1	1	lat	lon	YYYY-MM-DD HH:MM:SS

Table 1.7 – Types Table

typenames
ST
LA
OK

Table 1.8 – Status Table

vid	v1	v2	ts	rpId
1	27	None	yyyy-mm-dd hh:mm:ss	1
2	13.47	None	yyyy-mm-dd hh:mm:ss	1
3	lat	lon	yyyy-mm-dd hh:mm:ss	1

1.5 Other Tools

To read and convert the analog values from the sensors to its digital equivalent, an *Analog to Digital Converter (ADC)* is used. More specifically, an 8-channel 10-Bit ADC is used to provide enough channels for all the ESSI sensors leaving room for growth.

Powering the proposed system drains the onboard ESU of a TUS. Therefore, a solar panel, a set of photovoltaic cells converting the sun's energy to *Direct Current (DC)* electricity, and a charge controller is used to charge the onboard ESU.

CHAPTER II

METHODOLOGY

The ECS revolves around the RPi which acts as a communication hub within the operating TUS. The RPi communicates with the CMS to store, update, and retrieve data. The RPi processes all data before any of it is sent out to the CMS or ESSO devices. The RPi is also capable of storing data in itself via a local buffer, but is only temporary until it can reconnect to a network to perform transactions with the CMS.

2.1 Overview of Design and Limitations

The multipurpose capabilities of an RPi, its credit card size, and low power draw reduces the overall hardware, space, and power requirements for the ECS. As a result, any other design considerations are left to the ESSI and ESSO, which many modern devices already come with. Using a real-world example, a typical modern vehicle contains a dozen and nearly 100 *Electronic Control Units (ECUs)*. These ECU's control mechanical parts or read/display information of a vehicle [5]. The proposed system uses external ESSI and ESSO, but integration using the onboard ESSI and ESSO of a TUS may vary depending on the hardware.

A separate cloud platform hosts the CMS, thus the RPi uses wireless communication to communicate and perform transactions with the CMS. The CMS hosts both the website and the database. Communication and transactions between the RPi and CMS use the two protocols, FTP and HTTP. In a production setting, it is recommended to avoid using FTP and HTTP; instead one should use SFTP and *HTTP-Secure (HTTPS)* as they provide a layer of security in

the transaction of files and information. Furthermore, it is recommended that the database and website not be located on the same machine; instead to give a layer of security both should be on separate machines in case the server hosting the website is compromised, the database is not compromised as a result.

2.2 Detailed System Flow Procedure

The RPi runs three major functions in parallel via multithreading:

- *Main function* - Handles the communication between the CMS and hardware
 - Requesting/Updating thresholds
 - Querying the ESSI sensors for updated readings
 - Querying the ESSO devices to do certain actions dependent on the ESSI readings and thresholds
- *Update function* - Handles the transactions between the RPi and CMS
 - Sending status updates to the CMS
 - Sending motion detection updates to the CMS
- *Camera function* - Handles the operations between the RPi and the Camera
 - Monitors for motion outside of the TUS
 - Determines the *Clarity of the Sky (CotS)*

2.2.1 Initialization

Initially, the main function starts by creating local buffers that temporarily store data if the RPi is unable to send data to the CMS. The main function also initializes RPi ID to mark all transactions as coming from a particular RPi/TUS and initializes the local buffers and variables of the other major and minor functions. This prevents the other functions from having to re-initialize when they run, resulting in saved time and better program efficiency. Finally, the main function starts the camera function on a separate thread, then enters the 'Update Interval' loop which has been described in the section below.

2.2.2 Main Function - Update Interval

At the outset of the update interval, the main function starts contacting the CMS via HTTP request to retrieve any updates on the TUS' ESSi thresholds. The main function receives the thresholds via FTP in the form of a response to a JSON request. If the main function cannot connect to the CMS, it uses a default set of thresholds or a previously received set of thresholds. This condition check is represented by the following pseudo-code:

- Request thresholds update from CMS
- If updated thresholds are received: continue with updated thresholds
- Else if updated thresholds are not received:
 - If previous thresholds are available: Refer to those thresholds
 - Else: Refer to the default thresholds

Next, the main function loads the thresholds from the JSON response and passes them to a separate function, *Read from Sensor (RfS)*, that is still within the main function. The RfS function reads all ESSi sensor values and checks the status of all ESSO devices followed by building a dictionary with those values and statuses. Lastly, RfS passes the dictionary to the main function to create a JSON file for the CMS to process. Finally, at the end of the update interval, the update function starts. While the update function is running on a separate thread, the main function goes into an idle state for one minute and then restarts at the beginning of the update interval. Figure 2.1 shows the flowchart of the main function.

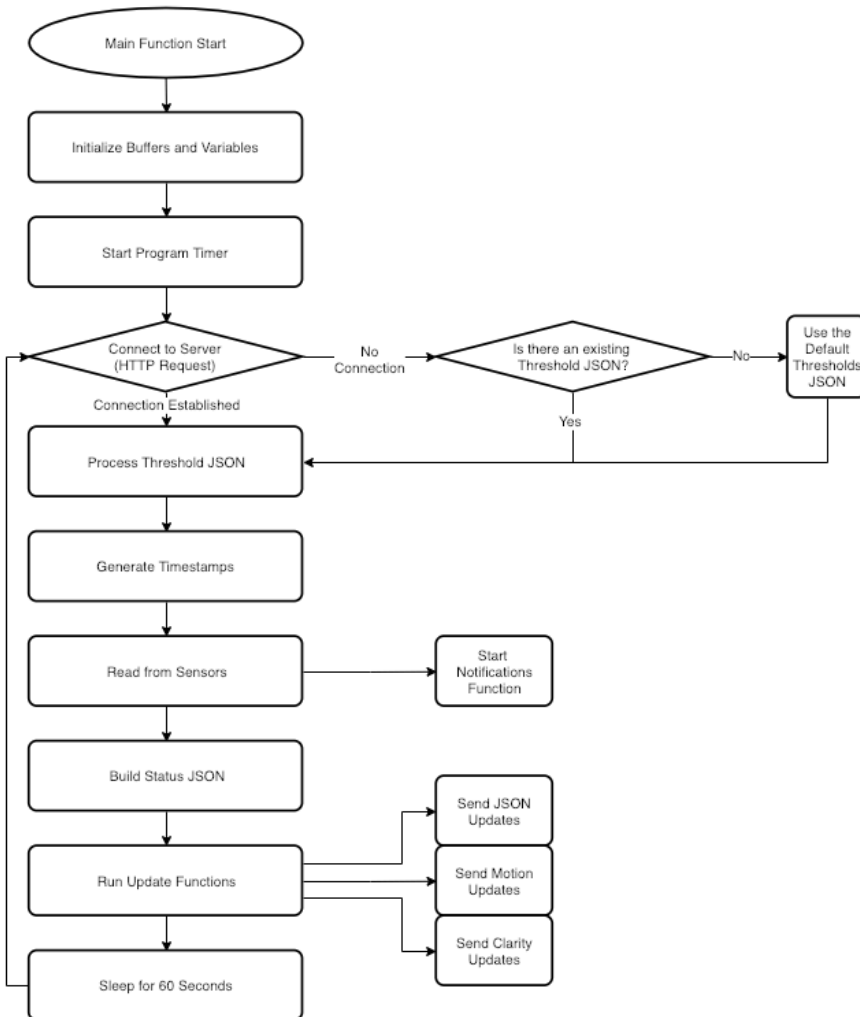


Figure 2.1 Flowchart for the Main Function

2.2.3 Main Function - Read from Sensors

The main function calls RfS with a set of thresholds as an argument. RfS uses these thresholds to compare with the values it reads from the ESSI sensors to check for threshold breaches and to determine what operations to do with specific ESSO devices. RfS also passes these values and thresholds to a separate function called *Notification Function (NF)*. For any threshold breaches, NF contacts the CMS via HTTP request with the offending value and the respective ESSI name requesting the CMS to notify the owner of the TUS via email. Finally, RfS

builds a dictionary using the acquired ESSI values and ESSO statuses and returns it to the main function. Figures 2.2 and 2.3 show the flowcharts of RfS and NF.

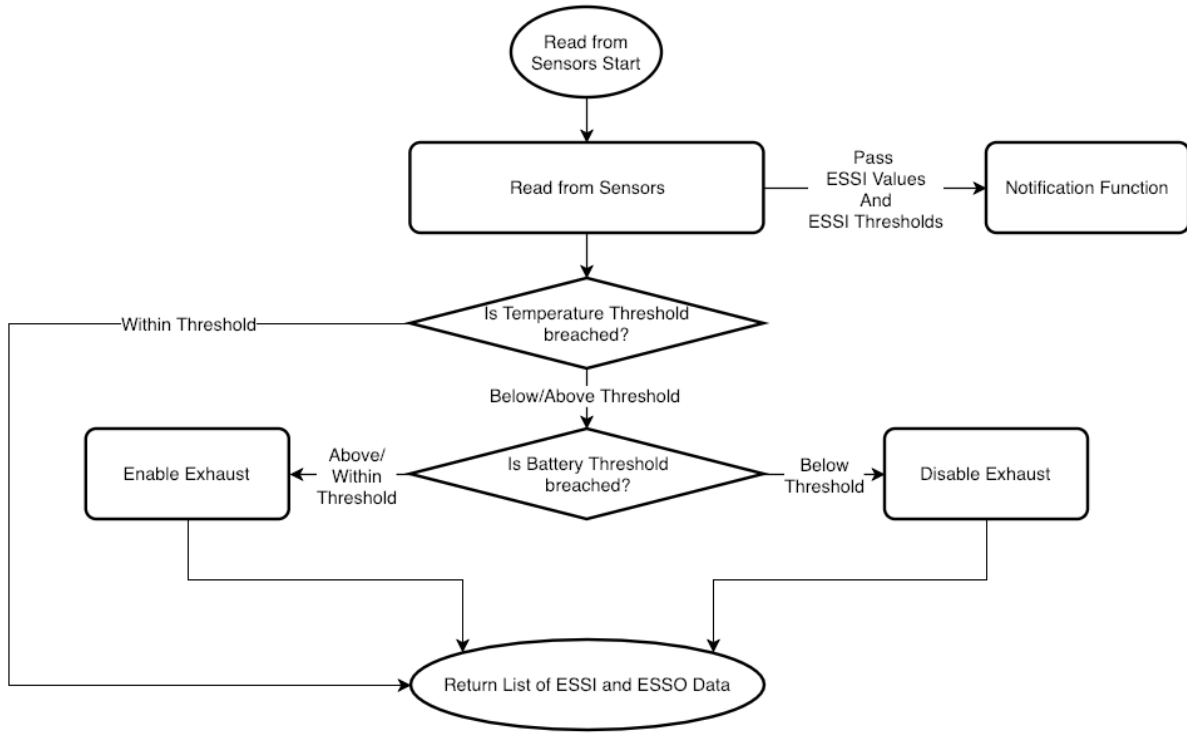


Figure 2.2 Flowchart for the Read from Sensors Function

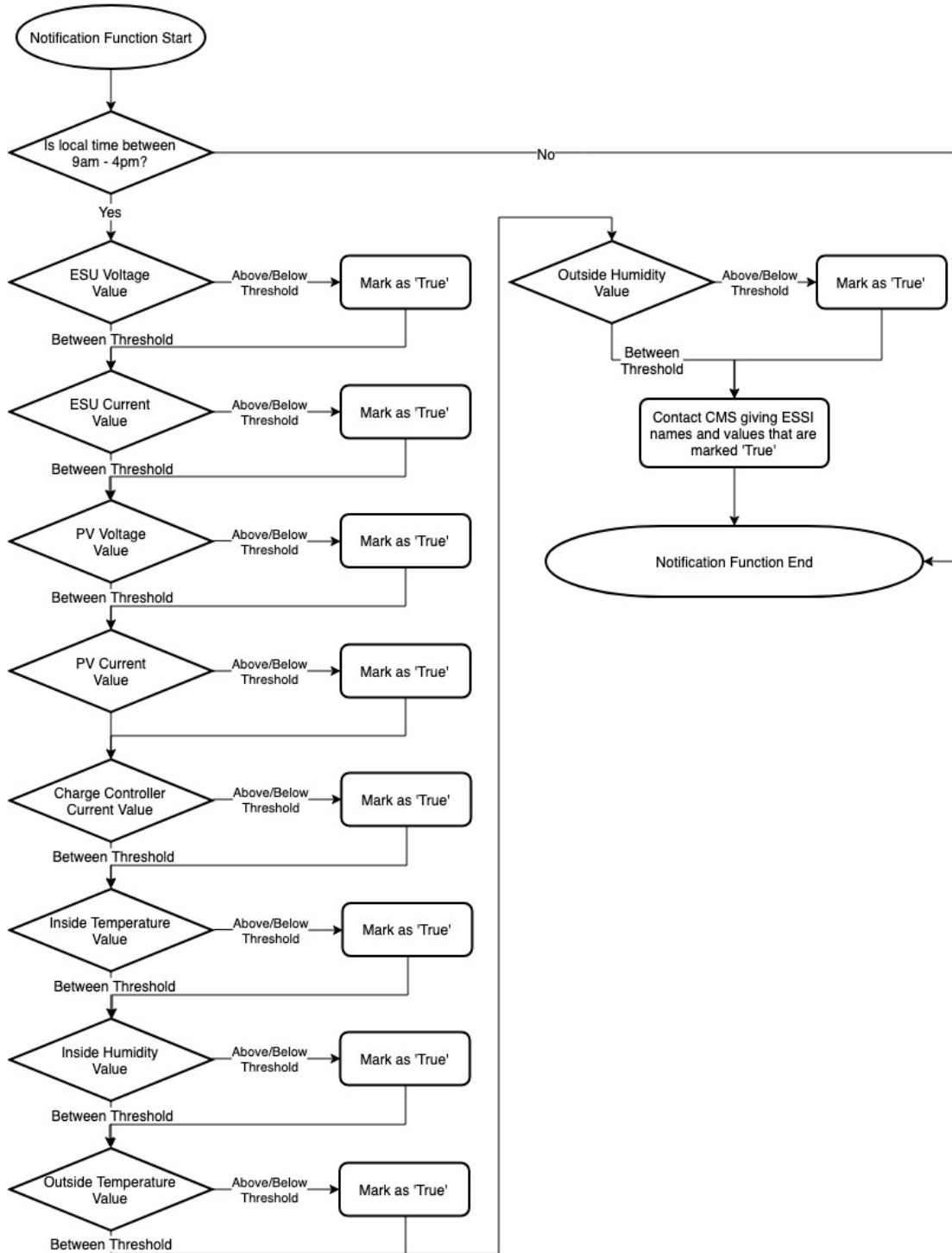


Figure 2.3 Flowchart for the Notification Function

2.2.4 Update Function

The update function consists of three sub-functions: *Send JSON Updates (SJU)*, *Send Motion Updates (SMU)*, and *Send Clarity Updates (SCU)*. Each of the functions send the CMS different content, however, similar steps are followed. Thus, the following explanation is common to the three sub-functions.

Initially, the function attempts to connect the CMS via FTP. When a connection is established the function sends its contents to the server. After every transfer, the function contacts the CMS via HTTP request, requesting for confirmation whether the file has successfully been transferred. If the CMS returns an “OK,” the function removes the file from the local buffer and checks if any more files are remaining in the buffer. If there are more files remaining, the process starts again and continues until the buffer is empty. In the case of an error at any point in the function, the file and any remaining files remain in the buffer until the next update interval occurs. Finally, when the function’s associated buffer is empty, the function ends and waits for the next update interval. Figures 2.4, 2.5, and 2.6 show the flowcharts of each respective function.

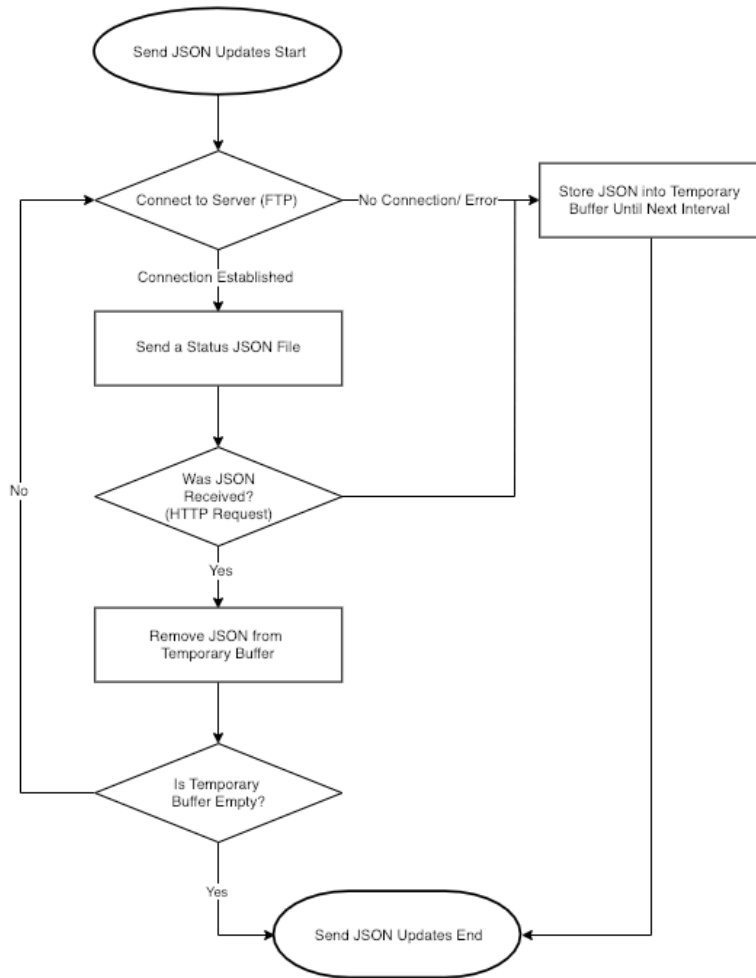


Figure 2.4 Flowchart for the Update Function – Send JSON Updates

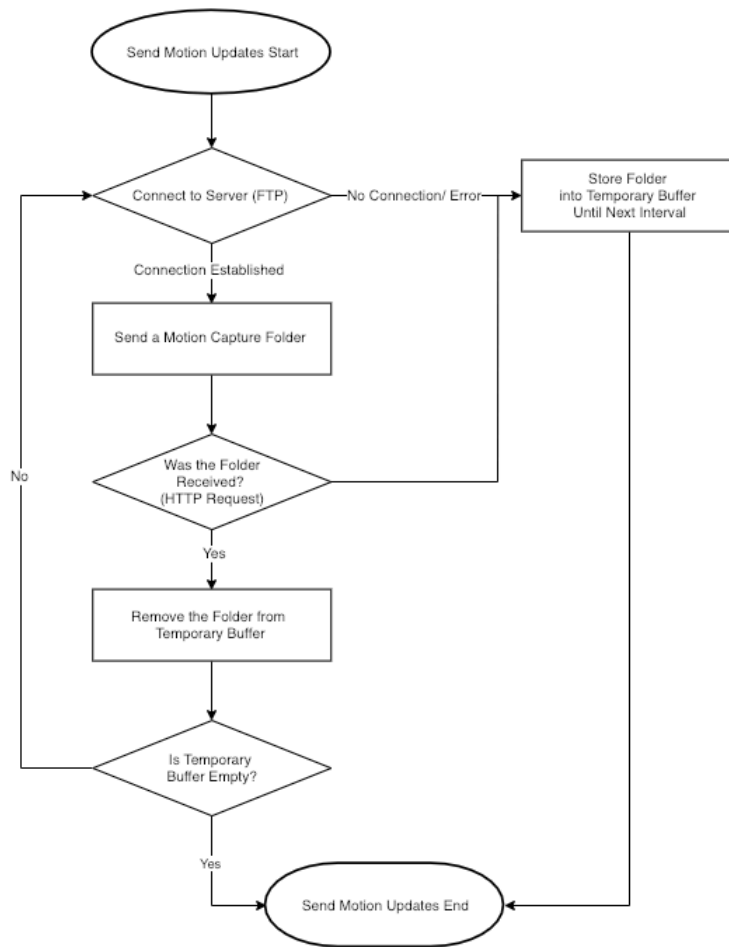


Figure 2.5 Flowchart for the Update Function – Send Motion Updates

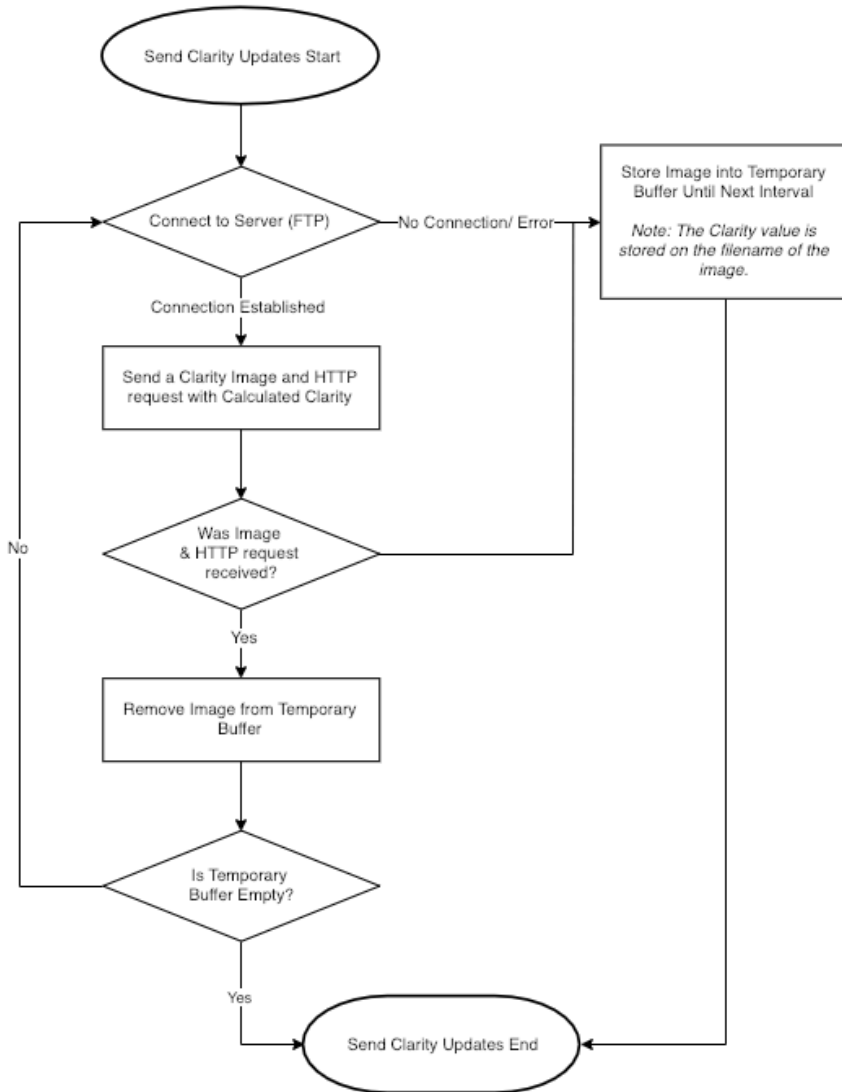


Figure 2.6 Flowchart for the Update Function – Send Clarity Update

2.2.5 Camera Function

The camera function serves two purposes: to monitor motion outside the TUS, and at the request of the update function to take a snapshot of the sky to determine its clarity to send to the CMS via the SCU function. The camera function implements an open source computer vision library called *OpenCV* to detect motion and determine the *CotS* [7]. *CotS* is used in data analysis to determine the relationship between clarity and the amount of solar energy the solar panel collects. *CotS* is also used to determine the relationship of clarity and the temperature readings the ESSI sensors collect. Motion detection is a security feature that triggers a motion flag when

the camera function detects motion. This flag causes the camera function to execute a separate function, *Motion Capture (MC)*. MC collects the image associated with the motion flag and collects a certain number of images before and after the flag to store into a local buffer until SMU accesses and sends the images to the CMS. Whether MC is running or not, the camera function continues to monitor for motion. Figures 2.7 and 2.8 show the flowcharts for the camera function.

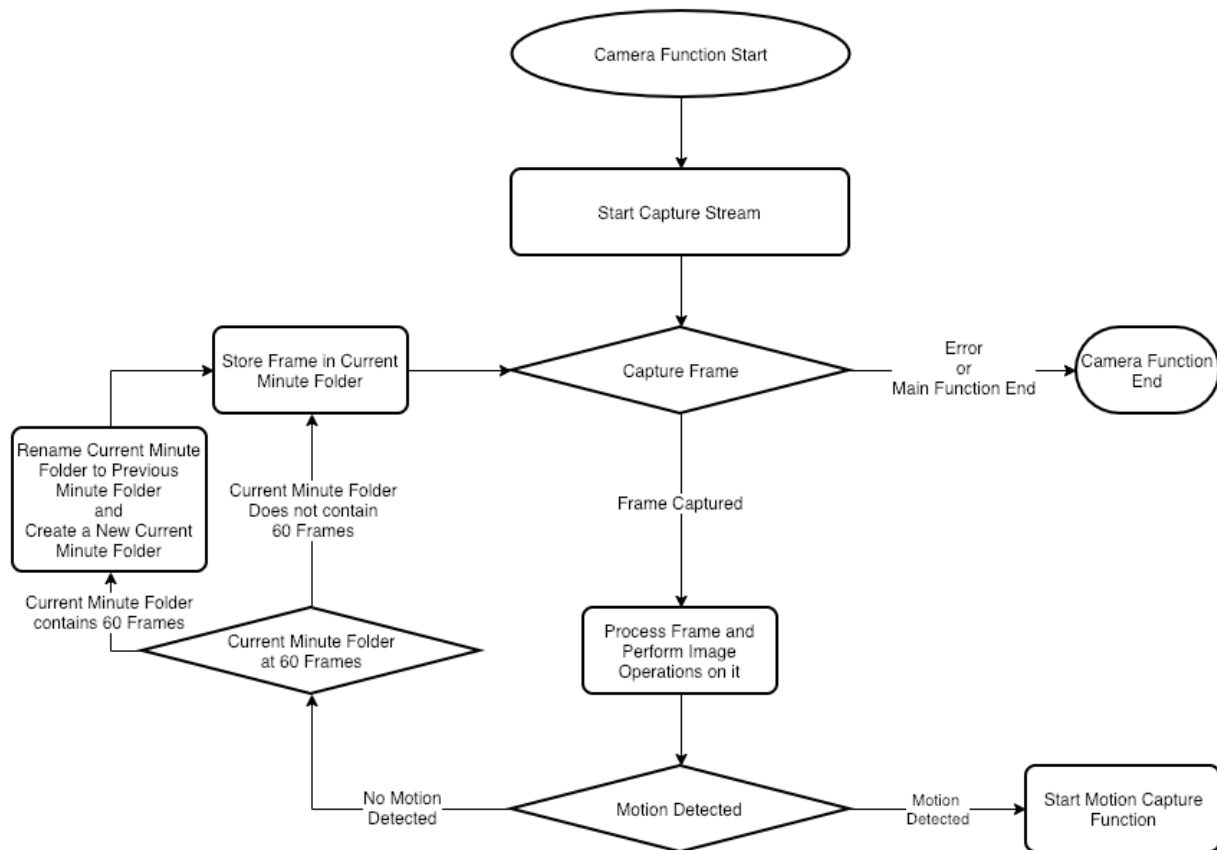


Figure 2.7 Flowchart for the Camera Function - Monitoring

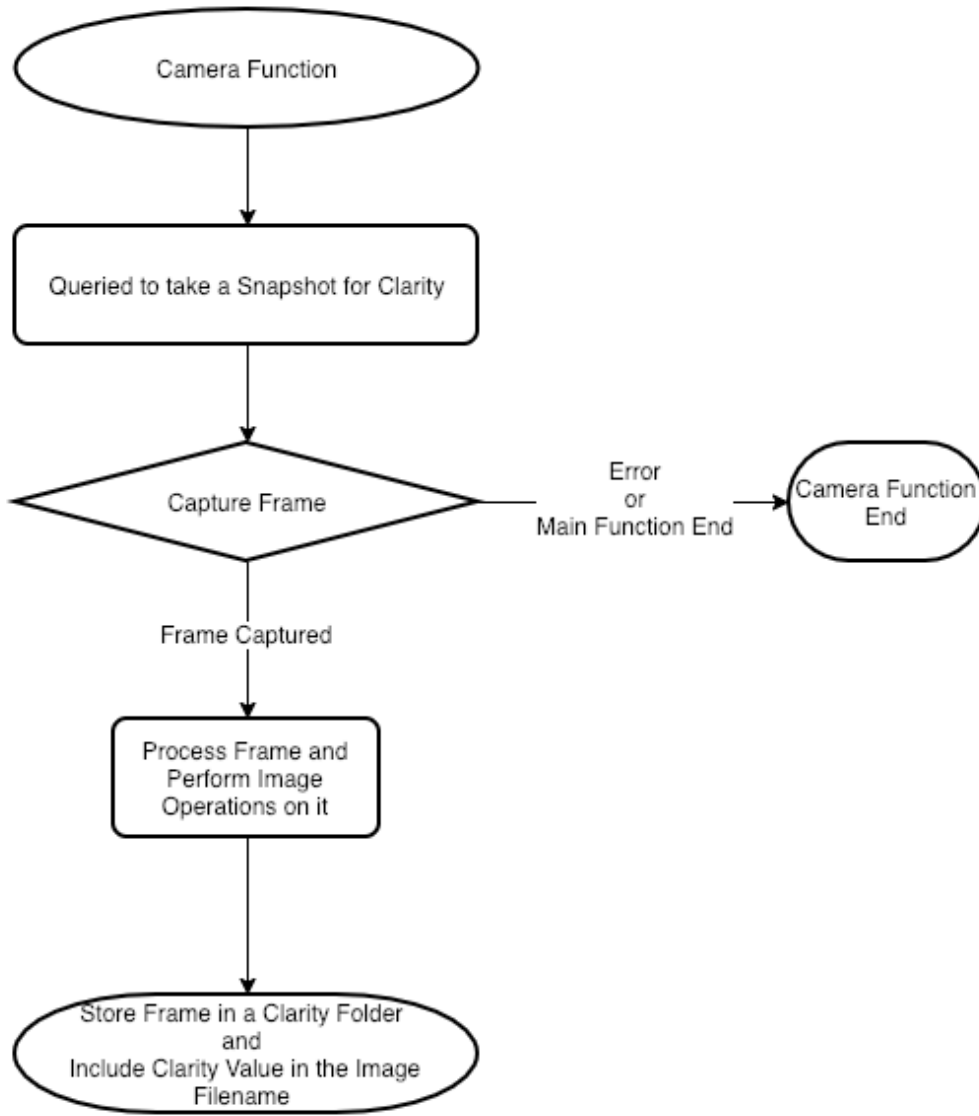


Figure 2.8 Flowchart for the Camera Function - Clarity

2.2.6 Camera Function – Motion Capture

When the camera function detects motion, MC collects the image associated with the motion flag and collects a certain number of images before and after the flag. MC takes the images and stores them into a buffer that the SMU function accesses to send to the CMS to store the images, log the incident and report the incident to the owner of the TUS. This ensures that there is plenty of context and room to identify a potential intruder. Finally, when MC collects the

last image it stores all the images into a local buffer and ends. Figure 2.9 shows the flowchart for the MC function.

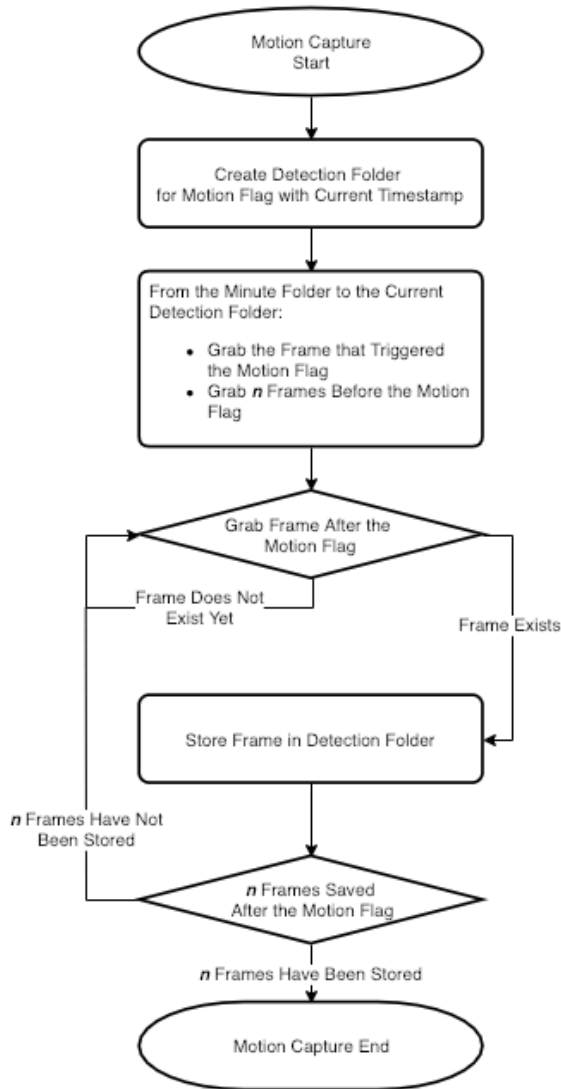


Figure 2.9 Flowchart for the Motion Capture Function

2.2.7 Camera Function – Clarity

When the update interval starts, a separate thread, SCU, calls for an image of the sky. Using OpenCV, the image of the sky is binarized to split the sky with any other objects on the image such as clouds or buildings. After binarization, clarity is calculated by the amount of white pixels in the binarized image compared to black pixels. Objects, such as clouds and buildings,

usually appear darker than the sky in images, as such the threshold values to segment the sky and the objects during binarization are adjusted accordingly. As a result, the pixels representing those objects are likely to become black during binarization. In addition to calculating clarity, the image itself is sent to the CMS with a naming convention of:

“RPiID_clarity_[timestamp]_[clarityval]”

where ‘RPiID’ is the ID of the RPi, ‘clarity’ is the purpose of the image, ‘timestamp’ is the timestamp of when the image was taken, and ‘clarityval’ is the calculated clarity. This is done such that if the camera is blocked from viewing the sky, there exists a context for the resulting inaccurate clarity readings. The CMS uses string operations to extract the information from the image filename followed by updating the database.

There may be cases where if the sky is completely cloudy or completely clear that thresholding cannot be done. In cases such as this another context needs to be considered such as determining clarity based on color using *Hue, Saturation, Values (HSV)*.

Figure 2.8 shows a flowchart for determining clarity and figure 2.10 shows an example of an image of the sky before and after binarization.





Figure 2.10 Clarity Image Before and After Binarization

2.3 ESSI Sensors

To find relationships among different factors impacting different aspects of sustainability of a TUS, one has to collect data and analyze it. The collected data is for current research and, also, for extending this research in the future. Data values such as latitude, time of the year, time of day, temperature, humidity, clarity, charging current and ESU levels are important regarding solar energy being collected. For practical reasons, the solar panel is assumed to be pointing upwards whether TUS is stationary or in motion. Therefore, data on the angle of the solar panel is not collected.

Other important ESSI values collected are for understanding the efficiency and operational aspects of a TUS by collecting and analyzing data values such as inside temperature of the TUS, current drain from the ESU, and *Cubic Feet per Minute (CFM)* of air being moved.

Below is a summary of data collected to help analyze different aspects of the system:

- GPS – Coordinates of the location of a TUS
- TS – Time Stamp
- OT – Outside TUS temperature
- IT – Inside TUS temperature
- OH – Outside Humidity

- IH – Inside Humidity
- Clarity – 0 for total overcast to 10 for sunny
- SC – Solar panel charging current
- SV – Solar panel charging voltage
- CC – Charging Current (charge controller to ESU)
- LV – ESU voltage level
- LC – Load Current from the ESU
- CFM – Volume of air moved from inside TUS to outside
- MOT – Motion detected in proximity of TUS
- IMG – Camera image frame

2.4 ESSO Devices

ESSO devices help maintain a specified state of TUS. The following devices are at system's disposal to utilize to maintain a specified state:

- EX – Exhaust to move air from inside of TUS to outside
- CAM – Motion detection and Clarity collection

2.5 Interfacing ESSI Sensors and ESSO Devices

Interfacing circuitry is needed to make the ESSI sensors and ESSO devices communicate with the RPi of a TUS. The sections below explain how each ESSI sensor and ESSO device interface with the RPi. Figure 2.11 represents the overall interfacing between ESSI sensors, ESSO devices, and RPi.

2.5.3 Outside and Inside TUS Temperature, Humidity (OT, IT, OH, IH)

Two digital temperature and humidity sensors are connected to the RPi's GPIO on Pin 23 and Pin 24.

2.5.4 Solar Panel System (SC, SV, CC)

The solar panel is mounted onto the exterior of the TUS and is connected to a charge controller, whereas the charge controller is connected to the ESU to charge it.

For the RPi to read the *Solar panel Voltage (SV)*, the solar panel's positive is run through a *Series Resistor Network (SRN)*, to lower the incoming voltage to the RPi's ADC range. This SRN connects to the ADC's Channel 2. For the RPi to read *Solar panel Current (SC)* and *Charge controller Current (CC)*, each value is read from its own current sensor on Channel 4 and Channel 5, respectively, of the ADC.

2.5.5 Energy System Unit Parameters (LC and LV)

A current sensor is connected in series with a fuse to read the ESU's *Load Current (LC)* while the ESU's positive is run through an SRN to read its *Load Voltage (LV)*. The current sensor and SRN connect to the RPi's ADC on Channel 1 and Channel 0, respectively.

2.5.6 Environmental System of the TUS

The environmental system of the TUS exhausts hot air inside the TUS and brings cooler air from the outside. It is enabled/disabled by the RPi via signals sent out of GPIO Pin 4. This signal goes to a relay that closes a switch to power on EX.

2.5.7 Camera (CAM) System Parameters: Clarity, Motion (MOT), and Image (IMG)

A camera is installed into the TUS and is connected to the RPi via the on-board camera interface.

2.6 Reading ESSI Sensors

2.6.1 GPS module

Python code is used to calibrate and read coordinates from the connected GPS module serially. The GPS module uses the *National Marine Electronics Association (NMEA)* protocol [2], thus an extra function is required to convert the latitude and longitude values to a format that most map applications use. The following code and formulas calculate the latitude and longitude:

```
# Latitude
if latGPS is not North then latGPS=-latGPS
latDeg = int(latGPS/100)
latMin = latGPS - latDeg*100
latAct = latDeg + (latMin/60)

# Longitude
if lonGPS is not East then lonGPS=-lonGPS
lonDeg = int(lonGPS/100)
lonMin = lonGPS - lonDeg*100
lonAct = lonDeg + (lonMin/60)
```

Variables North, South, East, and West are part of the data that is given by the GPS module, so when given the direction, the sign of the value is changed.

2.6.2 Time Stamp

Time stamp uses the '*Year-Month-Day Hour-Minute-Second*' (YMD HMS) format for ordered and non-duplicated time stamps. The time zone that the time stamp uses is the *Central Time (CT)* time zone, but is converted to a user's local time when accessing the website. Time stamp is read internally from RPi using the following Python code:

```
dateAndTimeFormat = "%Y-%m-%d %H:%M:%S"
timestamp = datetime.now(timezone("America/Chicago")).strftime(dateAndTimeFormat)
```

where

- dateAndTimeFormat - is the (YMD HMS) format,
- datetime.now - gives the current time using the time zone passed as an argument, and
- strftime - formats the resulting time from datetime.now to the (YMD HMS) format.

2.6.3 Outside and Inside TUS Temperature, Humidity (OT, IT, OH, IH)

The temperature/humidity sensors for reading inside and outside temperature/humidity of the TUS are connected to the RPi's GPIO via Pin 23 and Pin 24, respectively. If the sensors return a reading that is above/below/within the specified thresholds, the RPi sends a signal to enable/disable the EX. To read from the sensors, a Python library from Adafruit Industries is used [1]. The following Python code snippet reads from the sensors:

```
# Sets sensor pins & reads from GPIO
DHT11_I = 23
DHT11_O = 24
humidity_inner, temperature_inner = Adafruit_DHT.read(DHT11_SENSOR, DHT11_I)
humidity_outer, temperature_outer = Adafruit_DHT.read(DHT11_SENSOR, DHT11_O)
```

where

- DHT11_I=23 – is inside temperature and humidity sensor and its assigned GPIO pin #,
- DHT11_O=24 – outside temperature and humidity sensor and its assigned GPIO pin #, and
- Adafruit_DHT.read() – the function from the Adafruit library that takes in the type of sensor and the GPIO pin #. It returns the temperature and humidity values recorded by the sensors.

Reading values from these digital sensors sometimes exhibit spikes in temperature and humidity readings. For example, humidity readings have been seen as high as 174, even though the expected range is 0% to 100%. The code can replace such instances with the previous valid value because humidity and temperature are not supposed to change abruptly in the assumed environment.

Furthermore, sometimes there are no readings available from the sensors. In such cases also, the previous valid reading is assumed since temperature/humidity do not change abruptly. This is expected based on the statement of Adafruit's Tony DiCola in the sensor library's code: "Note that sometimes you won't get a reading and the results will be null because Linux can't guarantee the timing of calls to read the sensor" [1].

2.6.4 Solar Panel System (SC, SV, CC)

The RPi reads SV from ADC Channel 2, however, as the solar panel is rated to give from 12VDC up to a maximum voltage of 24VDC the solar panel goes through an SRN to lower the voltage to a value within the full voltage range of the ADC [6]. The following equation

$$(BV / 1023) * V_{Sup} / (R2 / (R1 + R2)) \quad (\text{Eq 2.1})$$

returns the original voltage value of the solar panel after its voltage has been reduced. The following is an explanation of the variables in the equation:

- BV - is the binary value read by the channel
- 1023 - is the maximum binary value of the channel
- V_{Sup} – Voltage that the RPi supplies to the ADC
- R1 – Resistor 1 of the SRN
- R2 – Resistor 2 of the SRN

To read solar panel current and charge controller current, SC and CC, the RPi reads voltage from current sensors connected to Channel 4 and Channel 5, respectively, of the ADC. When the RPi reads these voltage values, it uses equation 2.2 to convert the values to the amount of current flowing:

$$(BV / 1023) * (V_{Sup} / G) * (A_{Max}/V_{Max}) \quad (\text{Eq 2.2})$$

where

- BV - binary value read by the channel,
- 1023 - maximum binary value of the channel,
- V_{Sup} – voltage that the RPi supplies to the ADC,
- G – on-inverting gain, and
- A_{Max}/V_{Max} - rating of the ammeter.

In the proposed system, the current sensor has a rating of 10A/75mV, as a result the voltage values the ADC reads are too small to achieve an accurate reading. Therefore, before

connecting to an ADC channel, the signal goes through an *operational amplifier (Op-Amp)* to amplify the voltage. In order to determine how much the voltage needs to be amplified, a gain is calculated. To calculate the appropriate gain, two resistors are selected such that given Equation 2.3 the resulting gain multiplied by the voltage yields a larger value while remaining within range of the ADC. The following equation determines the non-inverting gain:

$$G = 1 + \frac{R_F}{R_2} \quad (\text{Eq 2.3})$$

where R_F is the feedback resistor and R_2 is a resistor in series. Figure 2.11 shows the circuit design in reading SC, SV, and CC.

2.6.5 Energy System Unit Parameters (LC and LV)

Reading LC and LV goes through the same process as the Solar Panel System as seen in Figure 2.12 which also includes the LC and LV.

The RPi reads LV from ADC Channel 1, however, as the ESU used in the proposed system is rated to give more than 12VDC, the voltage is reduced via an SRN to be within the range of the ADC [6]. Using Equation 2.1, the original voltage value is retrieved.

To measure LC, voltage is read from the respective current sensor on Channel 3. Since the current sensor is capable of 10A/75mV, the voltage from the sensor is very low for normal current values. Therefore, a non-inverting op-amp is used to amplify the voltage read from the current sensor. Similar to SC and CC, Equation 2.2 is used to calculate the actual current flowing through the sensor.

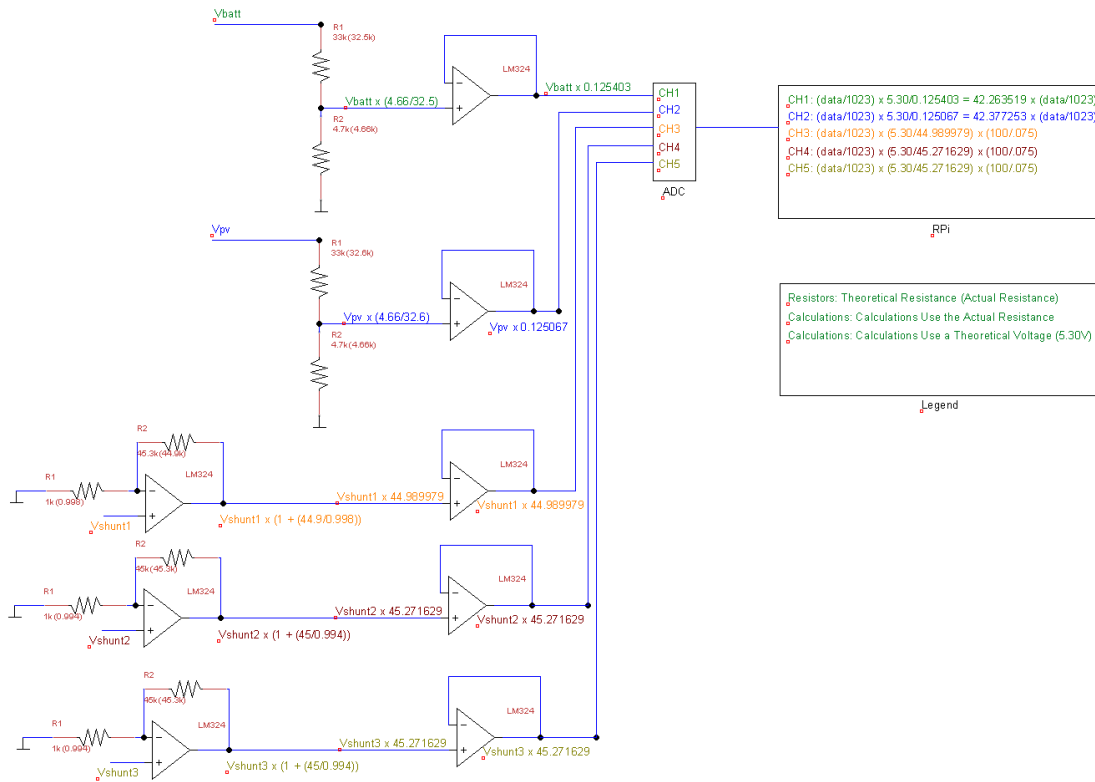


Figure 2.12 Circuit Design of SRNs, Voltage Followers and OpAmps for SV, SC, CC, LC, & LV

2.6.6 Environmental System of the TUS (CFM)

The CFM rating of the EX determines the value of CFM of air exhausted from inside to outside pulling fresh air from outside.

2.6.7 Camera System Parameters: Clarity, Motion (MOT), and Image (IMG)

The camera is monitoring the front scene of TUS for motion and a scheduled call by the update function to capture an image of the sky.

Motion detection is on a minute-directory system where frames pertaining to different minutes are stored. If motion is detected, images are captured based on the current image when the intrusion happens and 4 frames before/after the intrusion. Refer to ‘motion_detection.py’ in Appendix B. The full code for the camera system is available on GitHub [3].

2.7 TUS Local State Buffer and Populating JSON

The state of ESSI sensors and ESSO devices is sampled at a defined sampling rate and stored locally as a JSON file. In the proposed system, the sampling rate is defined at one minute. If network connectivity to the CMS is available, a Python update function sends the JSON file to the CMS via FTP and queries the CMS if the file is received via HTTP request. This repeats until all files in the local buffer have been sent. The full code for each update function and their FTP and HTTP transactions can be viewed in Appendix B. The full code is available on GitHub [3].

The CMS knows the identity of a TUS by its ID that is denoted by 'rpid' in Figure 2.13. This is important since the CMS handles multiple TUS instances. Figure 2.13 also shows the format of the JSON file that the RPi sends to the CMS.

```
{
  "rpid": "xx",
  "log": "YYYY-MM-DD HH:MM:SS",
  "batteryvoltage": "xx.xx",
  "batterycurrent": "xx.xx",
  "solarpanelvoltage": " xx.xx ",
  "solarpanelcurrent": " xx.xx ",
  "chargecontrollercurrent": " xx.xx ",
  "temperatureinner": "xx",
  "temperatureouter": "xx",
  "humidityinner": "xx",
  "humidityouter": "xx",
  "gps": ["latitude', 'longitude']"
}
```

Figure 2.13 Format of JSON file

2.8 CMS Processing of JSON

The RPi sends an HTTP request to the CMS requesting the status of the JSON file after it sends it via FTP. If the JSON file is on the CMS, a PHP script notifies the RPi of the successful transaction with a status code. The PHP script also parses the JSON file where SQL queries embedded in the script takes the parsed data to populate the database. If the JSON file is not on

the CMS or an error occurs during the processing of the JSON file, the PHP script returns an appropriate status code to the RPi and contacts the administrator via email about the error. The status codes are:

- 1) OK
- 2) NO
- 3) ERROR

where 'OK' is all files/updates are successfully received/processed, 'NO' is all files are not received, and 'ERROR' indicates an error during processing. The RPi continues with its normal functions given an 'OK', while the RPi stores the files in its buffers given a 'NO' or 'ERROR'.

2.9 Website

The website, as mentioned in the introduction, is the main hub for the user to interact with and view updates from the respective RPi. It provides the user with a view of the current status of the TUS (Figure 2.15), the ability to change the thresholds of the vitals (Figure 2.16), the ability to view and download logs (Figure 2.17) and statistics (Figure 2.18), and view images taken by the camera (Figure 2.19). The website's backend is written in PHP as all data/information is stored on a database. The website's frontend is written in HTML whereas the interaction and form processing are written in JavaScript. For data analysis purposes, the user can download *Comma-Separated Values (CSV)* formatted files from the statistics page according to the user's selection of data, timestamp, and time interval.

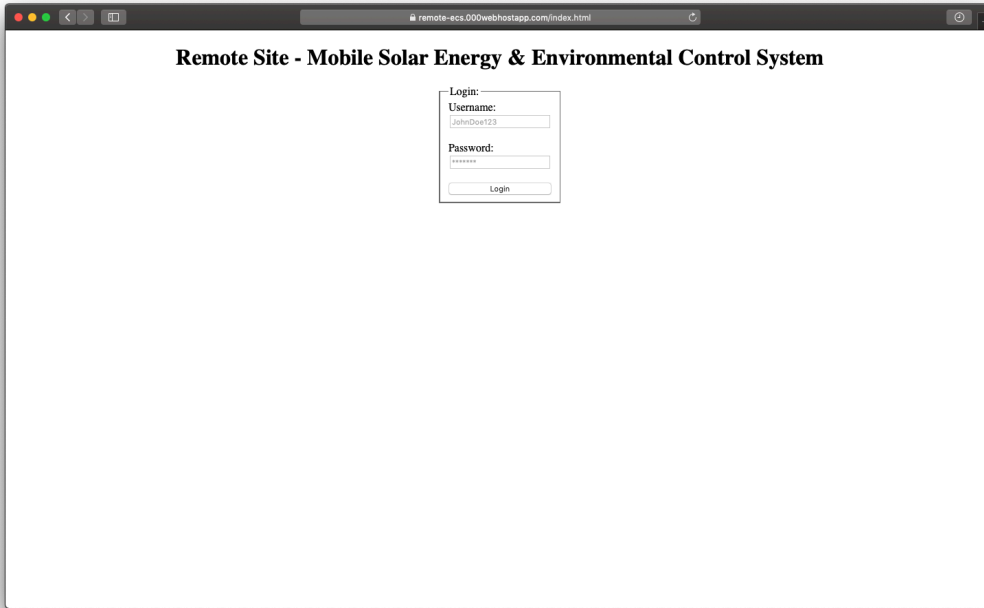


Figure 2.14 Website's Login/Registration Page

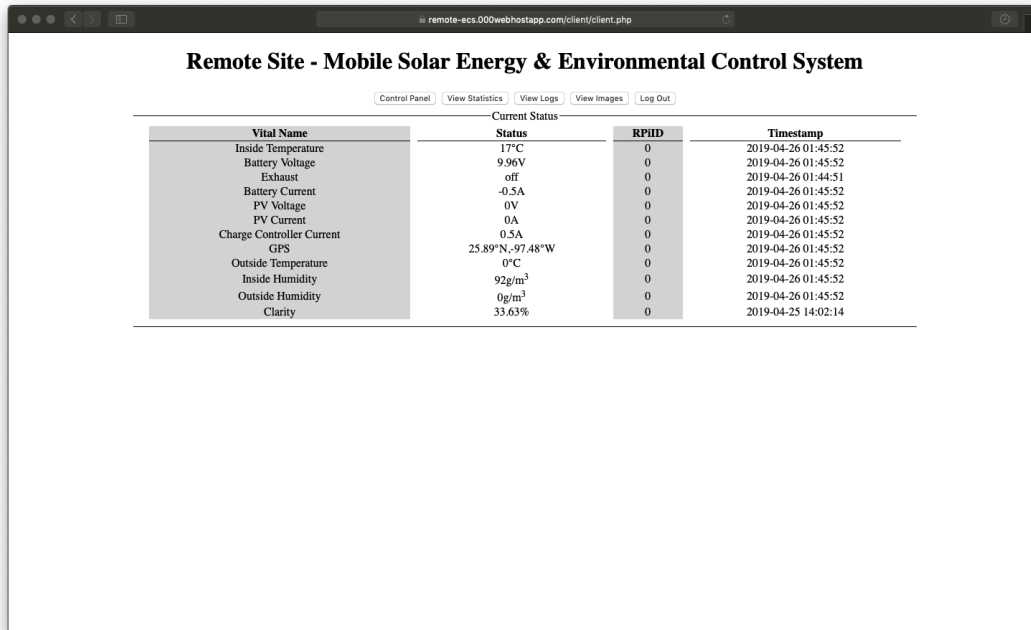


Figure 2.15 Website's Main User Page

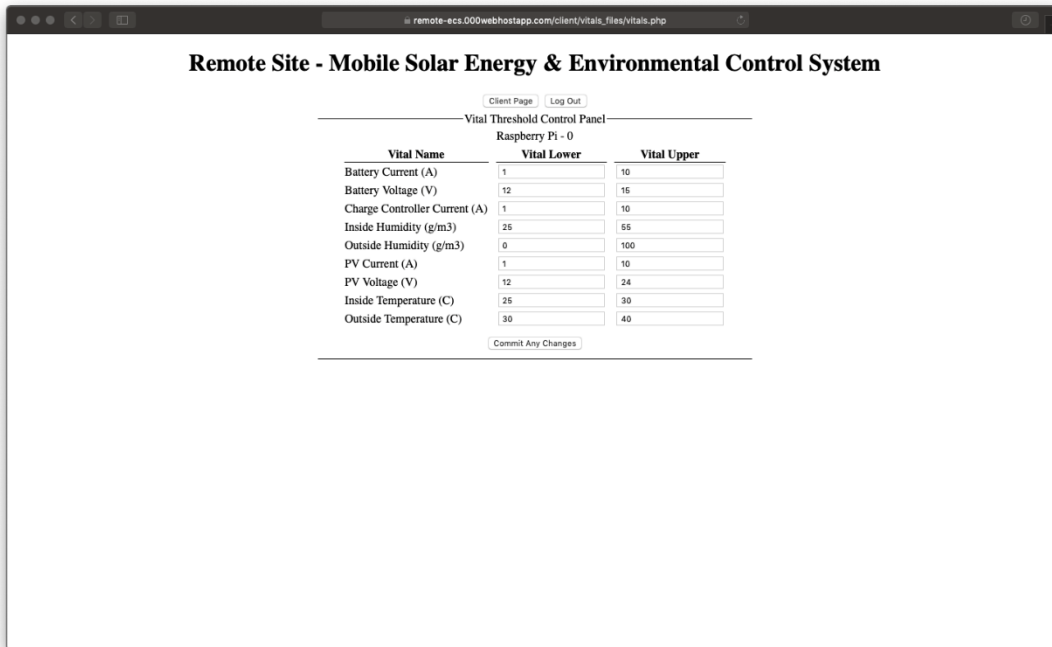


Figure 2.16 Website's Control Panel Page

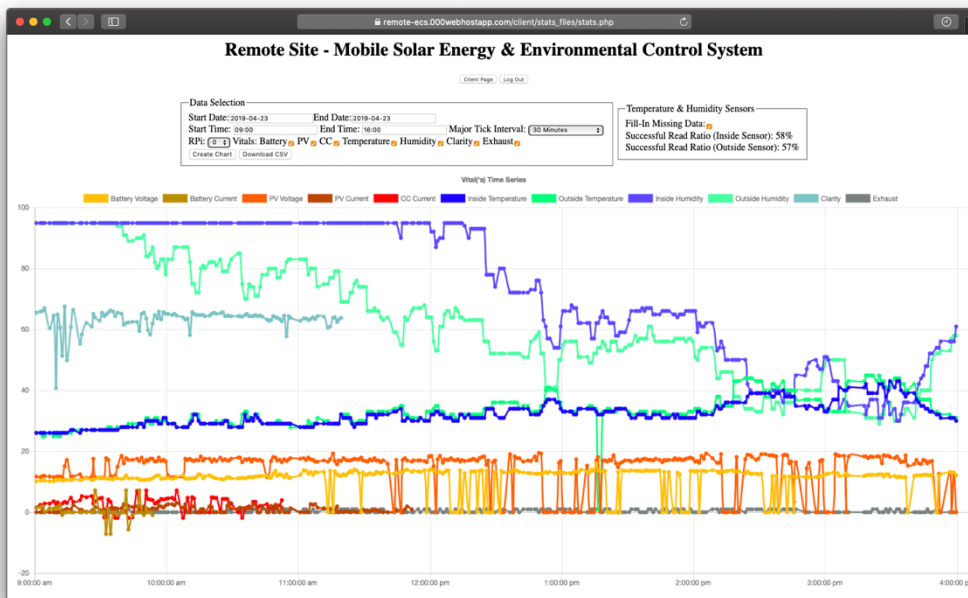


Figure 2.17 Website's Statistics View and Download Page

remote-ecs.000webhostapp.com/client/log_files/listlogs.php

Remote Site - Mobile Solar Energy & Environmental Control System

Client Page | Log Out

Log

Vital Name	TYP	RPIID	Vital Value	Timestamp
Battery Current	ST	0	-0.47A	2019-04-26 01:42:51
Inside Humidity	ST	0	93g/m ³	2019-04-26 01:42:51
Charge Controller Current	ST	0	0.47A	2019-04-26 01:42:51
Outside Temperature	ST	0	0°C	2019-04-26 01:42:51
Outside Humidity	ST	0	0g/m ³	2019-04-26 01:42:51
PV Current	ST	0	0A	2019-04-26 01:42:51
Battery Voltage	ST	0	10.04V	2019-04-26 01:42:51
GPS	ST	0	25.89°N,-97.48°W	2019-04-26 01:42:51
Inside Temperature	ST	0	18°C	2019-04-26 01:42:51
PV Voltage	ST	0	0V	2019-04-26 01:42:51
Clarity	ST	0	0%	2019-04-26 01:42:48
Battery Current	ST	0	-0.71A	2019-04-26 01:41:52
Inside Humidity	ST	0	93g/m ³	2019-04-26 01:41:52
Charge Controller Current	ST	0	0.71A	2019-04-26 01:41:52
Outside Temperature	ST	0	17°C	2019-04-26 01:41:52
Outside Humidity	ST	0	92g/m ³	2019-04-26 01:41:52
PV Current	ST	0	0A	2019-04-26 01:41:52
Battery Voltage	ST	0	10.03V	2019-04-26 01:41:52
GPS	ST	0	25.89°N,-97.48°W	2019-04-26 01:41:52
Inside Temperature	ST	0	18°C	2019-04-26 01:41:52
PV Voltage	ST	0	0V	2019-04-26 01:41:52
Clarity	ST	0	0%	2019-04-26 01:41:48
Charge Controller Current	ST	0	0.6A	2019-04-26 01:41:05
Outside Temperature	ST	0	16°C	2019-04-26 01:41:05
Outside Humidity	ST	0	92g/m ³	2019-04-26 01:41:05
PV Current	ST	0	0A	2019-04-26 01:41:05
Battery Voltage	ST	0	9.99V	2019-04-26 01:41:05
GPS	ST	0	25.89°N,-97.48°W	2019-04-26 01:41:05
Inside Temperature	ST	0	18°C	2019-04-26 01:41:05
PV Voltage	ST	0	0V	2019-04-26 01:41:05
Battery Current	ST	0	-0.6A	2019-04-26 01:41:05
Inside Humidity	ST	0	93g/m ³	2019-04-26 01:41:05
Clarity	ST	0	0%	2019-04-26 01:40:48

Figure 2.18 Website's Log View Page

remote-ecs.000webhostapp.com/client/image_files/image.php

Image Name	Click to View
0_capture_[2019-04-22 02-08-10].jpg	Click Here!
0_capture_[2019-04-22 02-08-11].jpg	Click Here!
0_capture_[2019-04-22 02-08-12].jpg	Click Here!
0_capture_[2019-04-22 02-08-13].jpg	Click Here!
0_capture_[2019-04-22 02-08-14].jpg	Click Here!
0_capture_[2019-04-22 02-08-15].jpg	Click Here!
0_capture_[2019-04-22 02-08-16].jpg	Click Here!
0_capture_[2019-04-22 02-08-17].jpg	Click Here!
0 [2019-04-22 02-08-19]	
0_capture_[2019-04-22 02-08-19].jpg	Click Here!
0_capture_[2019-04-22 02-08-20].jpg	Click Here!
0_capture_[2019-04-22 02-08-21].jpg	Click Here!
0_capture_[2019-04-22 02-08-22].jpg	Click Here!
0_capture_[2019-04-22 02-08-23].jpg	Click Here!
0 [2019-04-22 02-08-24]	
0_capture_[2019-04-22 02-08-21].jpg	Click Here!
0_capture_[2019-04-22 02-08-22].jpg	Click Here!
0_capture_[2019-04-22 02-08-23].jpg	Click Here!
0_capture_[2019-04-22 02-08-24].jpg	Click Here!
0_capture_[2019-04-22 02-08-25].jpg	Click Here!
0_capture_[2019-04-22 02-08-26].jpg	Click Here!
0_capture_[2019-04-22 02-08-27].jpg	Click Here!
0 [2019-04-22 02-16-20]	
0_capture_[2019-04-22 02-16-19].jpg	Click Here!
0_capture_[2019-04-22 02-16-20].jpg	Click Here!
0_capture_[2019-04-22 02-16-21].jpg	Click Here!
0_capture_[2019-04-22 02-16-22].jpg	Click Here!
0_capture_[2019-04-22 02-16-23].jpg	Click Here!
0_capture_[2019-04-22 02-16-24].jpg	Click Here!
0 [2019-04-22 02-16-25]	
0_capture_[2019-04-22 02-16-22].jpg	Click Here!
0_capture_[2019-04-22 02-16-23].jpg	Click Here!
0_capture_[2019-04-22 02-16-24].jpg	Click Here!

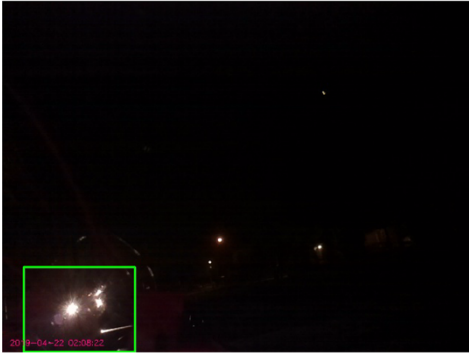


Figure 2.19 Website's Image View Page

CHAPTER III

RESULTS

The goal of the research was to implement a sustainable ECS using a mobile and IoT approach. The following results are explained in the following sections.

3.1 Mobile and Internet of Things Implementation

Using the RPi's wireless connectivity, whenever the RPi is in the range of a known network, it automatically connects to the network and starts updating or retrieving updates to/from the CMS. This allows an individual to be anywhere in the world and still manage the respective RPi and TUS. An individual can access the data locally by extracting the content intended for the CMS from the RPi, if it has not been sent to the CMS yet. Using a website as the online interface to an individual's RPi and TUS, an individual can view the status of the respective TUS and alter the associated thresholds to operate ESSO devices. As a result, this research developed and implemented a platform in which an individual can control and sustain an ECS within a TUS locally or globally.

3.2 Data Collection and Data Visualization

This research also developed and implemented a platform which an individual can download to use and visualize the data of RPi(s). Through the use of a time series line chart, as seen on Figure 2.16 on the website, the user can view the data the respective RPi has collected given data parameters of start date, end date, start time, end time, and time interval. The individual can also select to view the data of a specific RPi or specific ESSI or ESSO value.

From these same settings an individual can also download a CSV formatted file for use in data analysis. Using the data for data analysis can further improve upon the controllability and sustainability of the ECS.

3.3 Time and Space Complexity

After some initial testing of the program and system over the time span of a week, data from every minute has been collected. From this data it is possible to determine the potential space the program uses if there is no network connectivity resulting in the temporary storing of motion detection images, temporary storing of clarity images, and temporary storing of JSON files.

To provide some background, the CMS executes a weekly script which checks if all RPi's have contacted the server, if not then the owner of the RPi is notified. If an RPi has not contacted the server then it may indicate that the RPi is failing to connect to a network. Thus, this space complexity calculation focuses on the total size for a week assuming the RPi cannot access a network for a week and detects constant motion.

In a week, the total size of all JSON files that have been created is around *2.5Megabytes (MB)* and the total size of all Clarity images that have been taken is around 100MB. The images from motion detection take significantly more storage, however, statistically speaking the motion is not expected constantly.

In motion detection, there are two main factors, no motion and constant motion. No motion is a best-case scenario where no motion is detected thus no images are being stored resulting in zero disk space used. Constant motion is a worst-case scenario where motion is constantly detected thus images are constantly being stored.

A few things to note:

- In the proposed system, the images taken by the RPi have a fixed resolution of 640x480.
- The images taken by the RPi vary in size depending on the image taken. For example, if there are many objects of different colors in the image, the size of the image increases, otherwise,

the size of the image remains small. So, this calculation assumes the worst-case of an image of size 100Kilobytes (KB).

- As previously described, the MC function captures images before and after motion is detected and captures the image when motion is detected. So, to prevent duplicates only one instance of MC can run.
- In the proposed system, MC collects four images before and after and one image at the instance motion is detected, thereby, collecting nine images per motion detection instance. This can be repeated every 4th second for constant motion, thereby, generating fifteen instances of nine images.

Given the above, the total size in a minute is calculated as follows:

$$15 \times 9 \times image_size \quad (eq\ 2.4)$$

Using the image size as 100KB, as mentioned before, the disk space needed for constant motion in a minute is:

$$15 \times 9 \times 100\ KB = 13.500MB$$

For one minute of constant motion detection. The next calculation is to determine the total size in a day, the calculation is as follows:

$$13.5MB \times 24 \times 60 = 19.44GB \quad (eq\ 2.5)$$

In total, for a day of constant motion 19.44 Gigabytes (GB) of storage is needed, which translates to 136.08 GB per week. Storage of that size is not reasonable for this application thus this presents an issue should the worst-case occur. Two solutions can be considered. Firstly, one can limit the camera capturing to one instance per minute needing 9.072 GB. Secondly, if the motion capture directory reaches a specific size threshold, the oldest images will be deleted from the disk. While both may ease the problem of storage, they are disruptive to the security of the system.

Another type of complexity to consider is time complexity. Given the large quantity of data collected as a result of collecting data every minute, the retrieval of data and creation of data visualization does take time. Thus, it is important to know the speed at which data can be

retrieved from the database as discussed earlier in Chapter I (Introduction). Also, generating the data visualization on the website takes time. The retrieval of data for one day from the database takes about one second. The visualization generation for one day worth of data takes 50 milliseconds.

CHAPTER IV

CONCLUSION, CONTRIBUTIONS AND FUTURE WORK

4.1 Contributions

4.1.1 Main Contributions

This research proposed and implemented an IoT system for integrating mobile sustainable environments locally and globally. This research developed and implemented a platform in which an individual can manage and monitor these environments, as well as visualize and download data via a website. In addition, this research developed a framework for a mobile application that acts as a second mode of a user interface to manage and monitor environments, visualize and download data. In doing so, the individual can be anywhere in the world and have remote accessibility to the respective TUS. Downloaded data can be used in data analysis to analyze ways to improve the controllability and sustainability of the environments.

4.1.2 Incidental Contributions

Incidentally, this research provides a platform for individuals to maintain a safe and secure environment using temperature control and camera monitoring. In addition, the proposed system implements a notification system for an individual to be notified in case thresholds have been breached or an intrusion is occurring. As a result, an individual is always kept up to date regarding the respective TUS. As a real-world example using temperature control, the proposed system can help prevent the death or severe injury of a person or animal left inside a hot vehicle without air conditioning. Another real-world example using camera monitoring, an individual

can have video evidence of someone breaking into a vehicle. If the vehicle is stolen and the RPi is connected to a network, the images before and after the fact are going to be uploaded to the CMS and can continue to do so until the RPi can no longer access a network.

4.2 Conclusion and Future Work

In conclusion, this research developed and implemented a sustainable ECS that is accessible locally and globally, while providing an individual with the ability to control the environment and collect related data.

When considering space complexity worst-case scenarios as discussed earlier, a possible solution is to provide the RPi with an adapter that can allow it to connect to a mobile network such as 3G in order to provide network connectivity if WiFi is not available. Another issue kept occurring with the CMS where the database would not insert or update values into the database despite the CMS successfully receiving these values. In order to get these missing values into the database, one must create a script to read through the JSON files and update the database manually. As this will fix the issue, it is not a permanent solution, thus a future improvement would be to find a more reliable hosting platform.

Lastly, the platform that this research developed and implemented still has a room to develop. Future work may consider an embedded systems approach. By integrating more RPi units and TUS into the system, new options in the computer science fields, such as Big Data and Machine Learning, present themselves due to the large quantities of data that can be collected from this system.

REFERENCES

- [1] DiCola, T. (2018). Adafruit_Python_DHT. Retrieved from https://github.com/adafruit/Adafruit_Python_DHT/
- [2] Baddeley, G. (2001). GPS - NMEA sentence information. Retrieved from <http://aprs.gids.nl/nmea/>
- [3] Garay, L. (2019). Mobile Solar Energy Environmental Control System. Retrieved from <https://github.com/lsg2550/Mobile-Solar-Energy-Environmental-Control-System>
- [4] Garay, L. (2019). Mobile Solar Energy Environmental Control System Website. Retrieved from <https://remote-ecs.000webhostapp.com/>
- [5] Cook, J. A., Kolmanovsky, I. V., Mcnamara, D., Nelson, E. C., & Prasad, K. V. (2007). Control, Computing and Communications: Technologies for the Twenty-First Century Model T. *Proceedings of the IEEE*, 95(2), 334-355. doi:10.1109/jproc.2006.888384
- [6] Microchip Technology Inc. (2007). 2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface. Retrieved from <http://ww1.microchip.com/downloads/en/devicedoc/21295c.pdf>
- [7] OpenCV Team. (2019). Open Source Computer Vision Library. Retrieved from <https://opencv.org/>
- [8] Python Software Foundation. (2019). threading - Thread-based parallelism. Retrieved from <https://docs.python.org/3.7/library/threading.html#module-threading>
- [9] Raspberry Pi Foundation. (2019). Raspberry Pi Documentation – Pi Power. Retrieved from <https://www.raspberrypi.org/documentation/faqs/#pi-power>
- [10] Raspberry Pi Specs and Benchmarks. (2019). Retrieved from <https://www.raspberrypi.org/magpi/raspberry-pi-specs-benchmarks/>

APPENDIX

APPENDIX A

THEORETICAL VS RECORDED VALUES TABLES

Table of Component Theoretical/Recorded Values

Variable	Theoretical Value	Recorded Value
Battery Voltage Divider Factor	$R2/(R1+R2) = 0.094$	0.092
Solar Panel Voltage Divider Factor	$R4/(R3+R4) = 0.094$	0.092
OpAmp Gain for measuring V1	$1 + R_1/R_{i1} = 46.3$	46.17
OpAmp Gain for measuring V2	$1 + R_2/R_{i2} = 46.3$	46.09
OpAmp Gain for measuring V3	$1 + R_3/R_{i3} = 46.3$	46.27

APPENDIX B

SOURCE CODE

B.1 GetAndSendStatus – This function retrieves the JSON file stored in the temporary buffer and sends it to the CMS. In case of an issue with the transaction, the JSON file remains in the buffer, otherwise, it is moved to a “sent” buffer for garbage.

```
try:
    for stored_file in sorted(os.listdir(TEMPORARY_STORAGE_DIRECTORY)):
        temporary_file = str(stored_file)

        if temporary_file.endswith(".json"):
            temp_storage_full_path = os.path.join(TEMPORARY_STORAGE_DIRECTORY, temporary_file)
            sent_storage_full_path = os.path.join(SENT_STORAGE_DIRECTORY, temporary_file)
            CTF.SendStatus(TEMPORARY_STORAGE_DIRECTORY + temporary_file)

            global_var.PIPAYLOAD["xmlfile"] = temporary_file
            server_confirmation = requests.get("https://remote-ecs.000webhostapp.com/index_files/piconfirm.php",
            params=global_var.PIPAYLOAD, timeout=5)
            print(server_confirmation.text.strip())
            global_var.PIPAYLOAD.pop("xmlfile")

            if server_confirmation.text.strip() == "OK":
                print("File confirmed received!")
                os.rename(temp_storage_full_path, sent_storage_full_path)
            else: break # Server did not receive or process the JSON correctly
        except Exception as e: print("Could not connect to server...\nStoring status file into {}...\nError
        Received: {}".format(TEMPORARY_STORAGE_DIRECTORY, e))

    print("Status background thread done!")
```

B.2 GetAndSendImages – This function retrieves the detected images and sends it to the CMS. Similar to GetAndSendStatus, if there is an issue with the connection, the images are stored until the next update interval, otherwise, the images are deleted from the disk.

```
try:
    detection_directory_contents = sorted(os.listdir(MD.DETECTION_DIRECTORY))
    for stored_frames in detection_directory_contents:
        if detection_directory_contents[-1] == stored_frames: break # Skip the last folder in case it is still being filled with images
        temporary_file_full_path = os.path.join(MD.DETECTION_DIRECTORY, stored_frames)

        for root, subfolders, files in sorted(os.walk(temporary_file_full_path)):
            CTF.SendImages(root, files)

            global_var.PIPAYLOAD["capture"] = stored_frames
            server_confirmation = requests.get("https://remote-ecs.000webhostapp.com/index_files/piimageconfirm.php",
            params=global_var.PIPAYLOAD, timeout=5)
            print(server_confirmation.text.strip())
            global_var.PIPAYLOAD.pop("capture")

            if server_confirmation.text.strip() == "OK":
                print("File and folders confirmed received!")
                shutil.rmtree(root) # Delete Capture Folder
            else: break # Server did not receive or process the images correctly
        except Exception as e: print("Could not connect to server...\nImages were not sent...\nError Received: {}".format(e))

    print("Images background thread done!")
```

B.3 GetAndSendImage – This function takes, processes, and sends an individual image to the CMS. The filename of the image consists of the timestamp and clarity value.

```
MD.QuickCapture() # Take a Quick Capture
try:
    for clarity_frame in sorted(os.listdir(MD.CLARITY_DIRECTORY)):
        temporary_file_full_path = os.path.join(MD.CLARITY_DIRECTORY, clarity_frame)
```

```

CTF.SendClarity(clarity_frame)

global_var.PIPAYLOAD["clarity"] = clarity_frame
server_confirmation = requests.get("https://remote-ecs.000webhostapp.com/index_files/piimageconfirm.php",
params=global_var.PIPAYLOAD, timeout=5)
print(server_confirmation.text.strip())
global_var.PIPAYLOAD.pop("clarity")

if server_confirmation.text.strip() == "OK":
    print("File and clarity status confirmed received!")
    os.remove(temporary_file_full_path) # Delete Clarity Image
else: break # Server did not receive or process the images correctly
except Exception as e: print("Could not connect to server...\nImage was not sent...\nError Received: {}".format(e))

print("Clarity Image background thread done!")

```

B.4 generate_send_xml – This is the main function that contacts the CMS for updates and sends updates to the CMS. This function also initializes and runs the other major functionalities of the program.

```

# Program Start Time
START_TIME = time.time() # Initialize Program Start Time
SEND_STATUS_THREAD = None # Thread for sending JSON
SEND_CLARITY_THREAD = None # Thread for sending single status/clarity image
SEND_IMAGES_THREAD = None # Thread for sending multiple motion detection images

# Start Camera Thread
CAMERA_THREAD = Process(target=MD.Main, args=(START_TIME, ), daemon=True)
CAMERA_THREAD.start()

try:
    while True:
        try: # Retrieve files with thresholds set by user
            print("Requesting threshold update from server...")
            server_threshold_confirmation = requests.get("https://remote-ecs.000webhostapp.com/index_files/pithresholdconfirm.php",
params=global_var.PIPAYLOAD, timeout=5)
            print(server_threshold_confirmation.text.strip())

            if server_threshold_confirmation.text.strip() == "OK":
                CTF.RetrieveThreshold(global_var.RPID)
                threshold_file_name = str(global_var.RPID) + ".json"

                # Tell the server that we retrieved the file
                global_var.PIPAYLOAD["result"] = "OK"
                requests.get("https://remote-ecs.000webhostapp.com/index_files/piserverconfirm.php", params=global_var.PIPAYLOAD,
timeout=5)
                global_var.PIPAYLOAD.pop("result")
            else: # Tell the server that we did not retrieve the file
                global_var.PIPAYLOAD["result"] = "NO"
                requests.get("https://remote-ecs.000webhostapp.com/index_files/piserverconfirm.php", params=global_var.PIPAYLOAD,
timeout=5)
                global_var.PIPAYLOAD.pop("result")
                raise FileNotFoundError
        except Exception as e:
            print("Could not connect to server/Issue with server...\nError Received: {}".format(e))
            if os.path.exists(str(global_var.RPID) + ".json"):
                threshold_file_name = str(global_var.RPID) + ".json"
                print("Using previous thresholds...")
            else:
                threshold_file_name = "default.json"
                print("Using system default thresholds...")

        with open(threshold_file_name, "r") as thresholdfile: thresholds = json.loads(thresholdfile.read())
        threshold_voltage_lower = thresholds["voltage_lower"]
        threshold_voltage_upper = thresholds["voltage_upper"]
        threshold_current_lower = thresholds["current_lower"]
        threshold_current_upper = thresholds["current_upper"]
        threshold_solar_panel_voltage_lower = thresholds["spvoltage_lower"]
        threshold_solar_panel_voltage_upper = thresholds["spvoltage_upper"]
        threshold_solar_panel_current_lower = thresholds["spcurrent_lower"]
        threshold_solar_panel_current_upper = thresholds["spcurrent_upper"]

```



```

threshold_charge_controller_current_lower = thresholds["ccurrentlower"]
threshold_charge_controller_current_upper = thresholds["ccurrentupper"]
threshold_temperature_inner_lower = thresholds["temperatureinnerlower"]
threshold_temperature_inner_upper = thresholds["temperatureinnerupper"]
threshold_temperature_outer_lower = thresholds["temperatureouterlower"]
threshold_temperature_outer_upper = thresholds["temperatureouterupper"]
threshold_humidity_inner_lower = thresholds["humidityinnerlower"]
threshold_humidity_inner_upper = thresholds["humidityinnerupper"]
threshold_humidity_outer_lower = thresholds["humidityouterlower"]
threshold_humidity_outer_upper = thresholds["humidityouterupper"]

# Read from Sensors
print("Reading from sensors...")
sensor_status_dictionary = RAFA.read_from_sensors(threshold_voltage_lower, threshold_voltage_upper,
threshold_current_lower, threshold_current_upper,
threshold_solar_panel_voltage_lower, threshold_solar_panel_voltage_upper,
threshold_solar_panel_current_lower, threshold_solar_panel_current_upper,
threshold_charge_controller_current_lower, threshold_charge_controller_current_upper,
threshold_temperature_inner_lower, threshold_temperature_inner_upper,
threshold_temperature_outer_lower, threshold_temperature_outer_upper,
threshold_humidity_inner_lower, threshold_humidity_inner_upper,
threshold_humidity_outer_lower, threshold_humidity_outer_upper)

print("Done reading from sensors...")

# Generate Timestamps
timestamp_for_log = datetime.now(timezone("America/Chicago")).strftime(DATE_AND_TIME_FORMAT)
timestamp_for_filename = timestamp_for_log.replace(":", "-")

# Create JSON
json_format = {"log": str(timestamp_for_log), "rpid": str(global_var.RPID)}
for key, value in sensor_status_dictionary.items(): json_format[key] = str(value)
json_file = TEMPORARY_STORAGE_DIRECTORY + "status" + str(global_var.RPID) + "(" + timestamp_for_filename + ").json"
with open(json_file, "w+") as status: json.dump(json_format, status, indent = 4)

# Send JSON in new thread
if SEND_STATUS_THREAD is None or not SEND_STATUS_THREAD.isAlive():
    SEND_STATUS_THREAD = Thread(target=get_and_send_status, args=())
    SEND_STATUS_THREAD.setDaemon(True)
    SEND_STATUS_THREAD.start()

# Send images in new thread
if SEND_IMAGES_THREAD is None or not SEND_IMAGES_THREAD.isAlive():
    SEND_IMAGES_THREAD = Thread(target=get_and_send_images, args=())
    SEND_IMAGES_THREAD.setDaemon(True)
    SEND_IMAGES_THREAD.start()

# Send a single image in new thread
if SEND_CLARITY_THREAD is None or not SEND_CLARITY_THREAD.isAlive():
    SEND_CLARITY_THREAD = Thread(target=get_and_send_clarity, args=())
    SEND_CLARITY_THREAD.setDaemon(True)
    SEND_CLARITY_THREAD.start()

# Wait for 60 seconds for the next read interval
timer = (time.time() - START_TIME) % 60
print("File transfers moved to a background thread...\nMain thread is now on standby for {0:.2} seconds...\n".format(str((60.0 - timer))))
time.sleep(60.0 - timer)
# while end
finally: CAMERA_THREAD.join()

```

B.5 CaptureIntrusion – This function is called when motion is detected near the TUS. This function first creates the detection folder for the time when motion is detected and stores this frame along with 4 frames before/after detection of intrusion into the folder. The content of this folder is sent to the CMS when the next update interval occurs.

```

print("Motion capture starting...")

# Create a detection directory for this instance of motion detected
detection_and_filename_path = DETECTION_DIRECTORY + str(global_var.RPID) + "_" + filenameSafeCurrentTime + "]"
if not os.path.isdir(detection_and_filename_path): os.mkdir(detection_and_filename_path)

# Load names of files in the previous/current directories

```

```

previous_minute_directory_list = sorted(os.listdir(PREVIOUS_MINUTE_DIRECTORY))
current_minute_directory_list = sorted(os.listdir(CURRENT_MINUTE_DIRECTORY))
current_frame_index = CURRENT_MINUTE_DIRECTORY.find(frameName) # Find the frame index where motion was detected
index_counter = 0 # Counter to keep track of how many frames we collect before and after motion detection

print("Grabbing N frames before motion...")
# Grab N frames before motion was detected
try:
    for current_minute_frame in current_minute_directory_list[current_frame_index:current_frame_index - secondsThreshold:-1]:
        current_minute_frame_full_path = os.path.join(CURRENT_MINUTE_DIRECTORY, current_minute_frame)
        shutil.copy(current_minute_frame_full_path, detection_and_filename_path)
        index_counter += 1
except IndexError as ie: # We've reached EoF for the current directory, move on to the prev (if it exists)
    try:
        size_of_previous_directory_list = len(previous_minute_directory_list)
        for previous_minute_frame in previous_minute_directory_list[:size_of_previous_directory_list - index_counter:-1]:
            previous_minute_frame_full_path = os.path.join(PREVIOUS_MINUTE_DIRECTORY, previous_minute_frame)
            shutil.copy(previous_minute_frame_full_path, detection_and_filename_path)
            index_counter += 1
    except Exception as e: pass # Movement has been caught in the beginning of the loop (or some other issue occurred) - thus there are no
images to grab in the previous directory

print("Grabbing N frames after motion...")
# Grab N frames after motion was detected
timeout_max = 15
timeout_counter = 0
index_counter = 0
index_hour = 0
index_minute = 0
index_second = 0
string_hour = ""
string_minute = ""
string_second = ""
while True:
    if index_counter == secondsThreshold + 1: break # If the N frames has been collected, break
    # Find current time (hour-minute-second), then split all 3 into an array [hour, minute, second]
    if index_second == 0:
        matches = re.findall(r'[0-9]{2}-[0-9]{2}-[0-9]{2}$', filenameSafeCurrentTime)
        splits = re.split(r'-', matches[0])
        index_hour = int(splits[0])
        index_minute = int(splits[1])
        index_second = int(splits[2])
        # print(splits)
        continue

    # Get new image name
    string_hour = str(index_hour)
    string_minute = str(index_minute)
    string_second = str(index_second)
    if len(string_hour) == 1: string_hour = "0" + string_hour
    if len(string_minute) == 1: string_minute = "0" + string_minute
    if len(string_second) == 1: string_second = "0" + string_second
    get_seconds_and_clock = re.sub(r'[0-9]{2}-[0-9]{2}-[0-9]{2}$', string_hour + "-" + string_minute + "-" + string_second,
filenameSafeCurrentTime)
    frame_full_path = CURRENT_MINUTE_DIRECTORY + str(global_var.RPID) + "_capture_" + get_seconds_and_clock + ".jpg"
    # print(frame_full_path)

    # Move image to detection directory
    time.sleep(0.1)
    if os.path.exists(frame_full_path):
        shutil.copy(frame_full_path, detection_and_filename_path)

    #Time/Clock Checks
    if index_second + 1 == 60:
        index_second = 0 # Reset seconds to 0
        if index_minute + 1 == 60:
            index_minute = 0 # Reset minutes to 0
            if index_hour + 1 == 13: index_hour = 1 # Reset hours to 1
            else: index_hour += 1 # Increment hour
            else: index_minute += 1 # Increment minute
        else: index_second += 1
        index_counter += 1
        timeout_counter = 0

```

```

else:
    timeout_counter += 1
    if timeout_counter == timeout_max: break

print("Done capturing motion...")
# Request Notification from CMS
notify_server.notification_for_motion()

```

B.6 motion_detection – This is the main function that uses the camera to monitor for motion.

```

# Set globals
global RAW_CAMERA
global RAW_CAPTURE
global CLARITY_CAPTURE

try:
    RAW_CAMERA = PiCamera()
    RAW_CAMERA.resolution = (WIDTH, HEIGHT)
    RAW_CAMERA.framerate = FRAMERATE
    RAW_CAPTURE = PiRGBArray(RAW_CAMERA, size=(WIDTH, HEIGHT))
except Exception as e: print("Exception Occured: {}".format(e))

# Initialize/Synchronize program time
START_TIME = time.time() if programTime == None else programTime
INTRUSION_THREAD = None # Thread for creating detection directories when motion is detected
first_frame = None # This frame is used to compare against the next frame to determine changes (or motion) between the frames

# Begin monitoring
for image in RAW_CAMERA.capture_continuous(RAW_CAPTURE, format="bgr", use_video_port=True):
    frame = image.array
    CLARITY_CAPTURE = frame.copy()

    # Convert frame to specified size and perform color and blur operations for comparisons
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #frame_gray = cv2.GaussianBlur(frame_gray, (21, 21), 0)

    # Initial run only - no previous frame to compare so set the converted frame to the firstFrame and continue to the next iteration of the loop
    if first_frame is None:
        first_frame = frame_gray
        RAW_CAPTURE.truncate(0)
        continue

    # Compute the difference between the first frame and the new frame - Perform image operations to find contours
    frame_delta = cv2.absdiff(first_frame, frame_gray)
    thresh = cv2.threshold(frame_delta, 25, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.dilate(thresh, None, iterations = 2)
    contours = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = contours[0] if imutils.is_cv2() else contours[1]

    # Loop through the contours
    for contour in contours:
        if cv2.contourArea(contour) < WIDTH: continue # If the contour is too small, move to the next one

        # Generate text and bounding rectangles of the detected object for the view in the windows, then show window
        current_time = datetime.now(timezone("America/Chicago")).strftime(DATE_AND_TIME_FORMAT)
        (x, y, w, h) = cv2.boundingRect(contour)
        cv2.rectangle(frame, (x,y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, current_time, (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.35, (125, 0, 255), 1)

    # Write image
    filename_safe_current_time = current_time.replace(":", "-")
    current_frame_name = str(global_var.RPID) + "_capture_[" + filename_safe_current_time + "].jpg"
    current_frame_name_full_path = CURRENT_MINUTE_DIRECTORY + current_frame_name
    cv2.imwrite(current_frame_name_full_path, frame)

    # Capture intrusion thread
    if INTRUSION_THREAD is None or not INTRUSION_THREAD.isAlive():
        INTRUSION_THREAD = Thread(target=CaptureIntrusion, args=(filename_safe_current_time, current_frame_name, 4))
        INTRUSION_THREAD.setDaemon(True)
        INTRUSION_THREAD.start()

# Clear Stream

```

```

    RAW_CAPTURE.truncate(0)
# End contours for loop

# Get timers and time (long) for minute directory check and image capture
timer_time = time.time()
timer_minute = (timer_time - START_TIME) % 60
timer_second = (timer_time - START_TIME) % 1
#total_timer_minute = 60 - timer_minute
#total_timer_second = 1 - timer_second
#print("Total Minute: {}".format(str(timer_minute)))
#print("Total Second: {}".format(str(timer_second)))

# Minute directory check - Move files in currMinuteDir to prevMinuteDir, if prevMinuteDir exists, delete all contents and store new files in
there (new thread)
if timer_minute >= 59.5:
    previous_minute_directory_list = os.listdir(PREVIOUS_MINUTE_DIRECTORY)
    current_minute_directory_list = os.listdir(CURRENT_MINUTE_DIRECTORY)
    for previous_frame in previous_minute_directory_list:
        #print(os.path.join(PREVIOUS_MINUTE_DIRECTORY, previous_frame))
        os.unlink(os.path.join(PREVIOUS_MINUTE_DIRECTORY, previous_frame))
    for current_frame in current_minute_directory_list:
        #print(os.path.join(CURRENT_MINUTE_DIRECTORY, current_frame))
        os.rename(os.path.join(CURRENT_MINUTE_DIRECTORY, current_frame), os.path.join(PREVIOUS_MINUTE_DIRECTORY,
current_frame))
        #print(os.path.join(PREVIOUS_MINUTE_DIRECTORY, current_frame))
    print("Taking clarity capture...")
    ClarityCapture()
    time.sleep(0.5)

# Capture Image
if timer_second >= 0.9:
    current_time = datetime.now(timezone("America/Chicago")).strftime(DATE_AND_TIME_FORMAT)
    filename_safe_current_time = current_time.replace(":", "-")
    current_frame_name = str(global_var.RPID) + "_capture [" + filename_safe_current_time + "].jpg"
    current_frame_name_full_path = CURRENT_MINUTE_DIRECTORY + current_frame_name
    cv2.imwrite(current_frame_name_full_path, frame)

# Clear Stream
RAW_CAPTURE.truncate(0)
# End while loop

```

B.7 ReadADCChannel – This function reads from a given ADC channel and returns a binary value to be processed.

```

"ADC will read from a given channel and convert its value from analog to digital"
analog_to_digital_channel_read = SPI.xfer2([1, (8 + channel) << 4, 0])
analog_to_digital_channel_data = ((analog_to_digital_channel_read[1] & 3) << 8) + analog_to_digital_channel_read[2]
#print("Channel {} reads a digital value of {}".format(channel, analog_to_digital_channel_data))
return analog_to_digital_channel_data

```

B.8 ReadGPS – This function serially reads from the USB GPS device to record the latitude and longitude of the RPi and vehicle.

```

timeout_max_count = 100
timeout_counter = 0
gps_status_latitude_longitude = [None, None, None]

while True:
    serial_line_read = SERIAL_GPS.readline().decode("utf-8")
    gps_data_read = serial_line_read.split(",")
    if gps_data_read[0] == "$GPRMC" and gps_data_read[2] == "A":
        # Latitude
        latitude_gps = float(gps_data_read[3]) if gps_data_read[4] != "S" else -float(gps_data_read[3])
        latitude_degree = int(latitude_gps/100)
        latitude_minute = latitude_gps - latitude_degree*100
        latitude_actual = latitude_degree + (latitude_minute/60)

        # Longitude
        longitude_gps = float(gps_data_read[5]) if gps_data_read[6] != "W" else -float(gps_data_read[5])
        longitude_degree = int(longitude_gps/100)
        longitude_minute = longitude_gps - longitude_degree*100
        longitude_actual = longitude_degree + (longitude_minute/60)

```

```

    gps_status_latitude_longitude[0] = GPS_NO_ERROR
    gps_status_latitude_longitude[1] = latitude_actual
    gps_status_latitude_longitude[2] = longitude_actual
    break
# Test for Timeout - May be caused by the GPS module not being able to detect its location
timeout_counter += 1
if timeout_counter > timeout_max_count:
    gps_status_latitude_longitude[0] = GPS_COORD_INACCESSIBLE
    gps_status_latitude_longitude[1] = latitude_actual
    gps_status_latitude_longitude[2] = longitude_actual
    break
return gps_status_latitude_longitude

```

B.9 ReadFromSensors – This function reads from all corresponding ADC channels and calls the respective functions to calculate values from the retrieved binary values. Within this function, a call to another function is made to determine whether any thresholds are breached so that the user may be notified. After which, this function returns a dictionary with all the values recorded to be processed into a JSON file.

Init - Global Var

```

# Init - Global Var
global PROCESS_NOTIFICATION

# Battery Thresholds
threshold_bvl = float(threshold_battery_voltage_lower)
threshold_bvu = float(threshold_battery_voltage_upper)
threshold_bcl = float(threshold_battery_current_lower)
threshold_bcu = float(threshold_battery_current_upper)
# Solar Panel Thresholds
threshold_spvl = float(threshold_solar_panel_voltage_lower)
threshold_spvu = float(threshold_solar_panel_voltage_upper)
threshold_splc = float(threshold_solar_panel_current_lower)
threshold_spcu = float(threshold_solar_panel_current_upper)
# Charge Controller Thresholds
threshold_cccl = float(threshold_charge_controller_current_lower)
threshold_cccu = float(threshold_charge_controller_current_upper)
# Temperature Thresholds
threshold_til = float(threshold_temperature_inner_lower)
threshold_tiu = float(threshold_temperature_inner_upper)
threshold_tol = float(threshold_temperature_outer_lower)
threshold_tou = float(threshold_temperature_outer_upper)
# Humidity Thresholds
threshold_hil = float(threshold_humidity_inner_lower)
threshold_hiu = float(threshold_humidity_inner_upper)
threshold_hol = float(threshold_humidity_outer_lower)
threshold_hou = float(threshold_humidity_outer_upper)

# Dictionary to hold {Sensor => Value}
temporary_sensor_dictionary = {}

# Read from Shunts - ADC Channels 3 & 4 & 5
v1 = convert_digital_to_analog_shunt(read_from_adc_channel(CHANNEL_C_BATT), VOLTAGE_REFERENCE, SHUNT_ONE_GAIN, 5)
# From Shunt #1 OpAmp
v2 = convert_digital_to_analog_shunt(read_from_adc_channel(CHANNEL_C_CC), VOLTAGE_REFERENCE, SHUNT_TWO_GAIN, 5) #
From Shunt #3 OpAmp
v4 = convert_digital_to_analog_shunt(read_from_adc_channel(CHANNEL_C_PV), VOLTAGE_REFERENCE, SHUNT_THREE_GAIN, 5)
# From Shunt #2 OpAmp

# Read Battery & Solar Panel Voltage - ADC Channels 1 & 2 - then calculate Battery, Solar Panel, and Charge Controller Current
battery_voltage_value = convert_digital_to_analog_divider(read_from_adc_channel(CHANNEL_V_BATT), VOLTAGE_REFERENCE,
VOLTAGE_BATT_DROP, 2) - v1 # From Voltage Divider #1
solar_panel_voltage_value = convert_digital_to_analog_divider(read_from_adc_channel(CHANNEL_V_PV), VOLTAGE_REFERENCE,
VOLTAGE_PV_DROP, 2) # From Voltage Divider #2
battery_current_value = (v2 - v1) / SHUNT_RESISTANCE
solar_panel_current_value = v4 / SHUNT_RESISTANCE
charge_controller_current_value = (v1 - v4) / SHUNT_RESISTANCE

# Debug
print("Voltage Read from Shunts: V1: {} V, V2: {} V, V3: {} V".format(v1, v2, v4))

```

```

print("Battery Voltage: {}V, Battery Current: {}A, PV Voltage: {}V, PV Current: {}A, Charge Controller
Current: {}A".format(battery_voltage_value, battery_current_value, solar_panel_voltage_value, solar_panel_current_value,
charge_controller_current_value))

# Inner and Outer Temperature Sensors
humidity_inner, temperature_inner = Adafruit_DHT.read(DHT11_SENSOR, DHT11_I)
humidity_outer, temperature_outer = Adafruit_DHT.read(DHT11_SENSOR, DHT11_O)

# GPS
try: gps_latitude_longitude = read_from_gps()
except Exception: gps_latitude_longitude = [GPS_COORD_INACCESSIBLE, 0, 0]

# Check for notification purposes
PROCESS_NOTIFICATION = Process(target=notification_for_thresholds, args=(battery_voltage_value, battery_current_value,
solar_panel_voltage_value, solar_panel_current_value, charge_controller_current_value, temperature_inner, temperature_outer, humidity_inner,
humidity_outer, threshold_bvl, threshold_bvu, threshold_bcl, threshold_bcu, threshold_spvl, threshold_spvu, threshold_spcl, threshold_spcu,
threshold_cccl, threshold_cccu, threshold_til, threshold_tiu, threshold_tol, threshold_tou, threshold_hil, threshold_hiu, threshold_hol,
threshold_hou), daemon=True)
PROCESS_NOTIFICATION.start()

# Exhaust Operations
try:
if temperature_inner >= threshold_tiu: # For Hot Air -> Cold Air
if battery_voltage_value > threshold_bvl:
temporary_sensor_dictionary["exhaust"] = "on"
GPIO.output(EXHAUST, GPIO.HIGH)
else:
temporary_sensor_dictionary["exhaust"] = "off"
GPIO.output(EXHAUST, GPIO.LOW)
except Exception: # Means that temperature_inner did not receive a value from the sensor, so just disable exhaust
temporary_sensor_dictionary["exhaust"] = "off"
GPIO.output(EXHAUST, GPIO.LOW)

# Populate tempDictionary with recorded values
temporary_sensor_dictionary["batteryvoltage"] = battery_voltage_value
temporary_sensor_dictionary["batterycurrent"] = battery_current_value
temporary_sensor_dictionary["solarpanelvoltage"] = solar_panel_voltage_value
temporary_sensor_dictionary["solarpanelcurrent"] = solar_panel_current_value
temporary_sensor_dictionary["chargecontrollercurrent"] = charge_controller_current_value

# Temperature Values
if temperature_inner is None: temporary_sensor_dictionary["temperatureinner"] = "NULL"
else: temporary_sensor_dictionary["temperatureinner"] = temperature_inner
if humidity_inner is None: temporary_sensor_dictionary["humidityinner"] = "NULL"
else: temporary_sensor_dictionary["humidityinner"] = humidity_inner
if temperature_outer is None: temporary_sensor_dictionary["temperatureouter"] = "NULL"
else: temporary_sensor_dictionary["temperatureouter"] = temperature_outer
if humidity_outer is None: temporary_sensor_dictionary["humidityouter"] = "NULL"
else: temporary_sensor_dictionary["humidityouter"] = humidity_outer

# GPS Values
if gps_latitude_longitude[0] == GPS_NO_ERROR: temporary_sensor_dictionary["gps"] = [gps_latitude_longitude[1],
gps_latitude_longitude[2]]
else: temporary_sensor_dictionary["gps"] = ["NULL", "NULL"]

# Return
return temporary_sensor_dictionary

```

APPENDIX C

ISSUES/FIXES

C.1 Raspberry Pi

During initial setup of the RPi, the pre-installed Raspbian OS came with a Linux kernel of version 4.14 that contains hardware compatibility issues with the ADC, i.e. returning wrong readings or no readings at all. As a result, the OS was downgraded to an older version that comes with a Linux kernel 4.9.35-v7+ that is compatible with the ADC. To downgrade Raspbian and the 4.14 kernel, an older image of Raspbian needs to be downloaded via the Raspberry Pi organization's downloads archive:

<https://downloads.raspberrypi.org/raspbian/images/raspbian-2017-07-05/>

BIOGRAPHICAL SKETCH

Luis Garay was born in Brownsville, Texas. After graduating from James Pace High School in Brownsville in 2013, Luis entered The University of Texas at Brownsville, now The University of Texas Rio Grande Valley, Brownsville, Texas. After completing a Bachelor of Science in Computer Science in 2017 from The University of Texas Rio Grande Valley, Luis continued his education by pursuing the computer science graduate program at The University of Texas Rio Grande Valley. During the summer of 2017, Luis was employed as a Research Assistant doing research on Multiple Linear Regression. As a new graduate student starting in fall 2017, Luis has been employed by the University as a graduate assistant aiding in outreach for the College of Engineering and Computer Science, webmaster for the College of Engineering and Computer Science website and its subsites, and as a teaching assistant. In spring 2019, Luis graduated from The University of Texas Rio Grande Valley with a Master of Science in Computer Science. Luis can be contacted by email at garayluis568@gmail.com.