Insight Power Smart Outlet

Brendon Burke, CSE, Mark Chisholm, CSE, Garrett Olson, CSE, and Kriss Strikis, CSE

***Abstract*—Home automation today consists of numerous smart devices and accessories that enable intelligent power management, scheduling and other automated and semi-automated control. These devices, however, currently require a manual setup process which does not scale well to whole-home implementation. Different types of home appliances can have distinctly different power usage profiles, and because smart outlets have access to these waveforms, classification of different plugged in device types is possible. In this paper, we describe our work toward implementing a smart outlet with the ability to automatically distinguish plugged in lights from other kinds of devices.**

I. Introduction

Homeowners today can buy multiple kinds of smart accessories. In particular, smart power outlets allow users to remotely monitor power usage and to remotely switch these devices on and off, either manually or part of automated policies. These include the Amazon Smart Plug[1] and Belkin's Wemo Insight Smart Plug with Energy Monitoring[2]. These offer different combinations of features, like integration with Amazon's Alexa smart home hub or Google's Nest smart home hub. The Amazon Smart Plug does not measure power consumption, but purports to be simple to use and set up. Belkin's advertising does not advertise easy setup, and user reviews express setup frustration. Other smart devices exist which directly include wireless connectivity and control, e.g. Philip's Hue light bulbs[3] or Kenmore's smart refrigerator[4].

On installation of these devices, useful metadata about these devices is not automatically known. A smart outlet's or a smart device's physical location must be manually tagged. In the case of a smart outlet, the attached dumb device must be manually specified. This device tagging would not only be necessary at first setup, but every time the user plugs in something different. The metadata associated with the old device remains until manually updated. A smart device could change its location, and has a cost overhead compared to traditional appliances. Managing these devices creates a high-friction process for end users.

A smart outlet could analyze the power usage of a plugged-in device, given that it could already perform power monitoring. Because e.g. a lamp has a notably different power usage profile from a refrigerator, this analysis could identify the type of plugged in device. If all smart outlets in the home identify the type of attached device, many types of home automation policies would be immediately implementable without any configuration required. For example, a homeowner could command all lights, and only the lights, to turn off.

Existing smart outlets measure power usage and provide remote on/off switching functionality. In order to add device identification to this functionality, a new outlet design requires an additional computation ability and the device classification algorithm in software. Previous research shows that such classification is possible with greater than 90% accuracy[5]-[7]. Our team built on this research by building a smart plug with these features to demonstrate the capability with real appliances. This would further the ability of a smart home installation to implement automated policies such as turning off all lights in the home when a user has left, regardless of where those lights are plugged in and without requiring special lights, while also ensuring that other devices remain powered, even if they were plugged into an outlet previously used for lighting.

System Specifications

Goal	Status
Device Weight	<1 lb
Device Dimensions	~ 12 x 3 x 3 cm (L-W-H)
Cost of Manufacture (1000)	~ $19
Frequency of Power Sampling	1 second or less
Outlet/Companion App Response Time	1 second or less
Power Measurement Error	<5%
Classification	<5 seconds with 80% accuracy

Table 1. System Specifications for our design.

Table 1 details our system specifications. The device must be small and light enough to easily plug into a wall outlet and stay there; this guides the weight and size requirements. The system cost should be affordable to consumers, and so we target $50. The power measurement should be updated in near real time, or about once per second. The power graph visible to the user should be updated about as fast, and the device

power switching should be about as responsive. For power measurement error, we expect to be close to existing power measurement devices, one of which we will use to benchmark our device. When we classify a device we want to be quick with letting the user know which type of device they have plugged in while correctly identifying the type of device.

II. Design

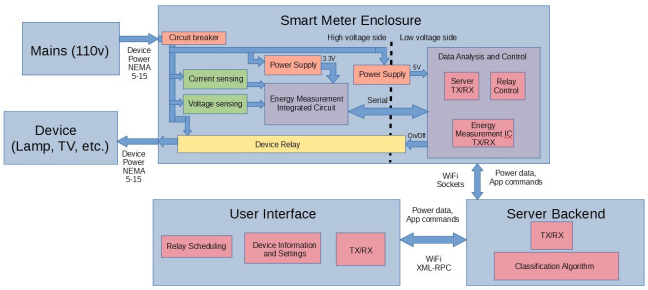


Fig. 1. The block diagram for our device demonstrated at FPR.

A. *Smart Meter Enclosure - High Voltage Side*

The first section of the design we will be looking at is the high voltage "side" of the outlet itself. This part of the design is where the actual power sensing and third party device interaction will take place. We are using a simple voltage divider as well as a shunt resistor to obtain the information needed to calculate power. This information is fed into our off-the-shelf energy management IC, the Cirrus Logic CS5490, which uses this information to calculate various aspects of the power a device is drawing. Additionally, this part of the design includes a power relay, to allow the outlet to essentially cut power to devices plugged into it. The results of this block can be easily tested by comparing the output to the power usage specifications of a given device.

The voltage input pins on the CS5490 accept a differential analog input with a maximum peak voltage of 250 mV when using the default gain of 10. The minimum and maximum common mode voltages accepted are -250mV and VDDA (3.3V)[8]. The manufacturer does not specify input resistance or maximum allowable current across these pins but the provided application notes recommend using a 1 kΩ resistor on one input pin. To provide the voltage signal we use a voltage divider, targeting an output signal peak of 100mV. For a 120V RMS input, actual peak input is $120V*\sqrt{2} \cong 170V$. Thus, to divide this signal to 100mV, we choose R1 = 1700*R2. Actual values used are 1.74 MΩ and 1 kΩ. To satisfy the common mode voltage range requirement we ensure that the energy measurement IC power supply is referenced to the neutral line of the input voltage.

The CS5490 accepts a current input signal from a variety of sensor types, but for size and reliability we choose to use a shunt resistor. The current input pins have very similar specifications to the voltage pins. To choose a shunt resistor value, we target an output signal peak of 100 mV. With an expected maximum current of 10 A, the corresponding resistor size is 0.01 Ω and is chosen as such. To limit the current through the input pins, the manufacturer's application notes suggest a 1 kΩ resistor on each input pin. To satisfy the common mode voltage requirement we place one end of the shunt resistor on the common ground on the neutral line.

B. *Smart Meter Enclosure - Low Voltage Side*

Our initial prototypes included optocouplers to isolate the Data Analysis and Control portion of the design from the potentially high current coming from rest of the device. Within this low voltage block, we used a Raspberry Pi 3B+[9], and later, an Onion Omega2+[14] to execute our classification algorithm. The microcontroller would connect to our local wireless network, connect to our backend server, then send initialization commands to the energy measurement IC to start it running in continuous conversion mode. Once in this mode, the CS5490 continually updates internal registers corresponding to each measurement which it calculates. The microcontroller sends additional serial commands to read these registers and temporarily stores them. Because these values generally correspond to a value between zero and one, or between negative one and positive one[8], we multiply the returned values for active and reactive power by a constant to scale it to the correct value. We find appropriate constant values by calibrating our device with a known load, verified by using a separate meter, the Kill-A-Watt. Once we have the correct, scaled values for power the microcontroller sends them to the server.

C. *User Interface*

The last block of the design is the user interface for the system we are designing. This will take the form of an Android app, written in Java in android studio. The app will provide a simple, but intuitive interface that will allow users to monitor power usage data from multiple outlets in real time. It will also provide users with the ability to control the outlet by turning it on and off and setting schedules and protocols for certains types of devices. The app will consist of two "screens"; the home screen where you are able to select a currently connected outlet, and a Graph screen, which will show information about the selected outlet both graphically and textually. Users will select which outlet they wish to view information about by accessing the side bar, which can be brought up by swiping right on the screen. The sidebar will also include a "refresh" button which will allow the user to refresh the list of detected outlets. Once a user selects an outlet the graph screen will appear displaying information such as what the current power usage is and what device is connected to the outlet. The user can return to the home screen, or navigate to other outlets information, by simply swiping right and bringing up the side bar again and selecting the desired option

D. Client-Server Architecture

For the communication between smart outlets and the server, there is a connection based on regular sockets. The connection is TCP based. For every outlet that connects, the server creates a thread to handle that specific outlet to listen for incoming power data. The power data is transmitted over the network in a json byte encoded format. The power data can be broken down into three bytes of data where each byte represents (from least significant to most significant) the value of the power reading. For example, if we read 0xAABBCC from the EMIC, the correct way for the server to interpret it is CC-BB-AA[8]. The server appends the final power data to a text file containing the power history of a specific outlet.

The companion app and the server communicate over a different protocol. The protocol is XML-RPC which allows a client to directly call methods on the server and receive the value that the methods return. The advantage of this model is easy communication between programs which use different programming languages. When we initially tried creating sockets for communication between the companion app and the server we were running into issues where either side could not properly decode bytes that are sent through the sockets. Our primary use for XML-RPC is to retrieve power samples from the server for a specific outlet.

The process for the companion app initial setup is as follows. First the companion app requests the list of currently connected outlets from the server. This list consists of the randomly generated UUIDs assigned to each outlet on first connection. From this list of UUIDs the companion app can request either power data or send commands to any specific outlet.

E. Classification Through Dynamic Time Warping (DTW)

The data we will be looking at to classify a device is the startup characteristics of the device. We want to have the outlet be able to classify a device quickly and accurately, so by looking at the startup active and reactive power of a device like in [7] we can get an idea of what kind of a device it is. Table 2 shows the three types of classes we are able to identify.

TYPES OF CLASSES FOR DEVICES

Resistive	Inductive	Non-Linear
Lamp	Fan	Computer/Laptop
Heater	Vacuum	Cell Phone
Toaster	Refrigerator	Television

Table 2. Classes that a device can fall into for our outlet.

Devices with lighting and heating elements fall into the resistive class. Resistive devices when powered on will rise to a steady level and could contain a slight decrease or increase in active power, and have little to no reactive power. Devices that contain a AC motor in order to achieve its task are inductive type loads. Inductive loads will exhibit a big spike in both active and reactive power which will decrease to a steady state until powered off. Lastly, most other electronic devices can be considered non-linear. A nonlinear device has power that will behave in an erratic way. Depending on what the non-linear load is doing, both the active and reactive power will greatly vary over time with the possibility of reactive power being greater than active power[7].

In order to properly utilize the data we are receiving, we have chosen to use Dynamic Time Warping, or DTW to classify our devices. In section G we go into depth of how DTW works and how we use it for classification.

F. Energy Measurement Integrated Circuit (EMIC)

The energy measurement chip we use is the Cirrus Logic CS5490[8]. It operates at a voltage of 3.3V, but connected to mains voltage indirectly through the current and voltage sensor input pins. To protect our microcontroller and connected accessories from a possibly dangerous voltage presence at the EMIC, we electrically isolated the two using optocouplers, which allow them to communicate without current flowing between them. The specific pins which are isolated between the two devices are the RESET, UART RX, and TX pins. In our minimized prototype, the Omega would not have connected human interface devices like a keyboard and mouse that the Raspberry Pi had during initial prototyping. For this reason and that a user would be physically isolated from the circuitry by a plastic enclosure, we removed the optocouplers from the design.

The Omega[14] uses a UART to communicate with the chip. The interface is managed through a python script with the PySerial library[15]. The specific settings for the UART interface that we used to communicate with the energy management IC are as follows. The baud rate is 600Bd, there are no parity bits, there is one stop bit, and there are eight bits for the payload size.

The CS5490 updates its internal registers with power calculations at regular intervals. The frequency which the chip updates these registers is configurable by choosing how many samples the chip should measure before averaging and storing them—the default is 4000 samples, and because the chip takes that many samples per second, the measurement registers are by default updated once per second. We chose to increase the sample rate to twice per second in order to capture more detail in the startup power waveforms. We experimented with increased sampling frequencies, but found that the serial baud rate would become a bottleneck after about four hertz. While the baud rate is configurable, we did not successfully increase it. In our limited attempts, the Omega could no longer read serial communications from the CS5490 after setting the register specifying the baud rate.

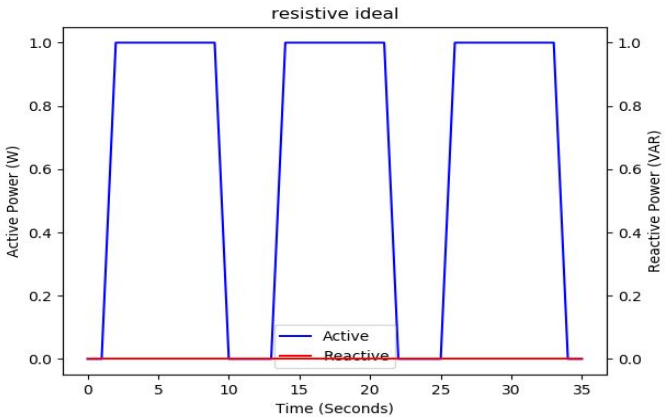
G. Dynamic Time Warping

Dynamic Time Warping is a standard way to measure how different two time series waveforms are. The value calculated from DTW is known as the distance. DTW calculates this distance by using a distance matrix. The values in a distance matrix can be calculated by using Equation 1, where Ai is a point in a time series and Bj is a point in another time series, D[a,b] is the distance calculated for previous points

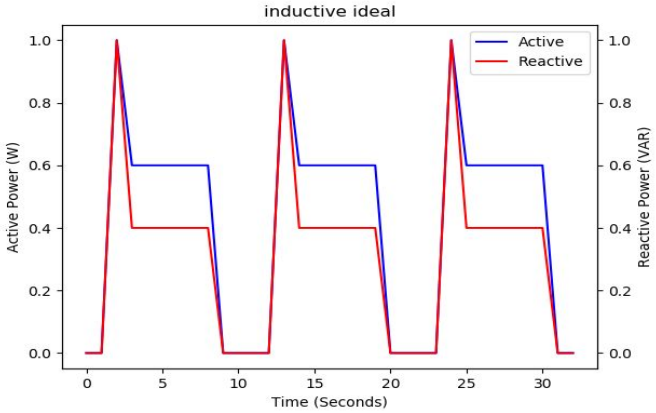
$$| Ai - Bj | + min(D[i-1, j-1] , D[i-1, j] , D[i, j-1]) \quad (1)$$

Once all the values in the distance matrix are calculated, the shortest path is found from the last value calculated to the first. As the distance calculated increases, two time series are going to be more dissimilar. Unlike Euclidean Distance which a small difference in waveforms can result in a huge distance calculated, DTW compresses the waveforms and aligns them. This compression results in a distance that is more accurate because it can account for slight variations in the waveforms[12].

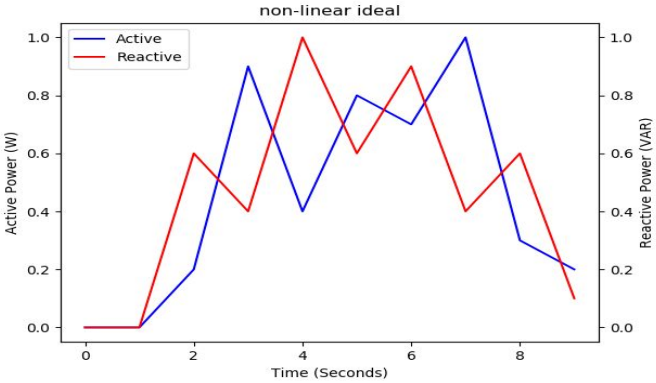
For our classification, we normalize incoming power data from the outlet between 0 and 1, and then we compare the waveform using [13] to what an ideal waveform for the different classes would look like. The comparison that results in the lowest distance is what the device is classified as. Figure 2 shows our ideal waveforms taken from what we learned in section E.



(a) An ideal resistive waveform for a device.



(b) An ideal inductive waveform for a device.



(c) An ideal non-linear waveform for a device.

Figure 2. (a) (b) and (c) show what an ideal device would behave like for each of the classes we are using.

III. Project Management

Goal	Status
Device/App Response Time < 1 second	Completed
Power Measurement Within 5%	Completed
App Graph Implemented	Completed
Power Data Samples Every 1 second	Completed
Classification 80% accurate	Incomplete (71.4%)

Table 3. Project goal completion status.

Throughout SDP our team has been able accomplish most of the goals we set out to reach by now. Our app and device both have a response time that is within our 1 second specification; the app is able to easily graph in real time the data that it is receiving from the outlet, and our device samples the power data every second. We are also able to measure power with 5% and are able to accurately classify a number of different devices. Classification is not up to where we would like it to be. We currently have an overall accuracy of 71.4% with our goal having been 80%. Table 4 is a breakdown of the accuracies by type, and then by device. We are the most accurate in correctly identifying resistive loads as they are and we struggle with inductive and non-linear loads.

Our group was able to effectively divide the work throughout the year. With Kriss spearheading hardware development, with the help of Garrett and Mark. Brendon and Garrett implementing the companion app, as well as establishing a wireless connection between the app and device. Mark assisting in both areas while researching and implementing our classification algorithm using DTW with Garrett's assistance. In order to communicate with one another when not in person, we have been primarily using Discord and have been emailing one another in order to stay on track and keep each other updated on the progress of each other's work.

Pertaining to Table 3 here is the explanation for how we made the determination that we completed our goals. With respect to the response time for the device and the app, this time relies almost entirely on network latency and the responsiveness of a few hardware elements. The network we setup for our demo was very similar to a network setup that could be found in almost any household. It consisted of the smart outlet connected to a central router with the server also connected to the same router. the average network latency between them we measured to be around 2ms. When the companion app connects to the server and the outlet wants to communicate with the companion app, the round trip latency is approximately 140ms. This means worst case network latency will be approximately 144ms. We obtained the network latency data by using the Linux ping network utility. The response time of the mechanical relay, which controls the power flow to the attached device, is at worst case 10ms[10].

Resistive	**26/28**	**92.8%**
Tall lamp	9/9	
Coffee maker	4/6	
Toaster oven	3/3	
150W lamp	10/10	
Inductive	**17/29**	**58.6%**
Vacuum (big)	7/9	
Vacuum (small)	1/8	
Fan	7/10	
Refrigerator	2/2	
Non-linear	**12/20**	**60%**
Printer	5/6	
Nintendo Switch	2/5	
Laptop	5/9	
Total	**55/77**	**71.4%**

Table 4. Accuracy of load type and device specific.

In order to benchmark the accuracy of our device's power measurement, we compare to a Kill-A-Watt device. This device has an LCD panel displaying power consumption of an attached device in watts. We read this displayed number and compare it to our own device's power measurement. The benchmark device, the Kill-A-Watt, visibly displays a precision of only +/- 1 W. Our low voltage testing did result in measurements within this range. However, because our low voltage test load consumed only about 4 W of power, the benchmark device's precision was insufficient for us to determine that our device's measurement was within 1% of it. Once we proceeded to measuring power usage by 110V

devices, we could more precisely gauge our accuracy by using a 150W light bulb, for example. After calibration, we would read this bulb's active power as 150W, with brief, infrequent measurements reading up to 157W. With this load, we then have accuracy approximately with 5% of our expected value.

The graph we use in our app was implemented using the GraphView Library for Android Studio[11]. This library offers a wide range of functions that allow us to change various attributes of the graph, all of which can be changed dynamically to adapt to our incoming data so that it is displayed in a simple, easy to understand way. We often change the scale of the graph to make sure all data points are being displayed, for example if the initial max value of the graph is set to 10, but we plug in a device with a power usage greater than 10W, the graph will automatically adjust, setting the new maximum to at least the highest observed value.

The way we attain the goal for power data samples every one second is to properly configure the energy management IC. On startup we send commands over UART to the energy management IC to setup. The first command we send is a hex value which changes the area of memory we can retrieve values from. These areas are called pages and on startup the active page is number zero. The commands on page zero allow us to query chip status as well as peak voltage and peak current. To get total apparent power, which we are using for classification. We send a command to change the active page from number zero to number 16. Once in page 16 we can query measurements like instant current, instant voltage, instant power, apparent power, and total apparent power. Before we send commands to get measurements we send a command to tell the energy management IC to start continuous conversion. Continuous conversion is how the energy management IC takes measurements and the frequency of the samples is by default 1Hz.

For classification, we believe most of the error is due to our devices not matching our ideals. Specifically, with non-linear loads, we chose an arbitrary varying waveform as our ideal, so if a non-linear device is varying in such a way that is significantly different from our ideal, that could be one source of error. Another issue could be that we are not sampling fast enough to always catch the spike that an inductive load should be producing. We also have only done trials in the tens range, with more time we could have done hundreds of trials and would have been able to narrow down sources of error.

IV. Conclusion

Overall, we are satisfied with the final result of our project. We managed to stay on track over the course of the year and implement almost every feature we envisioned showing at Demo Day. This project was a very educational experience as we were able to explore some topics that as a group of four CSEs, we may have had little to no exposure to otherwise.

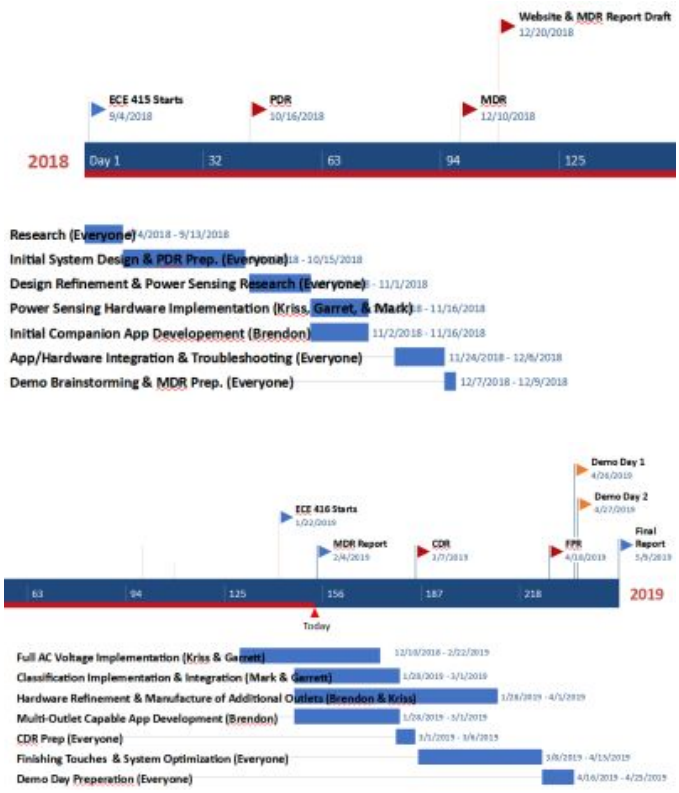


Fig. 3. Timeline showing our progress throughout the year with important deadlines labeled..

V. Acknowledgments

We would like to give a special thank you to our advisor David Irwin for the time and effort he has dedicated to helping us bring this project to life. We would also like to thank Shira Epstein and the staff of M5 for all the help and resources they provided to us throughout the process. Finally, we'd like to thank Professors Hollot and Soules for helping to keep us organized and on track.

VI. References

[1] *Amazon Smart Plug*. Available: https://www.amazon.com/Amazon-Smart-Plug-works-Alexa/dp/B01MZEEFNX/.

[2] Anonymous "Wemo Insight Smart Plug with Energy Monitoring," Available: https://www.amazon.com/dp/B01DBXNYCS.

[3] Anonymous "Philips Hue White Ambiance BR30 60W Equivalent Dimmable LED Smart Flood Light," 2019. Available: https://www.amazon.com/Philips-Hue-Ambiance-Equivalent-Assistant/dp/B01KJYSOHC/.

[4] Anonymous "Kenmore 4675049 Smart 24 cu. ft. Counter Depth French Door Bottom Freezer Refrigerator in Black - Works with Amazon Alexa," 2019. Available: https://www.amazon.com/Kenmore-Smart-French-Bottom-Mount-Refrigerator/dp/B0745QL5HD/.

[5] L. P. R. Ambati and D. Irwin, "AutoPlug: An automated metadata service for smart outlets," in *2016 Seventh International Green and Sustainable Computing Conference (IGSC),* 2016, . DOI: 10.1109/IGCC.2016.7892604.
[6] S. Barker *et al*, "Non-intrusive load identification for smart outlets," in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm),* 2014, . DOI: 10.1109/SmartGridComm.2014.7007704.
[7] S. Barker *et al*, "Empirical Characterization, Modeling, and Analysis of Smart Meter Data," *IEEE Journal on Selected Areas in Communications,* vol. 32, *(7),* pp. 1312-1327, 2014. . DOI: 10.1109/JSAC.2014.2332107.
[8] (Mar). *CS5490 Product Datasheet*. Available: https://statics.cirrus.com/pubs/proDatasheet/CS5490_F3.pdf.
[9] Anonymous "Raspberry Pi 3 Model B+ product brief," 2018. Available: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf
[10] *HF3FD Subminiature High Power Relay*. Available: http://www.hongfa.com/pro/pdf/HF3FD_en.pdf.
[11] *Android Graph View plotting library*. Available: http://www.android-graphview.org/.
[12] *How DTW (Dynamic Time Warping) algorithm works.* From: https://www.youtube.com/watch?v=_K1OsqCicBY.
[13] (May)*DTW (Dynamic Time Warping) python module*. Available: https://github.com/pierre-rouanet/dtw.
[14] *Onion Omega2 Documentation*. Available: https://docs.onion.io/omega2-docs/omega2p.html.
[15] *pySerial Documentation*. Available: https://pythonhosted.org/pyserial/