Senior Design Project Report

SOLAR ENERGY MONITORING SYSTEM (SEMS)

Prepared by:

RANDY HANIA
GIOVANNI AZPEITIA

Faculty Advisor: Farid Farahmand
Industry Advisor: Chris Stewart
Client: Invention Planet LLC

https://azpher171.wixsite.com/sems

May 16, 2018

Abstract

Photovoltaic solar energy monitoring system do exist, however, most systems require expensive components to enable users to monitor their solar production. This poses a problem for the potential customer looking to invest into solar panels for the first time since they do not have a solar system installed already. A solution to this issue would be to implement a standalone solar energy monitor to measure and plot solar power with respect to time. This will enable the potential customer to determine solar energy viability at a certain location without the need of having an expensive system already in place. This document will explain how SEMS aims to help potential solar system users to monitor their location's solar energy production.

Table of Contents

1. List of Figures

2. List of Tables

3. Introduction

Solar panels have been an alternative option to generating energy for some time. However, they are inefficient. According to Energy Sage, the most efficient solar cell on the market now is only 22.5% efficient [1]. The average efficiency of solar panels on the market is in the 14-16% range [1]. With such low efficiency, the energy generated by each cell is significantly less than what one would expect. To calculate power, we measure the voltage and current generated with a load and use the power equation. Power multiplied by time is energy and energy is what most people care about because electrical energy is measured in Kilowatt-hour [2]. In 2016, the average annual electricity consumption for a U.S. residential utility customer was 10,766 kilowatt-hours (kWh), an average of 897 kWh per month . [3]

According to the International Energy Agency, Solar power is anticipated to become the world's largest source of electricity by 2050 [4]. We hear it all the time; “solar is the future”, or “solar is good for the environment”, “solar systems can save you money”. Maybe one of your friends or neighbors just got a new solar system installed on his roof and you are thinking about doing the same. Then, you do your research, you contact a solar company, get a quote and realize that the cost is quite high. Now you start thinking about this investment and you want to know if it’s worth it but you want more than just an online calculator, you want something that will be from your location. What if there was a low-cost device than can help you make this decision? A device that will tell you the power generated by a solar panel, save the data for you and help you estimate of how much money you can save?

The purpose of this project is to design and develop a universal solar power monitoring system to measure power, save it in a database and plot it over time. This plot is the energy produced by the used solar panel and it will show the data being collected 24/7. It aims to be accessible to anyone with a link to the provided website. The device will provide a visual way of representing solar energy production and should act as an educational tool to further understand the operation and efficiency of solar panels.

We plan on designing this project with a low-budget while maintaining high accuracy and performance. Solar energy will be harvested from a 25-Watt solar panel, regulated to 5 volts and it will power a Raspberry Pi Zero W. The Raspberry Pi Zero W will calculate the power produced by the panel and transmit the data via WiFi to the cloud. Users will be able to login the website and visually see the power production of their location. The system aims to operate for at

least 12 hours by utilizing a rechargeable lithium ion battery to enable the device to be fully self-sustaining when there is no sunlight.

3.1. Literature Review & Previous Works

Major solar companies, such as Enphase and Smappee, have come up with ways to measure this energy production. Enphase created a cellphone app that links up to a compatible solar panel inverter and it displays the energy produced and consumed for the system [5], but in order to measure the production and consumption of energy, a fully functional solar panel system needs to be already installed. The disadvantage of this method is the cost; the app itself is free, but the solar panel system needs to be already installed, and it has to be their own product [5]. Now, what if there was an energy monitor capable of monitoring any solar panel attached without the hassle of having costly systems already in place? Simply connect a solar panel to the Solar Energy Monitoring System (SEMS) which will measure and display the energy generated and using this data you can calculate the power a solar system can produce in your particular location.

3.2. Problem Statement

Potential solar panel users do not have a certain way of knowing whether their location is viable for solar systems. A solution to this problem is to design a low-cost, standalone solar energy monitoring system to enable the potential user to monitor the energy produced by a solar panel in their particular area. With the data gathered through this device, customers can decide whether solar systems are a suitable option for them.

3.3. Methodology

To design a solar power harvesting system to measure and plot solar power over time. Voltage produced by the solar panel will be stepped down to 5 volts to power a Raspberry Pi Zero W that would calculate energy and plot the data. The solar cell would be switched between the Raspberry Pi Zero W circuit and the battery charging/battery circuit with a relay. The solar cell would switch between the two circuits once an hour to quickly measure and record solar

power then switch back to the battery circuit to charge the lithium ion battery powering the Raspberry Pi Zero W.

3.4. Challenges

There were several challenges in this project. The main one being power consumption. The SEMS is solar powered which means that we need a battery and since one of the requirements is 24 hour monitoring, each module needed to be aimed and tested to maximize battery life. This meant changing several components of the device, such as; step-down converters, microcontrollers, resistors, wires and sensors.

Another main challenge in the project was augo WiFi connectivity. We needed to be sure the SEMS was always connected the the network, otherwise it would not send data and the device would not meet its purpose. To do this, we created a script which contains the name and password of the WiFi provided and it auto-runs when the SEMS is turned on.

Scheduling became a challenge as well because when both members of the team work full-time jobs, it can be very difficult to find a time to work together. This project taught the team how to manage the schedule to make sure the deadlines were met.

3.5 Customer Survey

We asked five people three questions shown below in table I. The goal of this survey was to determine if they were interested in solar systems.

Question Asked	Number of Yes	Number of No
Do you have solar panels installed in your house?	0	5
Would you consider installing solar panels?	2	3
Is cost a factor in your decision(investing in solar)?	4	1

Table I Customer Survey Results

Our survey concluded that most users were interested in solar systems, however, initial cost was a huge factor.

3.6. Marketing Requirements (MR)

- MR1. Low Cost. One of the goals of the SEMS is to help potential solar customers to make a better decision when investing in solar systems. To achieve this, we need to make the SEMS low-cost.
- MR2. Portable. The device is easy to carry and will not present any problems if it needs to be moved. Customers will barely see it in their roof.
- MR3. 24 Hour Monitoring. When making a big decision, such as investing in solar systems, customers need to gather all the data available to use it for their benefit.
- MR4. Web Accessible. Online access makes the SEMS simpler for the customer because it gives them access to their data anywhere.
- MR5. Weatherproof. The device will be on the customer's roof. Therefore it needs to be weather resistant.

3.7. Engineering Requirements (ER)

- ER1. 25 Watt Monocrystalline Solar Panel [MR1]. Average solar systems use 250 W panels. In order to save money to the customer and to make the SEMS low-cost we are using a 25 W panel. This gives us the advantage of using a ratio to scale up our data and simulate the power produced by the 250 W solar panel.
- ER2. 6-22V to 5V Regulator. Our modules need 5 Volts to power up, therefore we need to step-down the voltage produced by the solar panel.
- ER3 INA219 Current Sensor. In order to take measurements to calculate power produced by the panel, we need a high current sensor, the INA219 allows us to take accurate measurements at high currents and voltages.
- ER4. Raspberry Pi Zero W [MR1, 2, 4, 5] This microcontroller consumes almost 50% less power than the Raspberry Pi 3 model B, which means more battery life.
- ER5. 6600 mAh Li-Ion Battery Pack [MR3] In order for the SEMS to monitor 24 hours, we need a batter with enough capacitance and with ideal size to fit in our weatherproof case.
- ER6. Li-Ion Charging Circuit with Booster. This module charges the battery and at the same time it boosts the current to power up our microcontroller.
- ER7. IP65 Weatherproof Case [MR5] The case protects the SEMS against sunlight, rain, snow and strong winds.
- ER8. 20 Ω 50 Watt Resistor. This is the load used in the measurement circuit to measure maximum power.
- ER9. ThingSpeak. This is the cloud we use to store our data, it allows us to use MatLab

4. Implementation

Design and build a solar powered device to automatically connect to WiFi, collect data, save it as a CSV file and at the same time send it to the cloud to display on our website. We achieve this by stepping-down the voltage received by a 25 W solar panel. We use this voltage to charge the battery, power our Raspberry Pi Zero W and take measurements of the maximum power produced by the panel every hour. We then send the collected data to the cloud and using an embedded HTML code, we display our data on our website.

4.1. System Architecture

4.1.1 Hardware Architecture

Figure 1 shows the basic architecture of the Solar Energy Monitoring System. The battery charging system is on the left side; it includes the buck converter, the battery charger, the Li-Ion Battery and the power booster. In the middle of the diagram is the Raspberry Pi Zero with Wifi included.

Top of fig 1 shows the data measuring system, which includes the INA219 current sensor and the load to find maximum power.

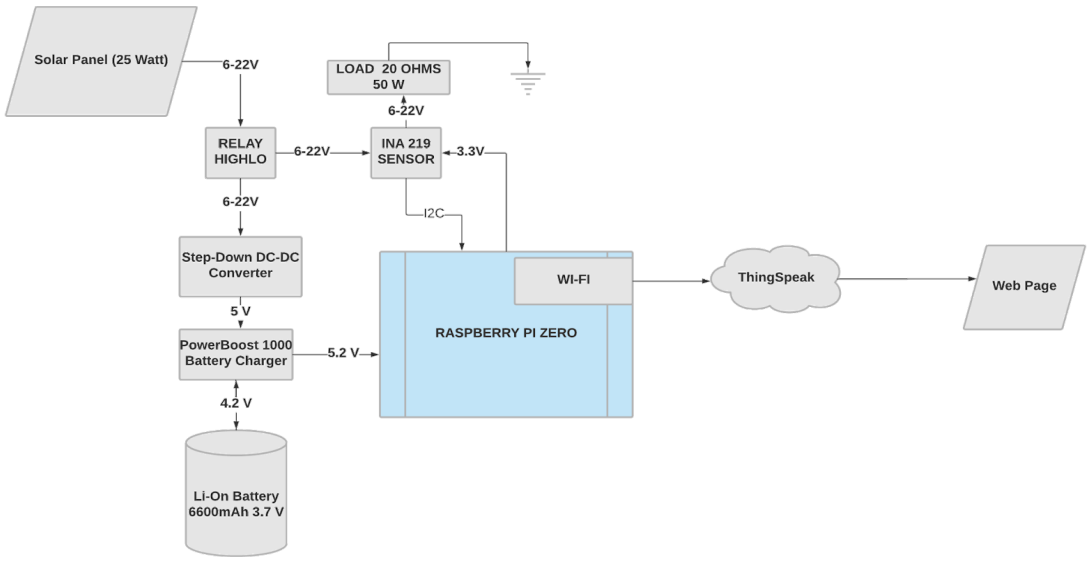


Fig 1 System Architecture of the SEMS

Fig 2 shows a more detailed schematic of the SEMS, it shows pin numbers as well as part numbers used in the device.

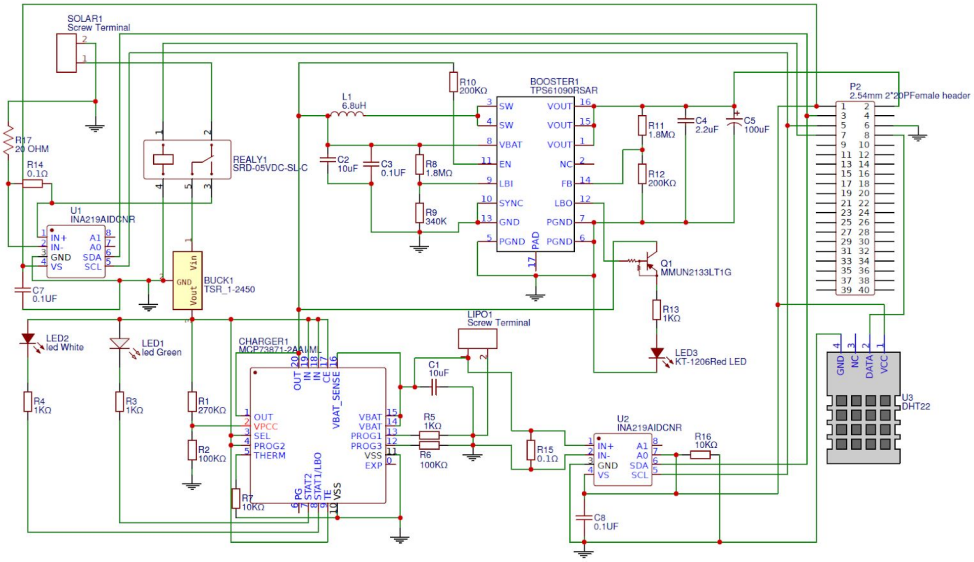


Fig. 2 Detailed Schematic of Solar Monitoring System

4.1.2 Software Architecture

During the software development phase, we experimented with different codes. At some point we had three different codes at one; one for INA219 sensor, another one for the RELAY and another one for the DHT22 sensor. That was causing of CPU to do more work because more codes were being used and also they were constantly running. With the final version we run the code once but we set a crontab to run the code once an hour or any desired time. This way, power consumption reduces and battery life is extended. Figure 3 shows the software block diagram.

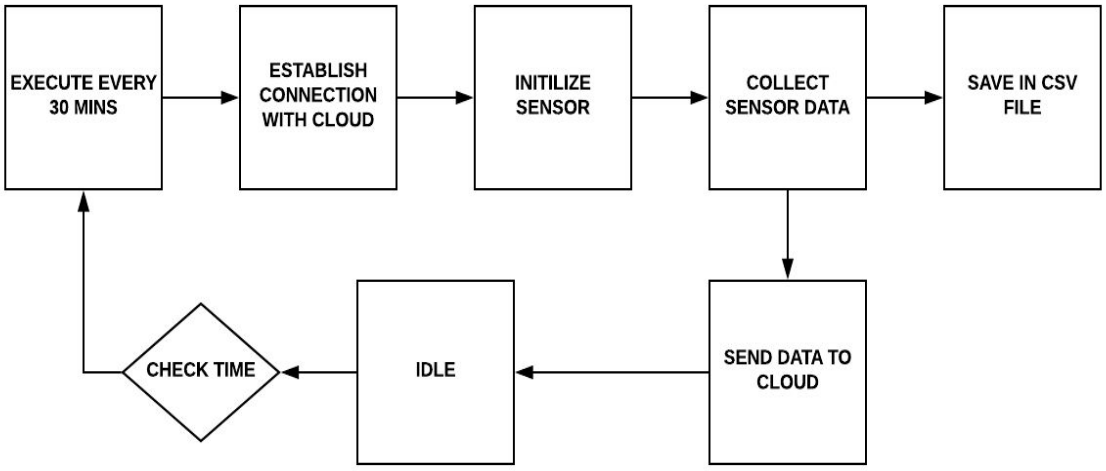


Figure 3. Software Diagram of the SEMS

4.2. Budget Part List

Table II is a breakdown of our proposed budget.

ITEM	MANUFACTURER	PART #	SUPPLIER	PRICE
25 W Solar Panel	**Aleko**	**SP25W12V**	**Amazon**	**$40.00**
Relay	**Tolako**	**BJ-DT0Y-0001**	**Amazon**	**$6.00**
Regulator	**Traco Power**	**TSR 1-2450**	**Adafruit**	**$15.00**
Charger/Booster	**Adafruit**	**1000C**	**Adafruit**	**$20.00**
Li-Ion Battery	**Adafruit**	**ICR18650**	**Adafruit**	**$30.00**
Current Sensor	**Adafruit**	**INA219**	**Adafruit**	**$10.00**
Load Resistor	**Electronics Salon**	**R-A50W/20-2**	**Amazon**	**$5.00**
Raspberry Pi Zero W	**Raspberry Pi**	**Zero W**	**Adafruit**	**$10.00**
Weatherproof Case	**Toyogiken**	**DS-AT-1217**	**Adafruit**	**$20.00**
			Total Price	**$156.00**

Table II. Budget Part List of the SEMS

4.3 Schedule

The schedule did not go as we planned because we had some issues with our power consumption. The project taught us that even when you have a plan, issues can show up and they will affect your schedule. Figure 4 shows the original project Gantt chart and Figure 5 shows the actual Gantt Chart.

	Task Name	Start Date	End Date	Assigned To
1	**Designing and Testing**	**11/14/17**	**12/01/17**	
2	Panel Characterization	11/14/17	11/20/17	Randy
3	Step-Down Circuit	11/14/17	11/20/17	Randy
4	Ion Battery	11/20/17	12/01/17	Giovanni Azp
5	**Development**	**12/01/17**	**03/02/18**	
6	Solar Panel	12/01/17	12/15/17	Randy
7	Battery	12/01/17	12/15/17	Giovanni Azp
8	ADC	12/15/17	01/01/18	Randy
9	Raspberry Pi	12/15/17	01/01/18	Randy
10	Integration	01/01/18	01/26/18	Both
11	WIFI	01/29/18	02/15/18	Giovanni Azp
12	Web Server	01/29/18	02/15/18	Randy
13	Project Integration	02/16/18	03/02/18	Both
14	**Testing and Final Presentation**	**03/03/18**	**05/02/18**	
15	Test the Device with Multiple Panels	03/03/18	03/25/18	Both
16	Correct Issues Found	03/26/18	04/07/18	Both
17	Get Ready for Presentation	04/08/18	04/21/18	Both
18	Get Documents Ready	04/08/18	04/21/18	Both

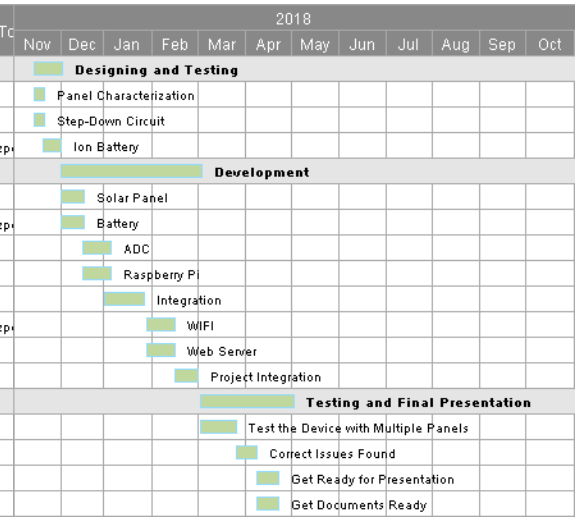


Fig.4 Gantt Chart for the SEMS project

Task Name	Start Date	End Date	Assigned To
Designing and Testing	**11/14/17**	**12/01/17**	
Panel Characterization	11/14/17	11/20/17	Randy
Step-Down Circuit	11/14/17	11/20/17	Randy
Ion Battery	11/20/17	12/01/17	Giovanni Azp
Development	**12/01/17**	**03/02/18**	
Solar Panel	12/01/17	12/15/17	Randy
Battery	12/01/17	12/15/17	Giovanni Azp
ADC	12/15/17	01/01/18	Randy
Raspberry Pi	12/15/17	01/01/18	Randy
Integration	01/01/18	01/26/18	Both
WIFI	01/29/18	02/15/18	Giovanni Azp
Web Server	01/29/18	02/15/18	Randy
Project Integration	02/16/18	03/02/18	Both
Testing and Final Presentation	**03/03/18**	**05/02/18**	
Test the Device with Multiple Panels	03/03/18	03/25/18	Both
Correct Issues Found	03/26/18	04/07/18	Both
Get Ready for Presentation	04/08/18	04/21/18	Both
Get Documents Ready	04/08/18	04/21/18	Both

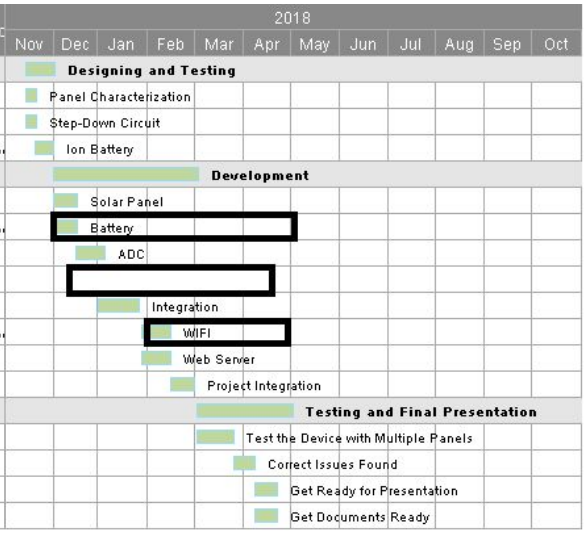


Fig.5 Actual Gantt Chart for the SEMS project

4.4 Testing

4.4.1 Test 1: To characterize the I-V curve of our 25 Watt solar panel

Goal: To determine load required for max power

This test meets ER1, ER8 and MR1

Parts used: 25 watt solar cell, 100 watt potentiometer 500 ohm, Fluke 115 multimeter

Procedure and Results

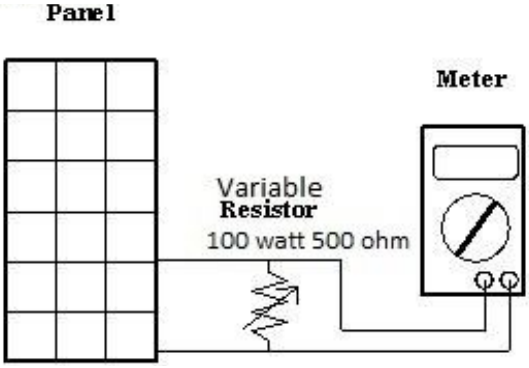


Fig. 5 Circuit to calculate power

Figure 5 shows the circuit used to test out different loads to determine maximum power. A variable resistor was used instead of a regular resistor.

Figure 6 below shows the testing environment. Note: the leaves were not an issue with direct sunlight.

Fig. 6 Test Setup

Results Table III shows the resulting I-V curve based on different loads. The maximum power is achieved at around 20 ohms. Figure 7 shows the I-V curve.

25 W Solar Panel I-V Data			
Solar Panel Voltage (V)	**Measured Resistance (Ω)**	**Measured Current (A)**	**Calculated Power (W)**
20.70	99.80	0.21	4.29
20.10	30.00	0.67	13.47
18.90	**20.00**	**0.95**	**17.86**
10.70	10.00	1.07	11.45
5.40	5.00	1.08	5.83

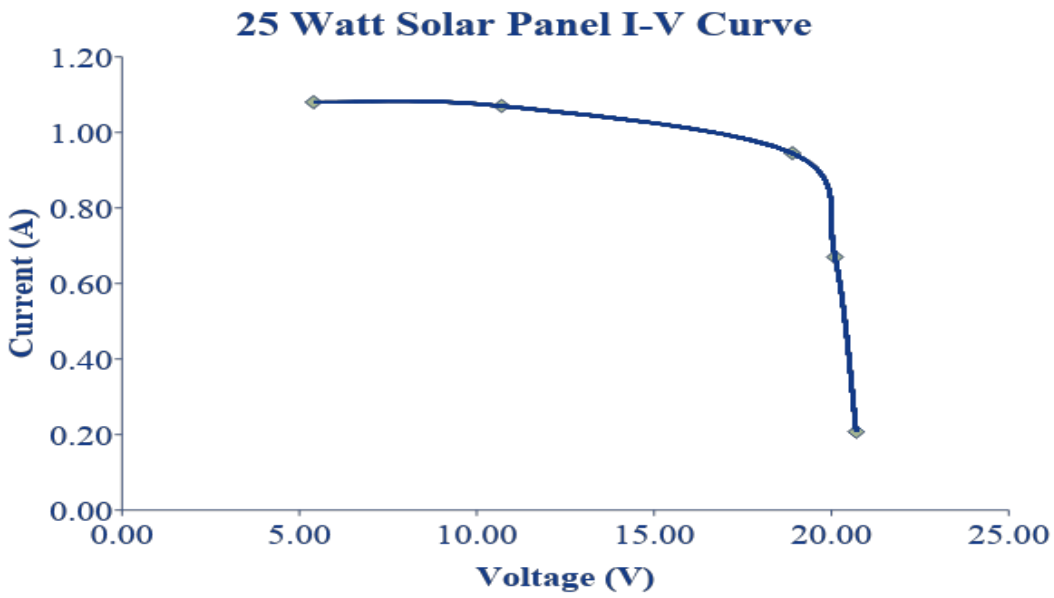


Fig. 7 Data and I-V Curve of the Solar Panel

Conclusion The load required for maximum power is around 20 ohms.

4.4.2 Test 2: Accuracy of INA219 Current Sensor

Goal: To verify accuracy of our INA219 current sensor.

This test meets MR3, ER3, ER8

Parts used: Power supply, ina219 current sensor, raspberry pi, 100 ohm resistor, ohmeter, lcd display. Figure 8 shows the setup used for this test.

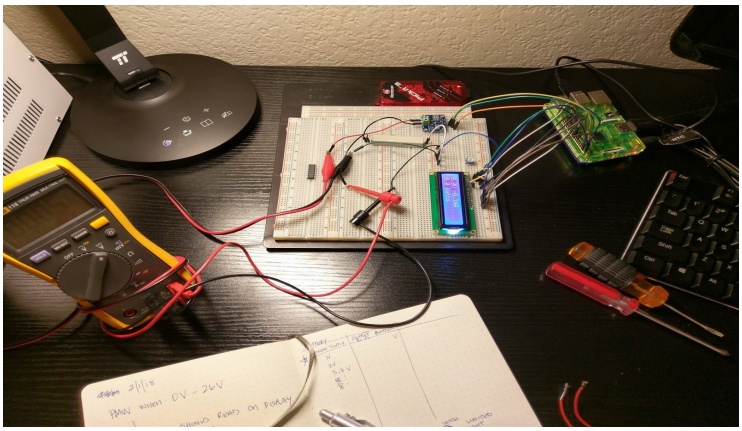


Fig 8. INA219 Test Setup

Procedure and Results: Various voltages were applied from the power supply. The power supply would display the delivered voltage and current the the lcd display in our test circuit would read off what the INA219 was reading. Table IV shows our tabulated data and figure 9 show the plotted data.

INA219 Accuracy Test					
Measured Voltage (V)	Measured Current (A)	Measured Power (W)	INA Voltage (V)	INA Current (A)	INA Power (W)
1.10	0.02	0.02	1.01	0.02	0.02
2.01	0.04	0.08	1.98	0.04	0.08
3.04	0.06	0.17	2.96	0.06	0.17
4.04	0.08	0.30	3.92	0.08	0.30
5.11	0.09	0.48	4.93	0.10	0.48
10.29	0.19	1.99	9.91	0.20	1.94
15.14	0.28	4.28	14.84	0.29	4.35
20.07	0.38	7.63	19.75	0.40	7.80
24.97	0.48	11.94	24.97	0.49	12.31

Table IV INA219 Test Results

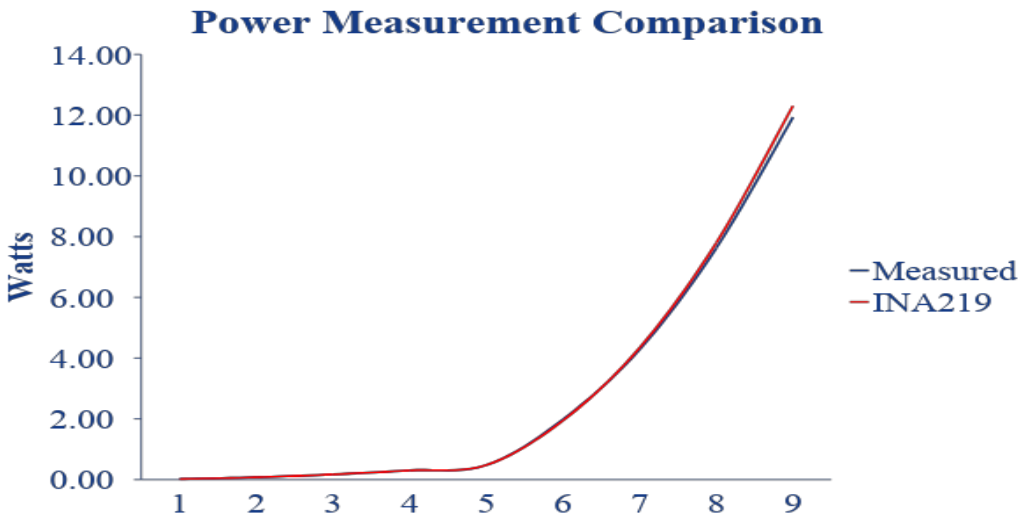


Fig 9. Power Graph

4.4.3 Test 3: Buck Regulator Test

Goal: To characterize the nature of a buck regulator. To take a dynamic voltage ranging from 6-22 volts and regulate it down to 5 volts at 1 amp.

This test meets ER2

Parts used: MP1584EN DC-DC Buck Converter, power supply, ohmeter, lcd display, raspberry pi, 100 ohm resistor.

Procedure and Results: Just like the INA219 test, different voltages were applied to the buck regulator and plotted to show a regulated voltage. Note, the cheap MP1584EN introduced some noisy voltage values for some input values. This may be because of the small nature of the regulator. Data is tabulated in table V.

POWER SUPPLY (V)	MEASURED 100 OHM (V)	MEASURED 100 OHM (mA)	REGULATOR (V)	BOOSTER OUTPUT (V)	BOOSTER OUTPUT (A)
1.09	1.104	11.02897103	0	0	0
2	2.006	20.03996004	0	0	0
3.03	3.043	30.3996004	1.8	0	0
4.03	4.035	40.30969031	2.9	0	0
5.14	5.115	51.0989011	4.2	3.65	0
10.3	10.29	102.7972028	4.4	5.12	1.4
15.2	15.14	151.2487512	4.8	5.12	1.4
20.2	20.07	200.4995005	5	5.12	1.4
25.1	24.97	249.4505495	5.2	5.12	1.4

Table V. Buck Regulator Test Results

4.4.4 Test 4: 1000C LiIon Battery Charging Circuit

Goal: To verify proper charging off li-ion battery.

This test meets MR3, ER3, ER5 and ER6

Parts used: INA219, buck regulator, Adafruit 1000C battery charger, 6600mAh Li-Ion battery, Raspberry Pi Zero W, 25 watt solar panel

Procedure and Results: Battery voltage was monitored with another INA219 current sensor. During hours of sunlight, the 1000C charging circuit would receive sufficient voltage (~ 7 V) and current to charge the battery at a 5V, 1A rate. The battery would charge up to max, 4.2 V, and during no sunlight hours, the battery would only drop down to about 3.8 V, the results can be visualized in figure 10.

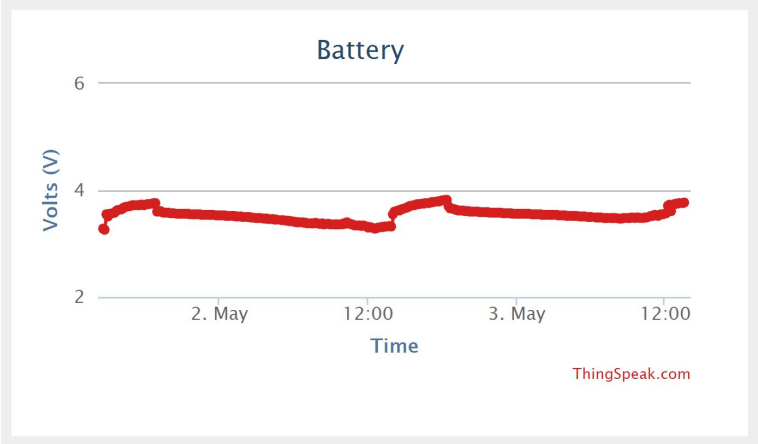


Fig 10 Battery Charging Plot

4.4.5 Test 5 : Raspberry Pi Power Consumption

Goal: To determine which microcontroller consumes less power

This test meets MR4, ER4

Setup: We run the SEMS codes using the two different pi and measure the power consumption using the INA219.

Results: Figure 11 shows that the SEMS with the Raspberry Pi Zero W consumes almost 50% less than running it with the Raspberry Pi 3 model B.

Fig 11 Raspberry Pi Comparison

4.4.6 Test 6: Enterprise Wifi Connectivity with RPi Zero W

This test meets MR4 and ER 4 and ER9

Goal: To connect to SSU-SECURE enterprise wifi using only terminal on RPi Zero W.

Parts used: INA219, buck regulator, Adafruit 1000C battery charger, 6600mAh Li-Ion battery, Raspberry Pi Zero W

Procedure: We only had to edit one system file on the raspberry pi zero to get it to connect to the enterprise wifi. The WPA_SUPPLICANT.CONF file in the wpa_supplicant folder in the /etc directory was needed.
Results: This part was crucial for our project since WiFi is an essential component for our device to work. By modifying this file, we were able to automatically connect to the WiFi.

4.4.7 Test 7: Saving and sending data to the web.

Goal: To collect data from our sensors, save it as a CSV file and display it on a website

This test meets MR4 and ER9

Procedure: Running the raspberry pi with the INA219 Current Sensor connected and display the data.

In a python script that runs automatically when the SEMS powers up and repeats every 30 minutes, we added the following line to be able to send data to the cloud.

```
try:
  baseURL='https://api.thingspeak.com/update?api_key=%s' % myAPI
  f=urllib2.urlopen(baseURL+"&field1=%s&field2=%s&field3=%s&field4=%s&field5=%s" % ('{0:0.2f}'.format(v),
  f.close()

  time.sleep(1)
  GPIO.cleanup()

except:
  print("\n\nThingspeak Failed...")
```

Fig 12 shows the python code to send data to the cloud

Results: We ran the code every 30 minutes and we were able to save all the data as a CSV file in the microcontroller. At the same time, we were able to send the data and save in the the cloud. We were able to display all our data and using MATLAB we modified our visuals.
Figure 13 shows our data collected by the INA219 during 24 hours.

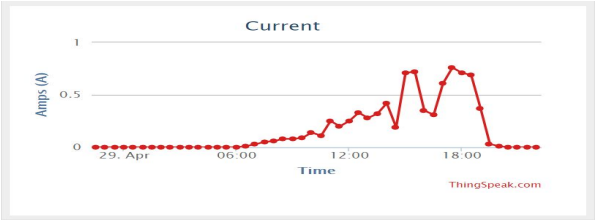


Fig 13 a

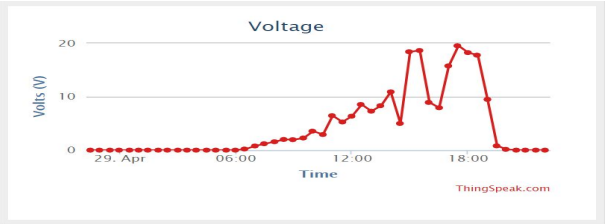


Fig 13 b

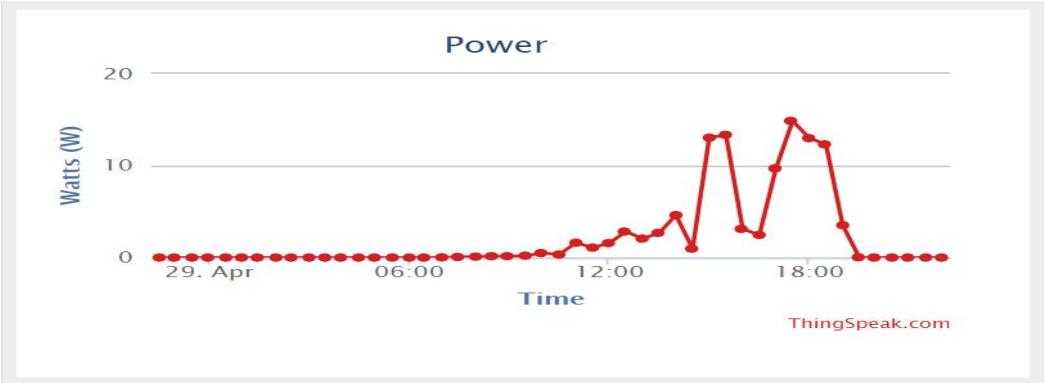


Fig 13 c

5. Final Product

Figure 14 shows the finalized product inside the weather-proof case. We plan on installing the SEMS on a pole on the SSU campus for students and the school community to see and access to our website by scanning a QR code. When they access to our website they will learn concepts about solar power, if they are not familiar with it, such as; voltage, current and power. Another goal of our project is to educate people about solar power and helping them making better decisions by knowing this.

Fig 14 Finalized device

6. Future Works

Future works for the SEMS: 1. connecting data with the PG&E SmartMeter and by doing so, more, better and a more reliable simulation is expected.

2. Computer modeling. We want to use the weather data gathered throughout the entire year and use it to simulate power produced on each season of the year.

7. Conclusion

Solar Energy is the future but investing in a solar system can be a very difficult question because of the cost of the system. We believe that the SEMS will provide the potential solar customer with enough data and knowledge to make a better decision in answering that question. The device is low-cost and very easy to use. Once it is installed in the customer's roof, it can stay there for as long as they need it and it will gather and display data on a website.

8. Acknowledgments

We want to thank our academic advisor Dr. Farid Farahmand for all his help and time. He was definitely the best advisor for us. Also, we cannot thank Chris Stewart enough for all his time and advising he gave us. We could not have done this project without your help.

9. References

[1] Aggarwal, Vikram. “2018 Most Efficient Solar Panels on the Market | EnergySage.” Solar News, EnergySage, 8 May 2018, news.energysage.com/what-are-the-most-efficient-solar-panels-on-the-market/

[2] “Electric Power.” Alternating Current (AC) vs. Direct Current (DC), learn.sparkfun.com/tutorials/electric-power/calculating-power.

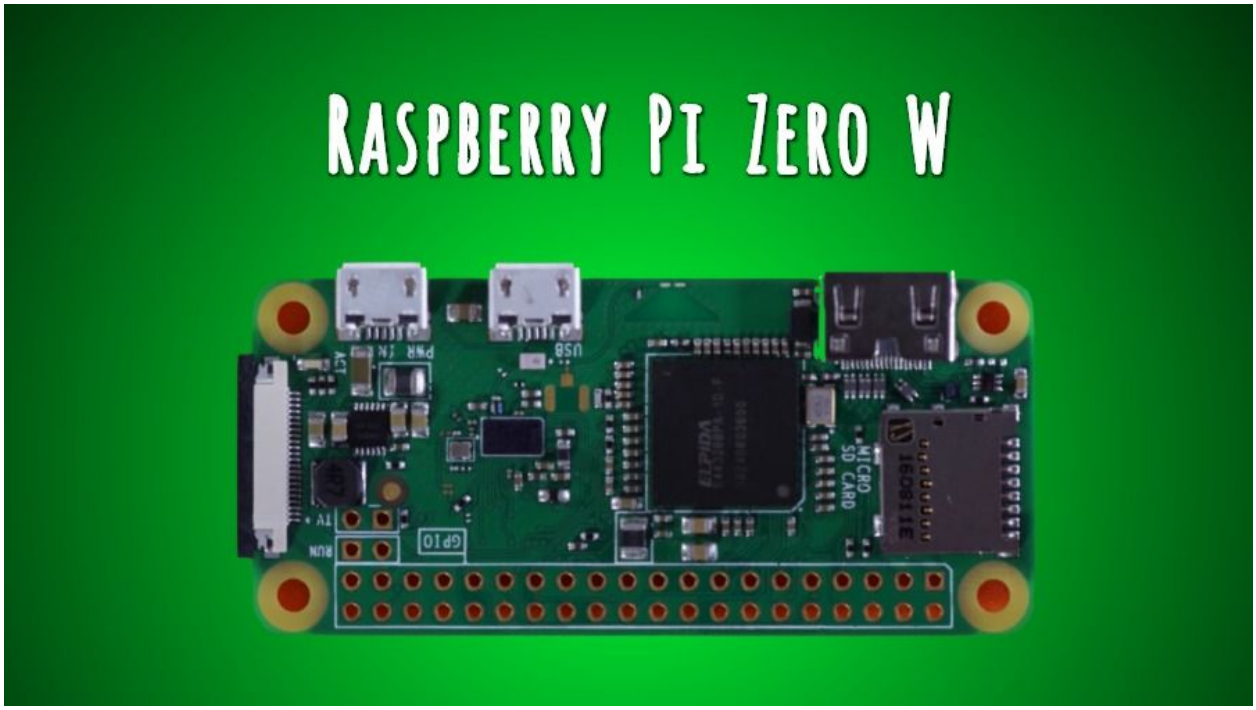
[3] “U.S. Energy Information Administration - EIA - Independent Statistics and Analysis.”- Today in Energy - U.S. Energy Information Administration (EIA), www.eia.gov/consumption/residential/data/2015/

[4] “Statistics.” s: Global Carbon Dioxide Emissions, 1980-2016, www.iea.org/statistics/.

[5] “Solar Monitoring Systems.” EnergySage, www.energysage.com/solar/solar-operations-and-maintenance/solar-monitoring-systems/

[6] “Understand Your Things.” Learn More - ThingSpeak IoT, thingspeak.com/.

Appendix: Tutorial 1 for Auto Connection to WiFi and Crontab

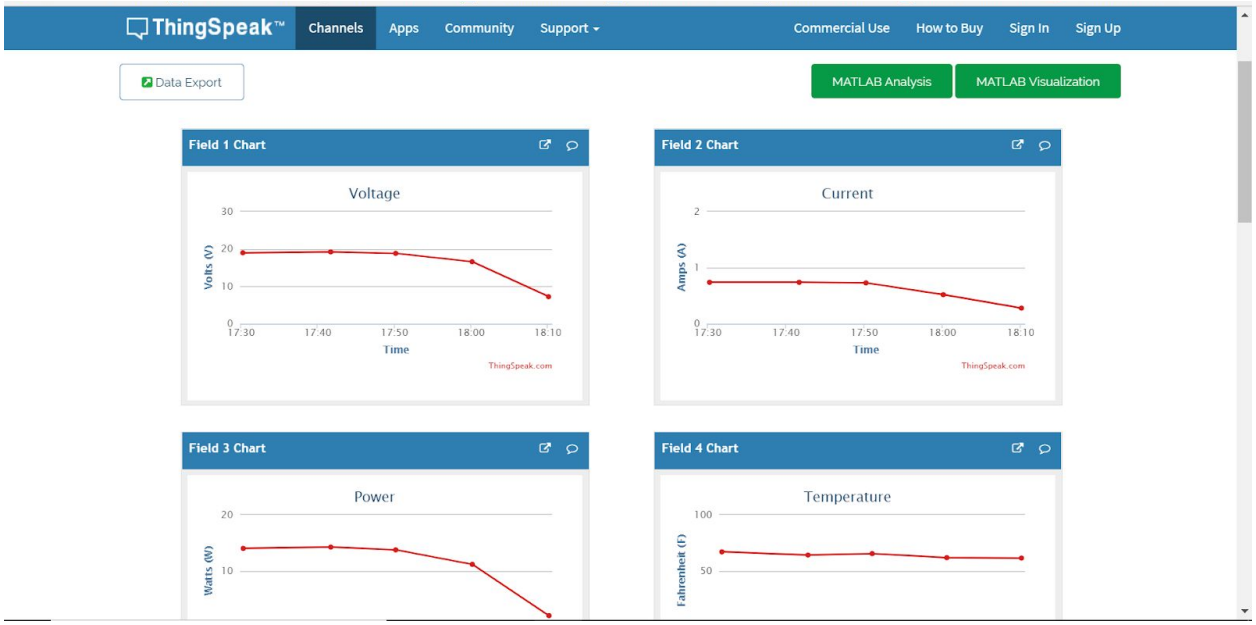


Sensors, or things, sense data and typically act locally. ThingSpeak enables sensors, instruments, and websites to send data to the cloud where it is stored in either a private or a public channel. ThingSpeak stores data in private channels by default, but public channels can be used to share data with others. Once data is in a ThingSpeak channel, you can analyze and visualize it, calculate new data, or interact with social media, web services, and other devices. Storing data in the cloud provides easy access to your data. Using online analytical tools, you can explore and visualize data. You can discover relationships, patterns, and trends in data. You can calculate new data. And you can visualize it in plots, charts, and gauges.

The use of MATLAB makes ThingSpeak a very powerful tool to store data, the website provides you with 3 million messages per year (8,200/day). ThingSpeak stores messages in channels. A message is defined as a write of up to 8 fields of data to a ThingSpeak channel. For example, a channel representing a weather station could include the following 8 fields of data: temperature, humidity, barometric pressure, wind speed, wind direction, rainfall, battery level, and light level. Each message cannot exceed 3000 bytes. So, each time our Solar Monitor System runs the code, it sends data to the website; this counts as ONE message. This gives us plenty of room to send data and it is more than enough for a single project. However, a license is needed. There is a free version offered, which is the one our project is using and it is enough for what we need but after a year the license has to be renewed, otherwise the channel will no longer accept data points.
To access ThingSpeak a MATHWORKS account is needed. If you have one just sign in, otherwise sign up.
Once you are logged in, create a new channel and add the number of fields you will be using, our SMS uses 6 fields.
The next step is getting your API key. Go to the API key tab and save the key number, it will be needed for the code.
Once you know your API, create a python script. You need to add the urllib2 in order to send data.
Add the line myAPI = "<your API code here>" after you added your libraries needed, this line declares your API key number and here is where you'll write down the number given to you on the website.
Next, configure your sensors and read the data.
Then you need to write a loop like the one shown below:

```
try:
  baseURL='https://api.thingspeak.com/update?api_key=%s' % myAPI
  f=urllib2.urlopen(baseURL+"&field1=%s&field2=%s&field3=%s&field4=%s&field5=%s" % ('{0:0.2f}'.format(v),
  f.close()

  time.sleep(1)
  GPIO.cleanup()

except:
  print("\n\nThingspeak Failed...")
```

Here we have the url where our data will be sent to, and the field number where our data will be on. You should see a screen like the one shown below in your channel.

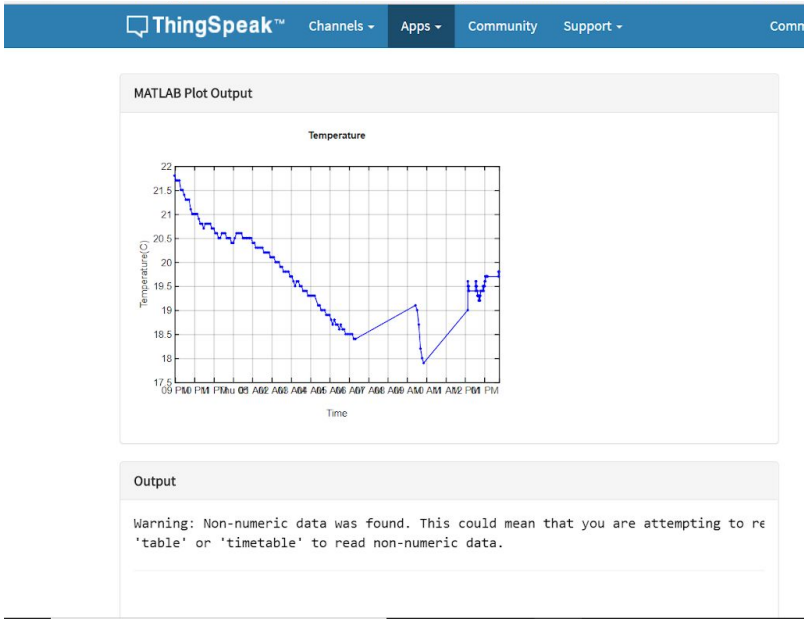


Once we have the data on our website, we can analyze it and this is where we use MATLAB. Click on the 'MATLAB Analysis", located at the top of your channel. Here you will have the option to write down a MATLAB code to visualize and analyze your data better.

Example Matlab code:

```
% TODO - Replace the [] with channel ID to read data from:
readChannelID = 437138;
% TODO - Replace the [] with the Field ID to read data from:
fieldID1 = 2;

% Channel Read API Key
% If your channel is private, then enter the read API
% Key between the '' below:
readAPIKey = [redacted]

%% Read Data %%
%time = datetime('today')-days(5),datetime('today');
%[data, time] = thingSpeakRead(readChannelID, 'Field', fieldID1, 'dateRange', [datetime('today
[data, time] = thingSpeakRead(readChannelID, 'Field', fieldID1, 'NumPoints', 300, 'ReadKey', r
%% Visualize Data %%

thingSpeakPlot(time, data,'LineStyle','-','LineWidth',1,'Color','Blue','Marker','.', 'Grid', '
```

Output:



According to its developers, ThingSpeak is an open source Internet of Things (IoT) application and API to store and retrieve data from things using the HTTP protocol over the Internet or via a Local Area Network. ThingSpeak enables the creation of sensor logging applications, location tracking applications, and a social network of things with status updates. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak.

What is IoT?

Internet of Things (IoT) describes an emerging trend where a large number of embedded devices (things) are connected to the Internet. These connected devices communicate with people and other things and often provide sensor data to cloud storage and cloud computing resources where the data is processed and analyzed to gain important insights. Cheap cloud computing power and increased device connectivity is enabling this trend.

IoT solutions are built for many vertical applications such as environmental monitoring and control, health monitoring, vehicle fleet monitoring, industrial monitoring and control, and home automation.

ThingSpeak allows you to aggregate, visualize and analyze live data streams in the cloud. Some of the key capabilities of ThingSpeak include the ability to:

-Easily configure devices to send data to ThingSpeak using popular IoT protocols.

-Visualize your sensor data in real-time.

-Aggregate data on-demand from third-party sources.

-Use the power of MATLAB to make sense of your IoT data.

-Run your IoT analytics automatically based on schedules or events.

-Prototype and build IoT systems without setting up servers or developing web software.

-Automatically act on your data and communicate using third-party services like Twilio or Twitter.

For the SEMS, two codes are required; one for the temperature and humidity sensor (DHT22) and the second code for the INA219 current sensor.

The codes are the following:

pi@raspberrypi: ~/ThingSpeak

GNU nano 2.7.4 File

```
#!/usr/bin/env python
from ina219 import INA219
from time   import sleep
import urllib2
myAPI = "KBLZQRU37URVOHKF"
ina = INA219(shunt_ohms=0.1,
             max_expected_amps = .62,
             address=0x40)
ina.configure(voltage_range=ina.RANGE_32V,
              gain=ina.GAIN_AUTO,
              bus_adc=ina.ADC_128SAMP,
              shunt_adc=ina.ADC_128SAMP)
def data():
    V, C, P = ina.voltage(), ina.current()/1000, ina.power()/1000
    return (str(V), str(C), str(P))
def main():
    print 'Measuring......'
    baseURL= 'https://api.thingspeak.com/update?api_key=%s' % myAPI

    while True:
        try:
            V, C, P = data()
            f = urllib2.urlopen(baseURL +
                                "&field1=%s&field2=%s&field3=%s" % (V, C, P))


            #print f.read()
            f.close()
            sleep(600)
        except:
            print "exiting..."
            break

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

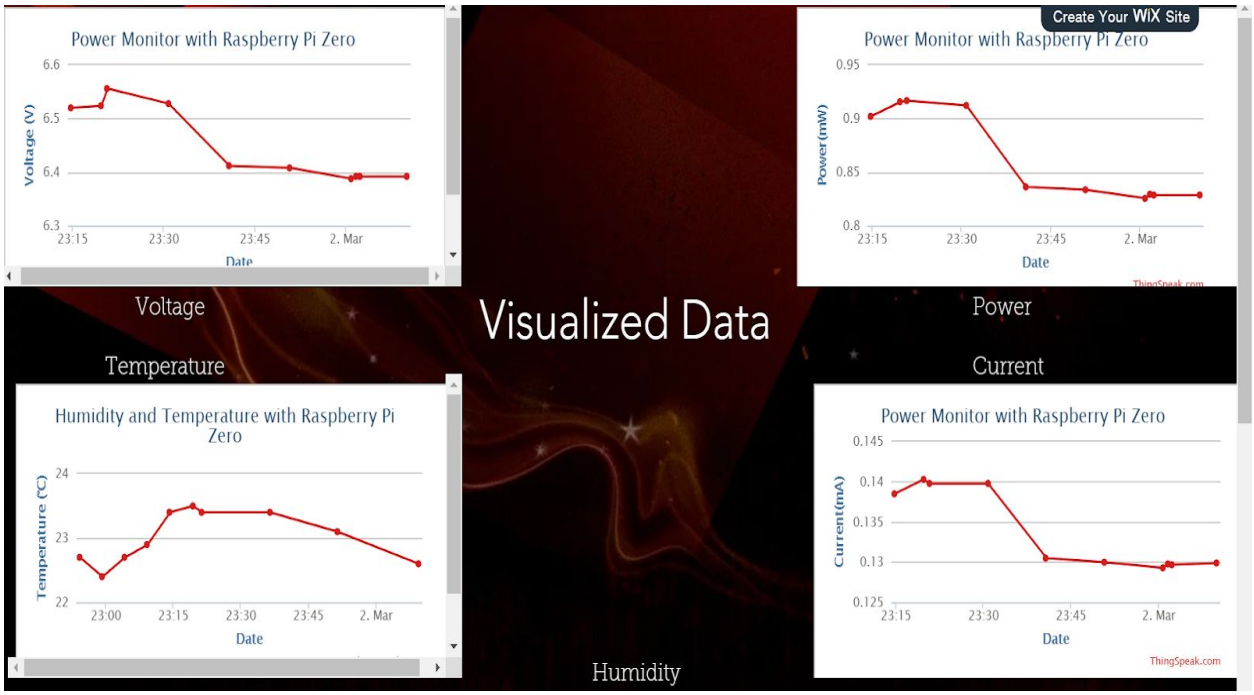
Code for the DHT22 Sensor

pi@raspberrypi: ~/ThingSpeak

```
 GNU nano 2.7.4                                              File: dht22.py

import sys
import RPi.GPIO as GPIO
from time import sleep
import Adafruit_DHT
import urllib2

myAPI = "257XH4JN18NYMBWC"

def getSensorData():
   RH, T = Adafruit_DHT.read_retry(Adafruit_DHT.DHT22, 23) #23 refers to GPIO 23, or pin 16 on the RPi
   return (str(RH), str(T))

def main():
   print 'starting...'
   baseURL = 'https://api.thingspeak.com/update?api_key=%s' % myAPI

   while True:
       try:
           RH, T = getSensorData()
           f = urllib2.urlopen(baseURL +
                               "&field1=%s&field2=%s" % (RH, T))
          # print f.read()
           f.close()
           sleep(900) #uploads DHT22 sensor values every 5 minutes
       except:
           print 'exiting.'
           break

# call main
if __name__ == '__main__':
   main()
```

After running the codes, we go to the ThingSpeak website, where an account was created to be able to own a channel. In this channel, an API key is given, which allows the user to send data to the website, plot it and save it.

The following images are examples of the data being plotted after the codes were ran.



Data sent to the website is plotted in real-time.
Another great feature of ThingSpeak is the ability to download all the sent data as an CSV file. We tested it and it works.
There are limits to the use of the website, but our project will not exceed those limits.
According to their website, the annual cap is 3 million messages; about 8200 messages per day. ThingSpeak stores messages in channels. A message is defined as a write of up to 8 fields of data to a ThingSpeak channel. For example, a channel representing a weather station could include the following 8 fields of data: temperature, humidity, barometric pressure, wind speed, wind direction, rainfall, battery level, and light level. Each message cannot exceed 3000 bytes.
Our channel only has 5 fields and our uploading time will be about every 30 minutes, therefore the limit will not be reached and it makes ThingSpeak a great tool for the SEMS.

[6] ThingSpeak

Appendix: Tutorial 2 Python Script Auto Start using Cron

Cron is a time based scheduler in UNIX systems. A system file in the raspberry pi called crontab is edited with a time format and command to allow an action to take place every declared amount of time.

The format for cron timing is the following:

```
# ┌──────────── minute (0 - 59)
# │ ┌──────────── hour (0 - 23)
# │ │ ┌──────────── day of month (1 - 31)
# │ │ │ ┌──────────── month (1 - 12)
# │ │ │ │ ┌──────────── day of week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                       7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * *  command to execute
```

In terminal type **sudo crontab -e** then enter desired time interval. In our case, we run a python script to turn on the relay and take an instant power measurement under a static load every hour.

0 * * * * sudo python /home/pi/solar/power2.py is what our crontab file contains.