UNIVERSITY OF CALIFORNIA

SANTA CRUZ

ELECTRICAL APPLIANCE IDENTIFICATION USING FREQUENCY ANALYSIS

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Olexiy Burov

September 2019

The Thesis of OLEXIY BUROV
is approved:

_____

Patrick Mantey, Chair

_____

Yu Zhang

_____

Martine Schlag

_____

Quentin Williams
Acting Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Electrical Appliance Identification Using Frequency Analysis

by

Olexiy Burov

We prototyped and then created a hardware device (SEADS) [2] that is able to sample high frequency current and voltage data using up to eight channels. SEADS was installed in the generic household with a variety of electrical appliances with two sensors on both lines of a single phase 240V circuit. The current and voltage measurements were taken applying bandpass filters at different frequencies of interest to isolate purely resistive and inductive loads. We identified the features of devices which consume most of the energy on the electrical panel and came up with algorithms to automatically identify when these devices are on or off. This information presents a great value to the end user since it allows to identify one's energy usage patterns and make more educated decisions. This is especially relevant in the states with time of use pricing that encourage the consumers to use energy at certain times of the day to reduce strain on the grid. In this work we created a practical solution to appliance identification in a real household using frequency analysis on the aggregate electrical current waveform. We were able to identify the most important appliances to effectively manage household energy consumption.

I dedicate this work to my parents. Thank you for making me who I am.

I want to thank Patrick Mantey, Ali Adabi, Eric Cao, Alec Rein and David Bernick for their extensive help and support.

Chapter 1

Introduction

A popular new area of research that focuses on appliance identification is the Non Intrusive Load Monitoring or NILM. The concept was initially introduced in 1992[4]. The idea is to have a single sensor in the main electric panel and monitor overall power consumption of the household. One can break down the aggregate power of the household into power consumed by individual appliances.

Such a breakdown can be very useful to a consumer because currently consumers have no information about the power consumption of their appliances and therefore cannot make any educated decisions related to their power consumption patters. For example, an old refrigerator can be consuming power inefficiently where a newer refrigerator would be more economically viable over the long term.

The popular approach in the field of NILM is to use frequency analysis and look at power spectrum of the current and try to identify consistent patterns for different appliances. This is useful because an electric circuit can be viewed as a transfer function with sinusoidal input (60 Hz 120V AC) and periodic sinusoidal output which can be a combination of different harmonic frequencies (usually odd) of 60Hz, such as 120Hz, 180Hz, 240Hz, etc. The reason harmonics are mostly even

has to do with the fact that they are introduced by nonlinear loads (rectifiers) [20]. Full cycle rectifiers are more common and introduce odd harmonics. The feature of odd harmonics is that negative and positive cycles are symmetrical. Half-cycle rectifiers (less common) introduce even harmonics which make the waveform look asymmetrical in its positive and negative cycles. That is why many appliances with even harmonics can be very easy to identify due to their simple rarity.

The power spectrum of the appliance is closely connected to the appliance circuitry and thus presents a good candidate for a classification feature. For instance, the harmonic content of the current is closely connected to the power factor of the appliance. To illustrate the idea of how appliances can be identified by their power spectrum we will discuss a few common household appliances. A simple heating element (electric kettle, stove, iron) has a power factor of 1.0 and doesn't have any other harmonic frequencies only the 60Hz. In this case the current waveform follows the voltage waveform exactly without any phase shift. On the other hand, a refrigerator compressor is a relatively high power motor with a power factor of 0.7-0.8 and has a lot of harmonic frequencies which distort the perfect 60Hz waveform.

Chapter 2

Related Work

2.1 NILM

The initial idea of Non Intrusive Load Monitoring was introduced by Hart in 1992 [4]. The general idea was that the aggregate power consumption of the household is the sum of consummations of all devices currently running:

$$P = \sum_{i=0}^{n} P_i$$

i $\in 0, ..n$, where n is the number of currently active appliances.

The benefit of looking at the problem from the perspective of disaggregation is that it requires only a few sensors on the main panel to observe the whole house in contrast to installing a sensor on every appliance of interest.

The basic work process of NILM is defined as follows:

- Data Acquisition
- Event Detection
- Disaggregation

2.2 Hidden Markov Models Approach

There is a research group in Vancouver which heavily focuses on NILM and tries different approaches. One of the interesting ways to model an appliance and then classify it was to use Hidden Markov Models [8]. The model had decent results on test data sets.

2.3 Neural Networks

2.3.1 LSTM

Another (more recent) approach is based on LSTM neural networks [11]. LSTM stands for Long-Short-Term-Memory network and coupled with a recurrent neural network can be very powerful in classifying text and time-series data. We tried a similar approach by classifying a vector of power data coupled with a harmonic data and their respective standard deviations based on an idea that uses standard deviation and derivative of the signal to perform event detection [12]. The approach showed decent results from empirical tests but the problem was that more labeled data was required to build a robust model.

2.3.2 GAN

A very novel study (unpublished as of date when this thesis was written) explores applying convolutional GAN networks for energy disaggregation [7]. The authors of that work presented very convincing results for identifying the energy consumed by different devices. Their approach significantly (about 20% better) improves the results obtained by previous state of the art approach recently published that also uses convolutional neural networks for energy disaggragation [16].

It is worth noting, that this GAN paper presents results for identifying four devices while our approach is able to identify at least eleven devices. It would be very beneficial to combine GAN networks for energy disaggregation with a bank of filters approach that this study utilizes. In a most simple implementation, this would allow separation of nonlinear loads from linear (purely resistive) loads and further simplify the problem. The benefit of observing frequency domain signals is that frequency domain is agnostic to power consumption whereas the same device can have many different modes of operation (different cycles) as well as different power levels (microwave cooking modes, dishwasher modes, washer modes etc) while maintaining the same frequency footprint.

Additionally SEADS-Plug is capable of producing training data of power and harmonics at the rate of 16Hz, which is a considerable improvement over the rates of 1Hz and $\frac{1}{3}$Hz and $\frac{1}{6}$Hz which is what the training dataset in REDD and UK-DALE have respectively. It is probable that the GAN architecture that authors present will be able to improve its performance when its context-aware architecture is able to utilize context about motor transients, which is an important feature often overlooked that we discuss in Chapter 7.1.

2.4 Neural Network + Image Processing

A new approach in time series data recognition has been developed receently, based on the notion that an appliance power factor could be inferred from the graph of its voltage-current trajectory [5]. Some of the researchers have successfully applied this idea and obtained promising results [3]. The main difference here is that instead of using time-series data for classification, visual data is used - the plot of voltage vs current, which is correlated with the power factor and harmonics.

Thus it is "picture recognition", analagous to recognition of faces or other real objects in 2-dimensions.

Chapter 3

System Overview

3.1 Hardware

In order to get data for the electrical current we used current transformers with a frequency response up to 1000Hz coupled with a 12-bit ADC that supports maximum sampling frequency of 100,000 samples/seconds. We use a small linux box (SEADS) as our processing unit. The communication with the ADC was carried out through SPI protocol.

Since current transformers act as low pass filters there is no practical point in sampling at frequencies that greatly exceed 2kHz (twice the nyquist frequency). Current sampling rate ranges from 2.2kHz to 4kHz depending on the number of channels. SEADS can technically go to 8kHz and higher given that it currently supports the maximum of 7 channels (one for voltage and the rest for current).

3.2 FFT Approach

The initial approach was to use Fourier Transform on the signal, shift it to get the positive half and look at the absolute values for amplitudes at different frequencies. This is the standard approach for many signal processing applications. In our case we discovered that the use of FFT was severely hindered by the imperfection of our instrumentation which was hard to compensate for. For instance, our instrumentation didn't allow us to perform the FFTs on the CPU because the operation was too compute intensive for a small ARM processor that SEADS has. The only way out was to use the GPU library which only allowed FFTs of the length of powers of 2 (64, 128..., 4096, 8192, etc). It was hard to align our sampling frequency with the length of FFTs and even with zero-padding we experienced inconsistent results due to a large amount of "spectral leakage". We also tried applying different windows to the sinusoidal signal in hopes of getting better frequency resolution or amplitude accuracy such as Hanning[19] window and and Hann window[15]. Even though the windowing improved the results considerably for lower frequencies (60Hz-240Hz) higher frequency harmonics experienced a lot of noise due to their very small amplitudes in relation to the lower frequencies and still present spectral leakage. Our suspicion was that the higher frequiencies also get lost due to the nature of their small amplitudes. For instance, if the currents in 60Hz, 180Hz are measured in amperes as soon as we go beyond 300Hz we are looking into milliamperes and lower. We tried amplifying the signal in software but it didn't improve the results.

3.3 Filtering approach

We decided to try out a different approach which proved to be more consistent and gave us better data to work with throughout the whole frequency band. We implemented 6th-order Butterworth bandpass filters centered at each harmonic frequency with a width of 20Hz [10]. Such a width was empirically tested to not be too wide since the signal of interest only contains frequencies which are integral multiples of 60Hz. The reason for this is because rectifiers and other switching power control elements in non linear loads tend to draw power at the peaks of the AC waveform. That is why the output current is periodic in 60Hz [18]. After filtering the signal at the frequency of interest the energy of the signal is calculated. Signal energy is defined as:

$$E = \int_{-\infty}^{\infty} = |x(t)|^2$$

For the finite descrete signal this can be expressed as the summation:

$$E = \sum_{i=0}^{n} |x[i]|^2$$

where n is the length of the signal.

The energy of the signal bandpass filtered at a particular frequency is proportional to the absolute values of FFT but is not subjected to the spectral leakage. Spectral leakage is a big problem in high accuracy FFT applications. Size of the FFT should be exactly the same as the sampling frequency to get ideal frequency bin distribution (each Hz has a separate bin). The small inconvenience is that not all software libraries support use of FFTs that are not a power of 2 but this could be solved with padding. A bigger problem is that our signal doesn't follow the exact

sampling frequency due to the nature of the hardware we use. Our somewhat wide filter contributes to the stability of the filtered signal but at the same time efficiently provides the energy around the band of a specific harmonic and not of the other ones. Also the numerical effects in FFTs with a small number of cycles and the discontinuities at the beginning and end of the segment input to the FFT are not present in the filter band approach.

Filters do have some transients at the beginning until they initialize but with continuous sampling that effect only occurs at the initialization and doesn't occur because filtering is performed continuously whereas FFTs operate on partitions of the signal. We weren't able to implement FFTs on the continuous data because of the computational limitations of the SEADS platform.

After empirical testing in the field the filtering approach proved to be much more consistent and contain less noise than the FFT approach while providing us with the same frequency signatures that we identified using the FFT approach. However, there are still some artifacts and rare discontinuities in the waveform. These occur due to the non-perfect continuous sampling that due to the limitations of SEADS processor. But those discontinuities are clearly outliers, which are filtered out by a median filter.

After filtering each chunk of the waveform, filter initial conditions are updated with the final conditions from the previous filtering. Sixth-order filters were realized as a cascade of three second-order filters for improved stability and precision at less computational cost. Higher order filters were tested but didn't improve the results due to the nature of the waveform (no signals present except around the center of the passband). Therefore we didn't use any higher order filters as we saw no benefit for the added computational requirements.

3.4 Continuous sampling

One of the very big challenges that we faced was to implement continuous, real-time sampling with SEADS, running a non-real-time Linux OS. In order to achieve that we pipelined the software into several processes that handle different parts of data mining.

- The first is the sampling process written in C. This process uses BCM2835 (Broadcom Chip on our board) to perform high frequency communication over SPI protocol. No other programming language on that platform is able to achieve the speed that we desired (potentially 8kHz for 8 channels). This process samples data continuously in a loop and saves it to a rotating buffer. Every second (with an uncertainty of roughly 0.3 milliseconds due to the nature of the OS) the signal is written in a comma separate value format to a file. The writing to the file is carried out on a separate thread since it takes roughly 100ms on our system to perform writing. Our system has a CPU with 4 cores. In order to improve the continuity of our sampling process we created a different CPU set dedicated specifically to that process. CPU set is a concept in linux, which allows to create sets of tasks for specific cores - hence the naming. We moved all of the other processes and all movable kernel threads to the second CPU set so that sampling process experiences minimal interruptions. In addition to that we created a temporary ramdisk in RAM so that writing to file doesn't involve any IO. The reason for that is because the next process that we will discuss reads from these timestamped files and we don't want the IO to introduce any further uncertainty.

- The second process is written in Python. It reads chunks of the waveform sampled by the first process and applies filters at different harmonic fre-

quencies and then calculates the energy of the signal as well as power and voltage. The vector of these values (voltage, power and harmonic energies) is then saved to an SQL database.

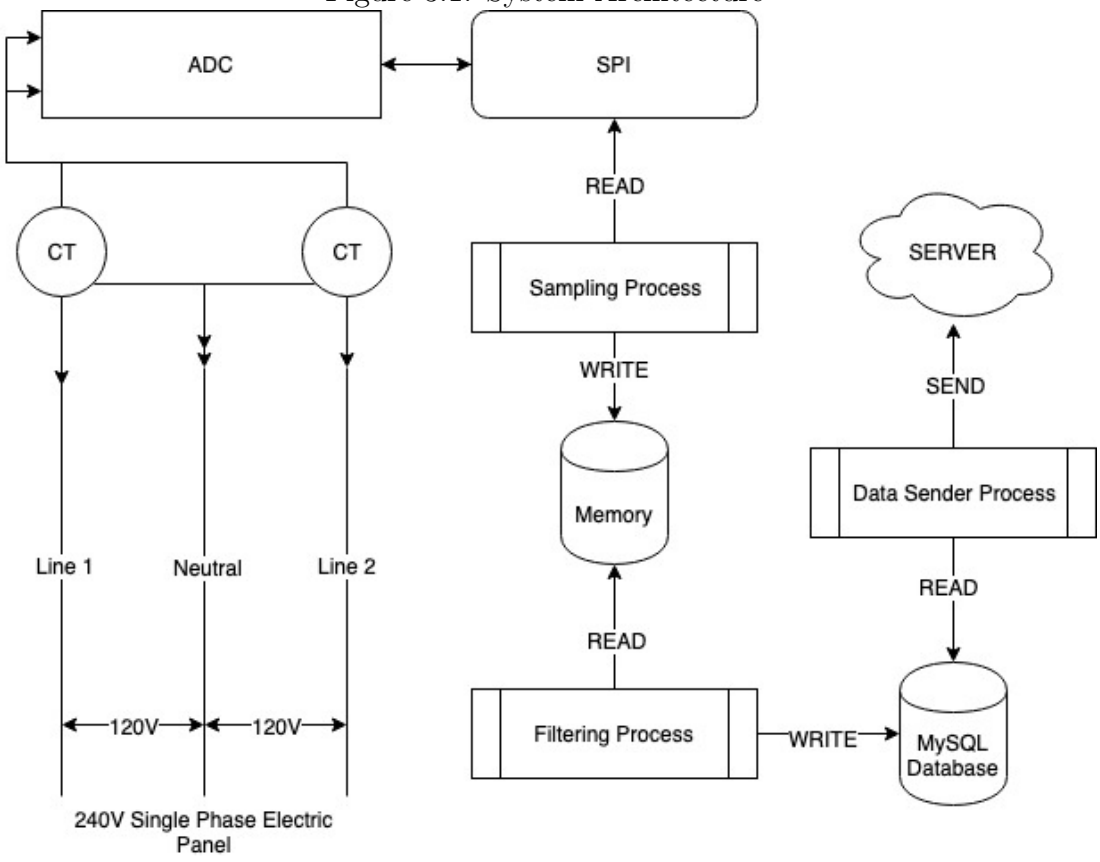
- The third process is also written in Python. It queries the database in bulk, packages the data, compresses the data using gzip and sends it to our server via HTTP POST request.

This architecture was dictated by the requirements of real-time sampling and continuous processing. First of all we use Python as much as possible because it has a rich library of scientific packages which allows us to quickly implement filters and data communication with our server. By breaking the whole process of data mining into three independent processes we are able to better utilize the multiple cores of the microprocessor in SEADS in an efficient way.
Departing from the FFT approach of taking a sample and then processing it to the continuous uninterrupted sampling, also lays much of the groundwork for further research that our group is interested in - detecting anomalies in the voltage signal - where the continuity of the waveform is essential for the meaningful analysis.

3.5 High Level Diagram

Both hardware and software mechanisms can be seen on the following system diagram:

Figure 3.1: System Architecture



Chapter 4

Experiments

4.1 Harmonics

We use the first sixteen harmonics to determine spectral behaviours of different appliances. Ali Adabi in his Ph.D. thesis here at UC Santa Cruz already demonstrated that with the first 15 harmonics one is able to classify appliances with 90% accuracy [1].

4.2 Power Calculation

The power and energy is measured on two 120V lines using a standard industry approach which is consistent with widely accepted methods [17]. The power of the sample is defined as:

$$P = \frac{1}{N} \sum_{j=0}^{N} e_j i_j$$

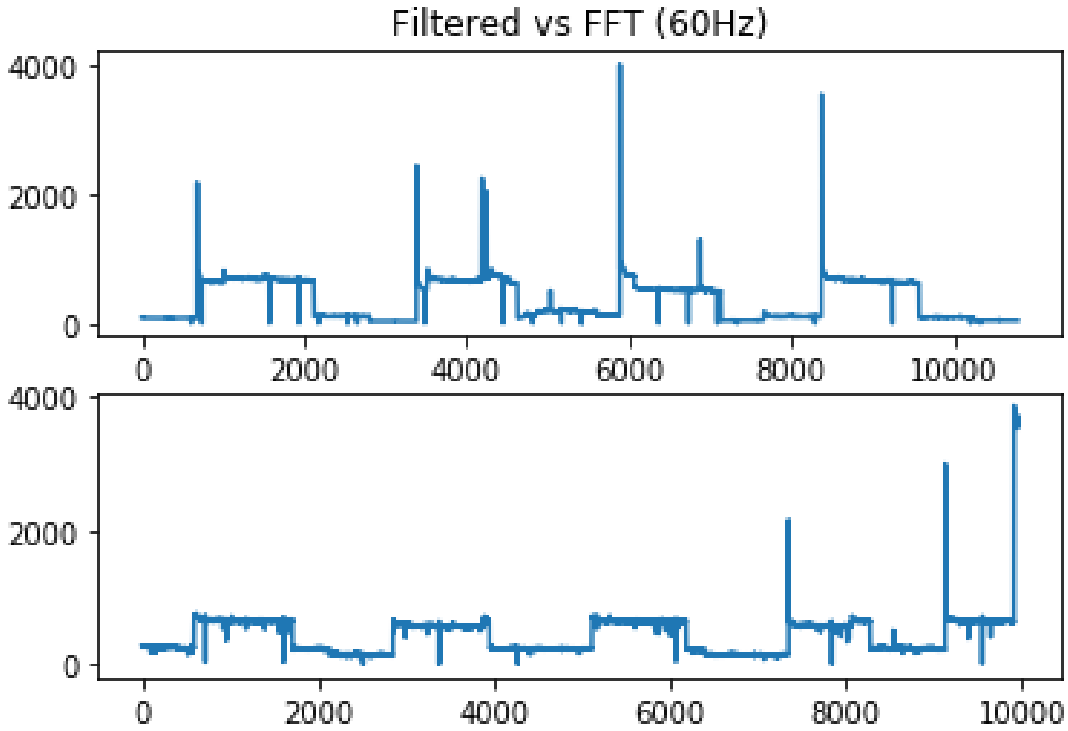
Where e_j and i_j are voltage and current samples respectively. Power is calculated over an integer number of voltage cycles. N is chosen to be an integer multiple of

the number of samples per cycle of the 60 Hz signal.

4.3 FFT vs Filters

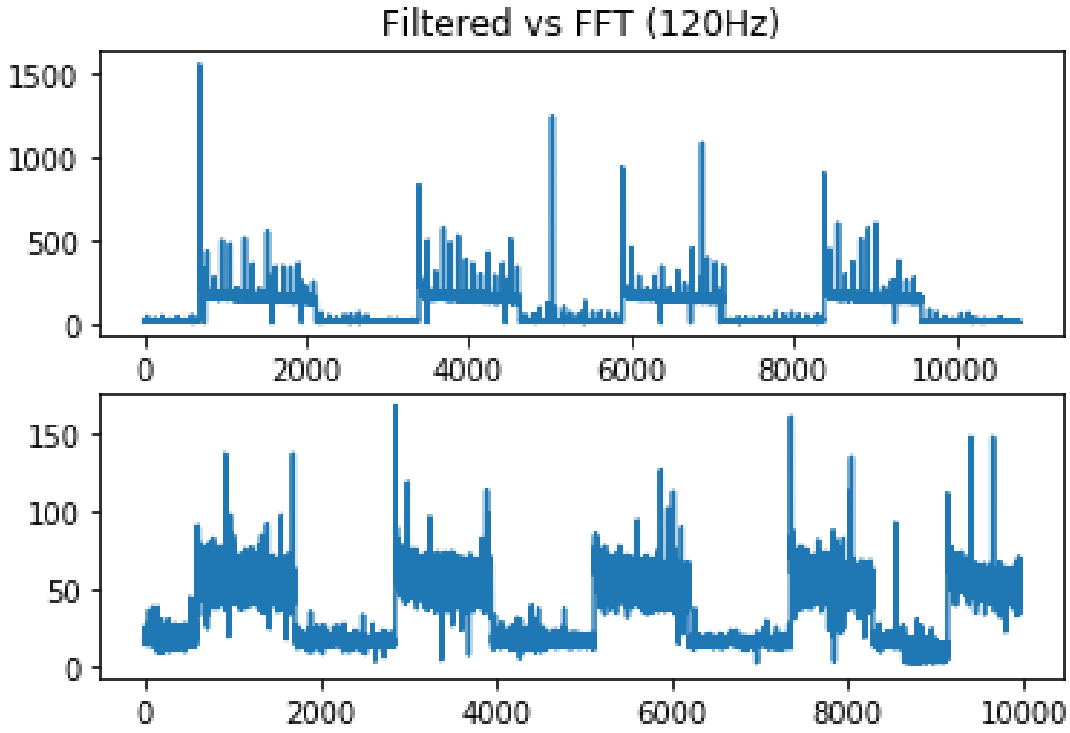
We mentioned above that we decided to use a series of bandpass filters instead of FFT analysis due to much better resolution. In order to demonstrate our motivation here we provide some waveforms of two refrigerators at different frequencies (the top graph always being the filtered waveform).

Figure 4.1: Comparison between refrigerator signatures using FFT and filters at 60Hz



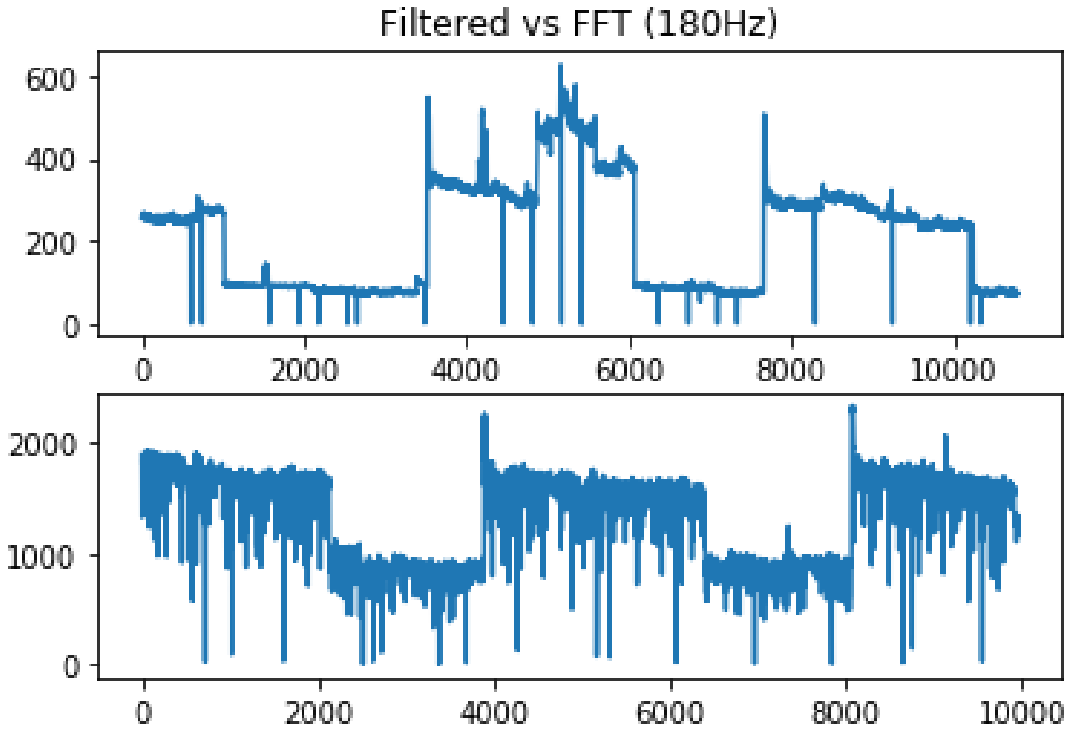
FFT signal is presented on the bottom. For 60Hz there is not much difference between the two. Both have occasional outliers artifacts that can easily be filtered by the median filter.

Figure 4.2: Comparison between refrigerator signatures using FFT and filters at 120Hz



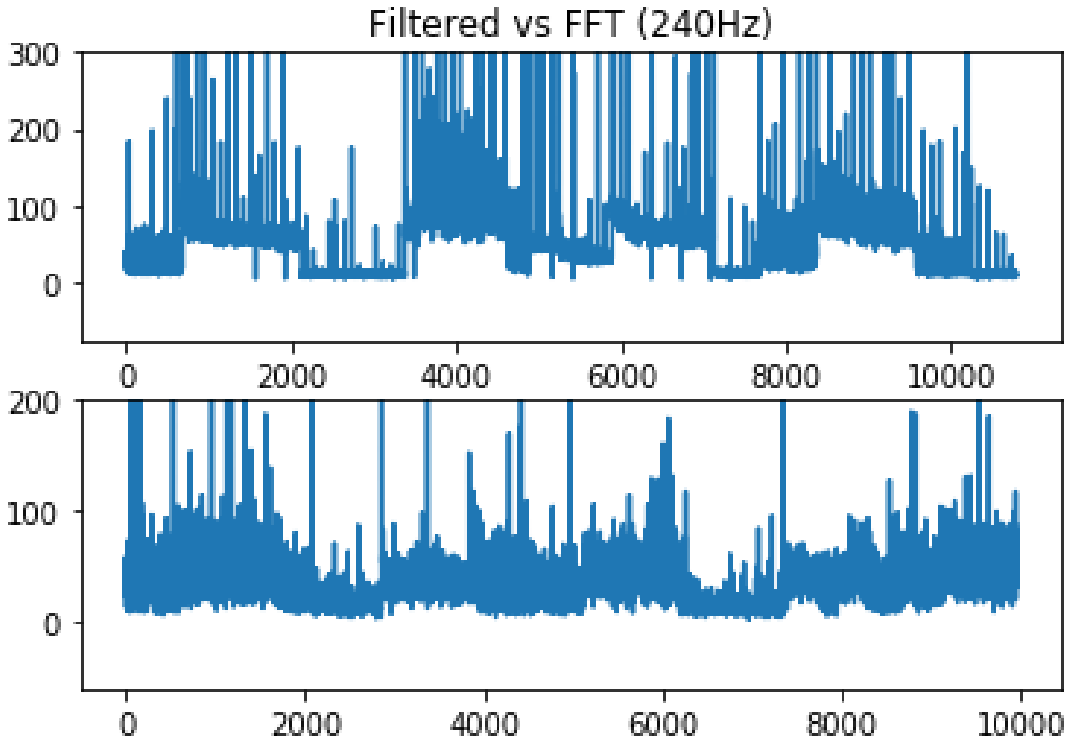
FFT signal is presented on the bottom. The 120Hz filtered waveforms do have some outliers (usually 1% of the data) due to some small interruptions in sampling which are picked up by the filter to look like even harmonic components (because they distor the symmetry of the waveform). One can already see that FFT data starts to deteriorate even though its output could be corrected by post filtering.

Figure 4.3: Comparison between refrigerator signatures using FFT and filters at 180Hz



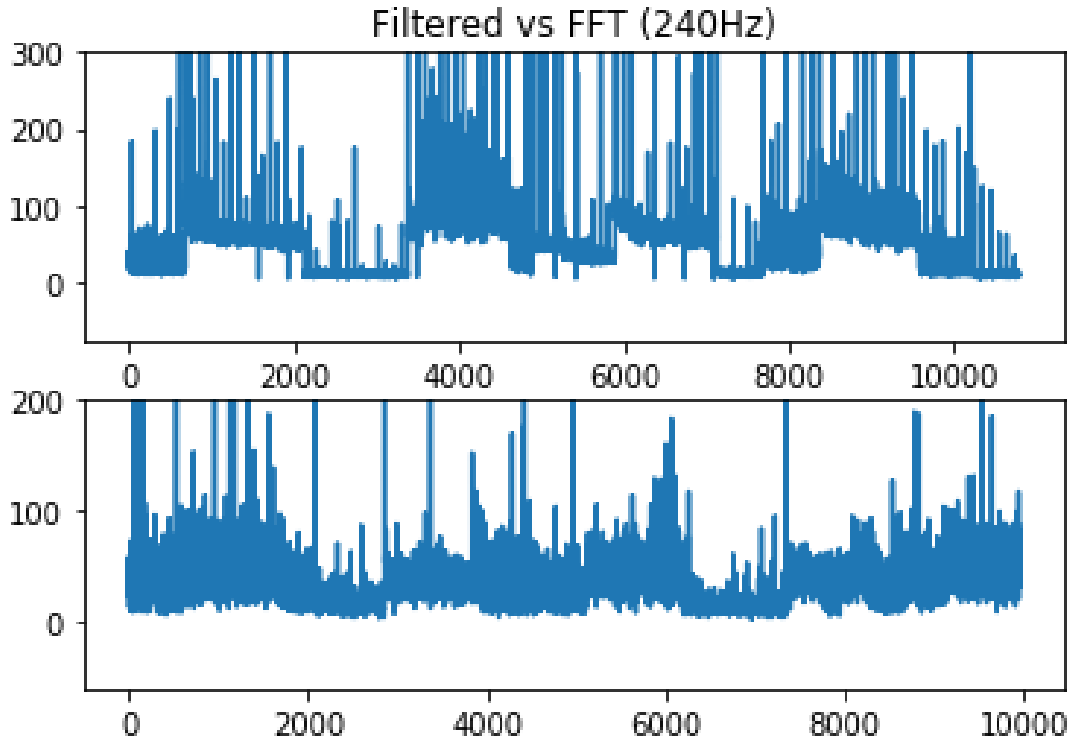
FFT signal is presented on the bottom. Here we see a very high deterioration of the FFT data compared to the filtered data. This is due to small amplitudes at these harmonics and spectral leakage.

Figure 4.4: Comparison between refrigerator signatures using FFT and filters at 240Hz



Here both approaches do not perform perfectly well but the filtering one is superior. The reason for so much noise is due to very low currents present in even harmonics. In theory perfect electrical appliance shouldn't have any even harmonics because they are unhealthy for appliance itself as well as the voltage source (the grid).

Figure 4.5: Comparison between refrigerator signatures using FFT and filters at 240Hz



Here we see an even further advantage of the filtering approach.

In the end we are able to nearly perfectly recover all frequencies in the range of our current transformers (0-1000Hz), while with the FFT approach the data becomes too noisy to recover after 360Hz. We do have a significant noise in the even harmonics but still less than FFT approach. The problem is mitigated by the fact that only one or two electrical appliances exhibit any behavior in even harmonics. The rest of the studied appliances have only odd harmonics. Even harmonics are mostly insignificant and hard to capture which is what we expected to observe due to the nature of most power control elements.

Chapter 5

Classification

5.1 Split Phase Power System

This study capitalizes substantially on the fact that devices in a typical household are all 120V. Some are 240V appliances (in fact major appliances we identified are 240V or on the second current leg). This has to do with the split phase power system which is very common in the households across the US. The basic idea is that there are two legs with 120V potential between each and a neutral but when one looks at the potential across the legs it is 240V. This configuration is illustrated in the system architecture Figure 3.1.

Ali Adabi for instance, used a term "spatial disaggregation" to refer to sensors on different electric panels which helps to isolate sets of devices. It is important to draw a distinction between spatial disaggregation and measuring two legs of 240V power system. The difference is that spatial disaggregation requires knowledge of wiring configuration of the household: how many panels, which devices on which panels, as well as which leg of the split-phase. Split phase power system is a more or less universal configuration[21] present in many households that allows 120V

and 240V depending on the device.

Some 120V appliances are on one of the two legs of the split phase, while 240 volt appliances have current on both. But these are superimposed on different backgrounds, those being the currents drawn by 120 volt. We also note that small loads, like LED lights, and even small appliancies like TV, PC power supplies, etc. do not contribute much to the overall power consumption over the long term. On both legs, one sees the (same) current being drawn by 240V appliances, and these are major loads. There is also a benefit of a clear baseline when it comes to the second phase because all of the "small" and irregular loads are typically not present on it.

5.2 Approach

Our approach relies mostly of static classification methods rather than using more complex methods such as neural networks. While neural networks can be impressively effective at many classification tasks we concluded that the limited amount of labeled data severely hinders their potential here. It is true that many people have obtained good identification accuracy on popular public labeled data([6][9][13]) sets but there isn't much work that generalizes to an average household.

We decided to focus on hand tailoring the features for devices that consume 95% of electricity on the panel that we measured.

5.3 Devices

- Refrigerator in the kitchen

- Old refrigerator in the garage
- Microwave
- Coffee maker and other resistive heating elements
- Dishwasher
- Washing machine
- Water pump
- Clothes dryer
- Oven
- Electric cooktop

5.4 Classification

In essence all of these appliances can be broken down into 4 groups:

- Purely resistive loads that run on 120V (coffee maker, iron, etc)
- Purely resistive loads that run on 240V (cooktop, oven, clothes dryer heating element and other large loads)
- Nonlinear loads that run on 120V (dishwasher motor, refrigerators, water pump, microwave)
- Nonlinear loads that run on 240V (dryer, oven with a fan motor)

5.5 Features

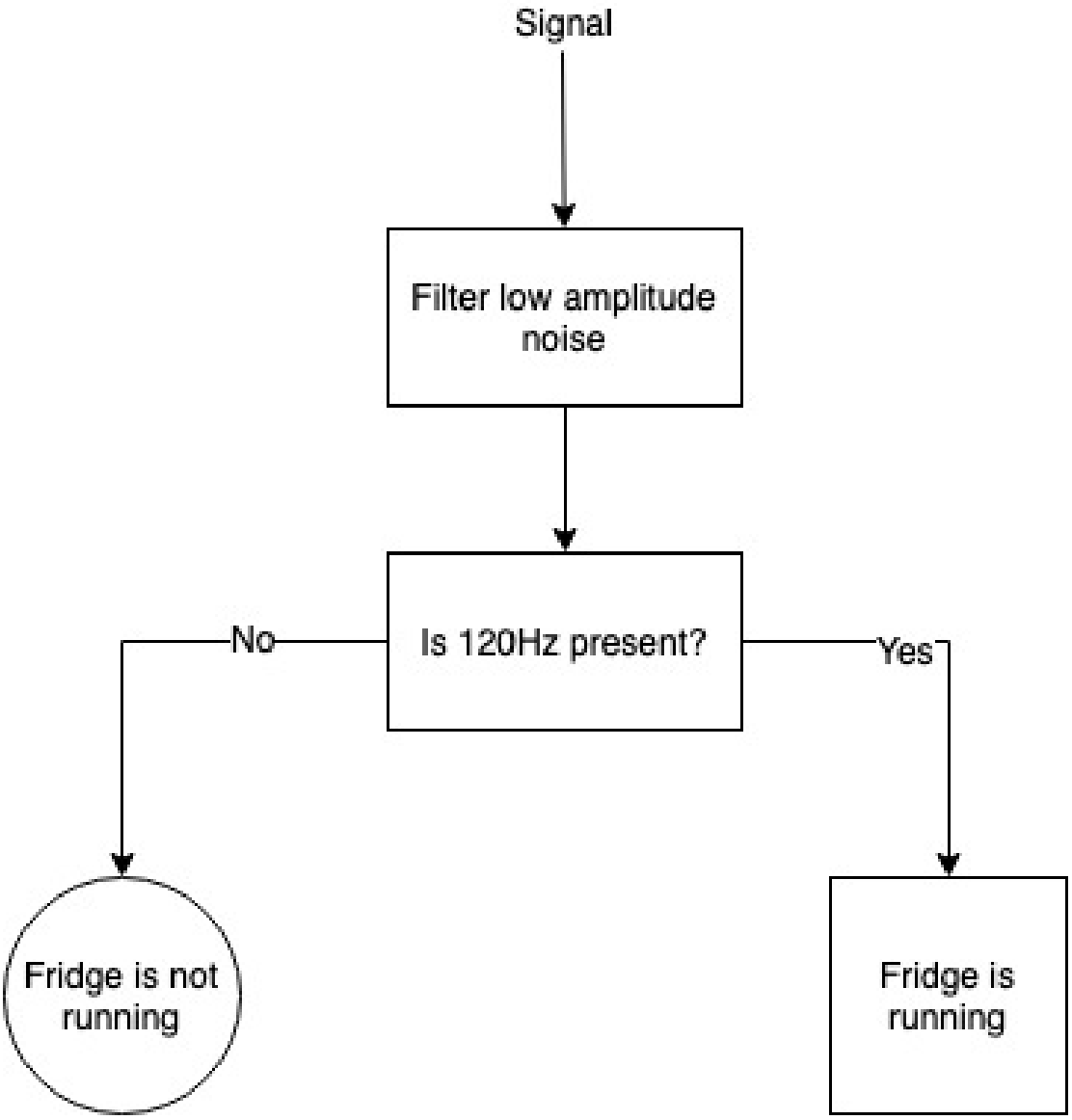
In our study for classifying devices we were able to identify most of the devices using the following feature set and a particular technique for each will be discussed in more detail.

- Unique Frequency
- Power
- Cycling behavior
- Phase (first, second, both)
- Running Time

5.5.1 Unique frequency

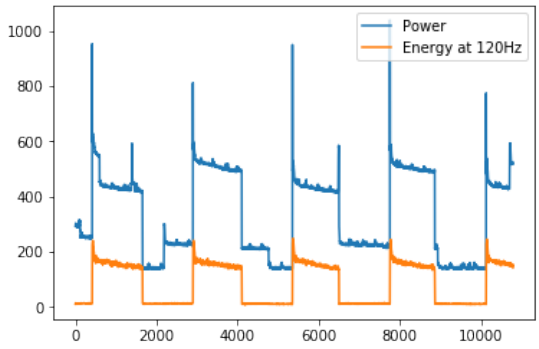
We call this approach "unique frequency" because only one particular device can be exhibiting a particular frequency that other devices do not have. The best examples of such devices that we identified in our test household are an older refrigerator in the garage and the dishwasher. Both have very prominent current signals in 120Hz - rather rare frequency for an electrical appliance that no other devices share. After some filtering of the signal the problem of the classification reduces to a simplest possible binary classifier illustrate in Figure 5.1

Figure 5.1: Simple binary classifier



In order to better illustrate the concept let us look at some power data with 120Hz energy shown in Figure 5.2

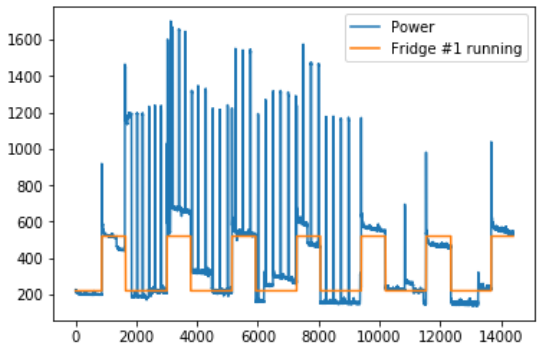
Figure 5.2: refrigerator power signature



As one can see there are two refrigerators running at the same time and only one of them has 120Hz energy in the signal. Characteristic of that 120Hz harmonics is that it makes the current sinusoid look slightly asymmetrical.

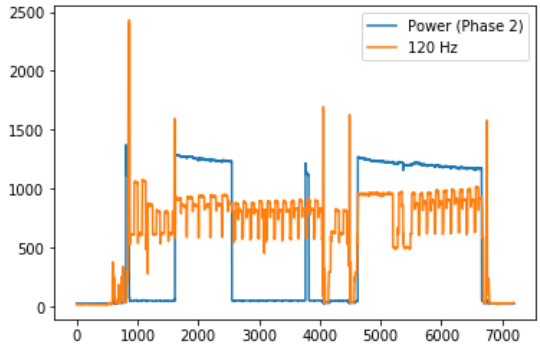
Applying the simple binary refrigerator classifier lets us easily find the running window of the refrigerator as demonstrated in Figure 5.3:

Figure 5.3: refrigerator classified



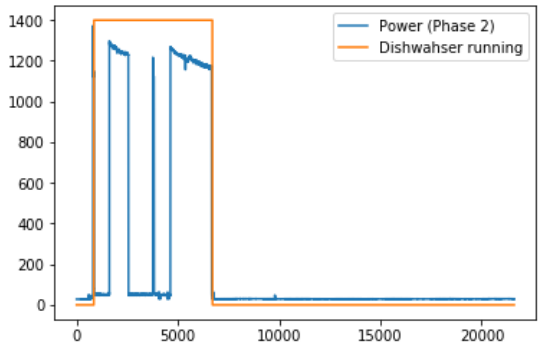
In this picture one can also see the coffee maker running with two refrigerators. The same could be done for the dishwasher running on the second phase in Figure 5.4:

Figure 5.4: Dishwasher power signature



After applying the binary classifier the exact running window can be found:

Figure 5.5: Dishwasher classified

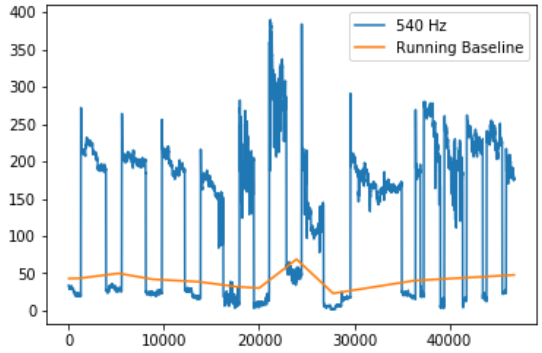


5.5.2 Prominent frequency with moving baseline

Unfortunately, not all devices have unique signatures that allow us identify them fairly easily. Some devices share all their freqeuncies with other devices thus their signatures overlap. The idea is to find a dominating frequency that distinguishes it from any other devices that have the same frequency. With 17 possible frequencies to monitor there is a good chance to identify such a frequency. In our classification we used that approach to identify the second refrigerator. The main distinctive feature of any refrigerator is a compressor that acts as a powerful inductive motor and easily overshadows other less powerful motors in the household.

This method is very similar in spirit to identifying a unique frequency except these are other appliances which exhibit that frequency though less prominently. These other appliances can be filtered out by the running baseline with some threshold. The demonstration of this concept is illustrated in 5 hours of data in Figure 5.6

Figure 5.6: refrigerator power signature



We identified 540Hz to be a good signature of refrigerator compressor, one can clearly see the huge jumps that correspond to refrigerator compressor actively running. However there are occasional "bumps" in the baseline when other lower power motors operate. The moving baseline with some thresholds lets us isolate the refrigerator fairly reliably. The moving baseline was calculated by finding the local minimums (these correspond to points at which the compressor turns off) and interpolating them across the energy signal to get a good baseline estimation for the refrigerator compressor.

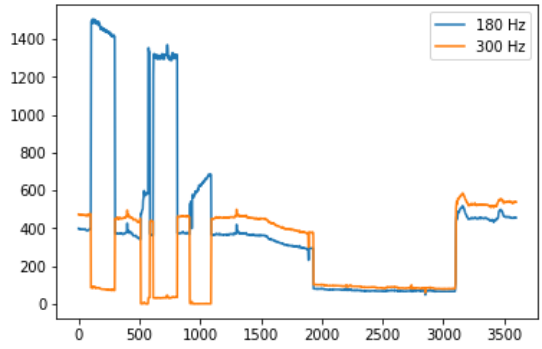
5.5.3 Inverse frequency

Another useful feature that we discovered is that for some devices power is proportional to certain frequencies, while being inversely proportional for some devices. In other words, when device "A" turns on the energy in frequency "X" increases while the energy in frequency "Y" decreases. Normalizing one frequency to an-

other allows us to subtract the two to be left out with only the running time of the device.

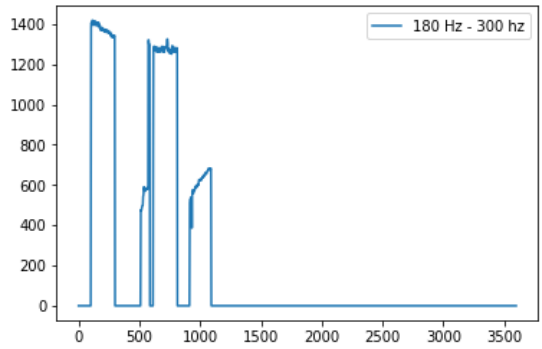
For instance, in the Figure 5.7 one can see the microwave at two fundamental frequencies (180Hz and 300Hz)

Figure 5.7: Microwave freqeuncy signature



As one can see, 180Hz energy is always smaller than 300Hz energy except when the microwave runs. Subtracting the two and applying a simple filter yields the following signal:

Figure 5.8: Difference between two energy signatures



We investigated this phenomena further by measuring both the microwave and the refrigerator running separately. Both devices had 180Hz and 300Hz frequencies present, albeit microwave had more energy in those frequencies due to it being a

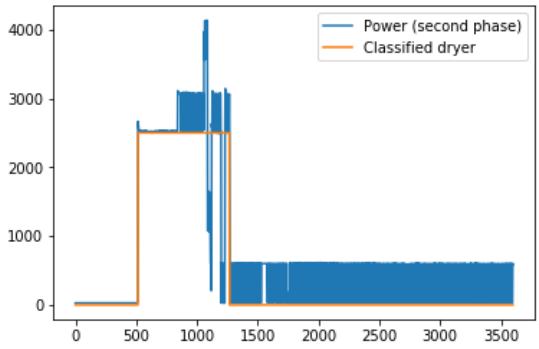
much more powerful appliance. But when these devices run in tandem the energy in 300Hz is less than when they ran separately. From an electrical point of view this could be explained by different phases of the harmonics. If 300Hz energy of the microwave is substantially out of phase (somewhere around 180 degrees) with 300Hz energy of the refrigerator the aggregate of the two will cancel out or get reduced. This is a useful and overlooked classification feature which we observed in many other devices but didn't pick for classification because other more prominent features were available. More study could be done on the relations of harmonics to other harmonics and their respective phase angles.

The classification is improved even further by applying a time filter of 30 seconds (microwave rarely gets run for less than that and is usually in multiples of 30 or 60 seconds).

5.5.4 Power classification

This method is useful in identifying very high power devices. For instance in our test household the most powerful device was an electric dryer with power consumption of 5kW (around 20Amps of clothes dryer). The dryer has a heating up phase and then runs in short cycles. By filtering out all the signals that consume less than 20A one can easily find the dryer (which draws the maximum current out of all devices). Classified clothes dryer power over time (followed by an electric cooktop) can be seen in Figure 5.9

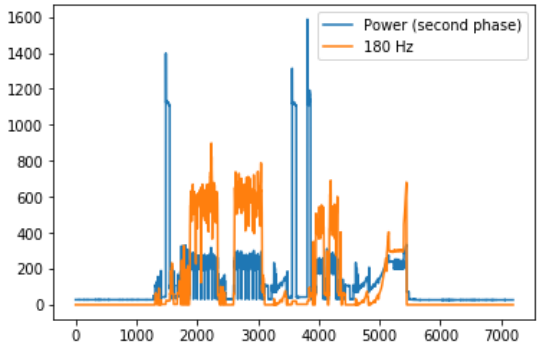
Figure 5.9: Classified Dryer



5.5.5 Running time

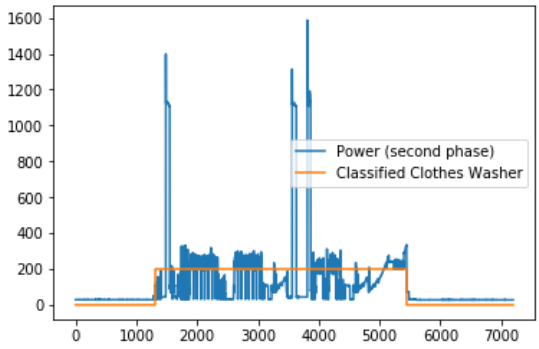
Some devices can be classified by a running time in addition to a very prominent frequency. For instance the clothes washer has the most powerful motor in the 180Hz range on the second phase that runs for long periods of time 20-60 minutes. For instance the signature of the clothes washer can be seen in Figure 5.10

Figure 5.10: Clothes washer freqeuncy signature



One can see two 30-40 minute loads ran one after the other (charecteristic water pump 1.4kW spike before the run). Applying a filter to remove low amplitude signals (such as water pump) and a timer filter to detect long running motors we are able to get the following classification results:

Figure 5.11: Classified clothes washer

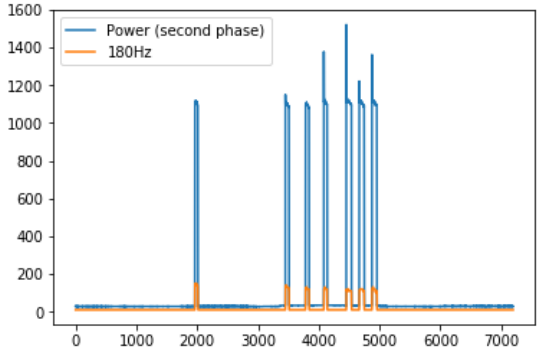


5.5.6 Lower-Upper Threshold classification

Certain devices exhibit very consistent frequency behavior and can easily be isolated by filtering out the signal by lower and upper bounds.

A good example is the water pump, which has a low power motor with very consistent energy in 180 Hz, that is easily overshadowed by powerful motors (clothes washer, dishwasher). The convenient part is that the signals that overshadow the water pump usually draw water and that activates the water pump in this house to maintain desired water pressure. So a dishwasher or clothes washer running imply that the water pump is running as well. One can see the amplified water pump signal in Figure 5.12

Figure 5.12: Water pump frequency signature



5.5.7 Moving standard deviation

An electric cooktop can appear to be a very challenging load to disaggregate. It is a purely resistive element with no harmonic frequencies to stand out. It is also highly volitile in its power consumption pattern: one small cooking surface can draw 400W while the bigger one can draw 1kW and any combination of several of them can vary in power quiet dramatically. However one distinctive feature that a cooktop has is a very frequently switching on/off cycle to keep the cooking surfaces hot. This can be identified by calculation of the standard deviation:

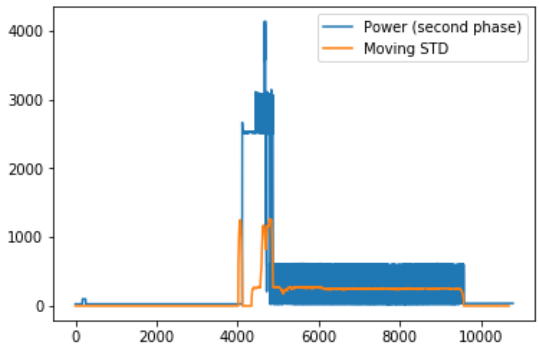
$$\delta = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

where μ is the mean of the signal.

The window of the standard deviation is determined empirically and is proportional to the duration of a device cycles. Shorter cycles have shorter windows. The value of standard deviation is proportional to the change in power during the switch on and off cycles so a good lower bound for filtering noise can be established.

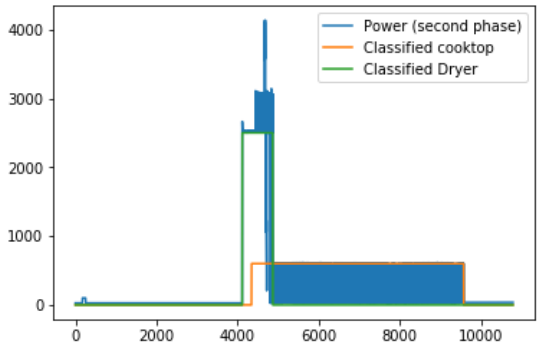
For instance Figure 5.9 shows a dryer followed by cooktop. Plotting the standard deviation with the power allows us to identify cooktop:

Figure 5.13: Cooktop standard deviation



Other devices have high standard deviations in certain running modes. A clothes washer and dryer can have many on/off cycles but can be identified much more reliably by power and harmonic content of the signal. So after finding signal with high standard deviation, one can filter out the running times for dryer and clothes washer to get better accuracy:

Figure 5.14: Cooktop classified



Identifying resistive loads

Many of the resistive loads share one characteristic behavior. They have three states: heating up, keeping hot and off. They are usually hard to identify because harmonic analysis approaches cannot be applied to them (resistive loads only have 60 Hz in the current waveform). However they can be classified by the durations of "keeping hot" cycles (usually 10-15 seconds) and power draw at these stages. For instance coffee maker make take several minutes to brew coffee but after it's done it keeps coffee hot for a long time by cycling through on/off (with long pauses). 1kW coffee maker will usually have 1kW short burst to keep things hot, similar for iron, cooktop and other devices. By finding a derivative of the power signal one can find the short bursts of a particular power pretty reliably.

Chapter 6

Results

6.1 Classifiers

As a result of this study we were able to identify a list of feature sets and create respective classifiers based on this features for major devices in the studied household. Some of the identified features were discovered accidentally (such as inverse effects with phase angles of the harmonics) but could potentially be useful inputs to Neural Networks and other approaches.

Table 6.1: Classifiers by features

Device:	UF:	PF:	IF:	Power:	Time:	STD:
refrigerator 1	120Hz	120Hz	n/a	n/a	n/a	NO
refrigerator 2	NO	540Hz	n/a	n/a	30-60 m	NO
Microwave	NO	180Hz	300Hz	1.5-2.0kW	0.5 m and longer	NO
Dishwasher	120Hz	120Hz	n/a	1.0-1.5kW	30-40 m	NO
Dryer	NO	n/a	n/a	2.5kW	30-60 m	Some modes
Clothes Washer	NO	180Hz	n/a	∼500W	20-60 m	Some modes
Water Pump	NO	180Hz	n/a	1.5kW	60-120 s	NO
Convection Oven	NO	780Hz	n/a	1.8kW	10-20 m cycles	No
Cooktop	NO	n/a	n/a	400W-1.5kW	10+ m	YES
Coffee Maker	NO	n/a	n/a	1kW	10-15 s to keep hot	NO

Table colums:

- UF - Unique Frequency
- PF - Prominent Frequency
- IF - Inverse Frequency
- m - minutes
- s - seconds
- STD - standard deviation
- Time - running time

6.2 Accuracy

The accuracy was measured in the following way. Hand labeled data for the last two weeks was taken and all classifiers were run on it. Then the results were compared against the labeled data:

Table 6.2: Classification Accuracy

Device:	Accuracy:
refrigerator 1	100%
refrigerator 2	80%
Microwave	100%
Dishwasher	100%
Dryer	100%
Clothes Washer	100%
Water Pump	90%
Convection Oven	100%
Cooktop	100%
Coffee Maker	100%

6.3 Discussion

Overall we were able to achieve great accuracy results. The main reason for such high accuracy values is the fact that we created hand-tailored classifiers for each appliance that we classified that made the approach less generalizable but very accurate. However, we think that the classification features and approaches used in this study can be useful for more generic approaches (such as neural networks) when large amounts of labeled data are available.

We looked at two distinct current legs of a split-phase system with high power appliances on the second leg and lower power appliances on the first leg. This let us split the appliances into two camps and made the data much cleaner. Other studies seem to look at the aggregate of both legs, SEADS allows to separate and unintrusively look at different current legs of multiple panels which applies to many household configurations.

We do not identify the exact power amounts consumed by each device but rather identify the running windows of each device which makes the problem simpler. The rough power consumption for each device can be estimated the following way: if one found the run time for each cycle, and the the power consumed by the device during each cycle is know, then the total consumption can be computed.

Chapter 7

Future Work

7.1 Transients

One feature that stayed somewhat ignored in this work is the power surge when motors turn on. These tend to stay fairly consistent and therefore can be used to identify different motors precisely. The problem is that the transients only lasts fractions of a second. These "spikes" have large high-frequency content and their identification requires analysis of this content. We only tried with 1Hz frequency for measuring power so the transients get averaged out with the stable state. We just started experimenting with higher power calculation frequencies (8-16Hz) and transients are captured much better but more testing is required. Even though transients provide much of insight into the power and size of the motor, harmonic frequencies on different phases allow us to capture all motors of interest with a good accuracy without transients as indicated in the above section.

7.2 Better sampling technique

We do realize that our sampling technique is not ideal. In order to achieve much better continuity the standard approach is to have a separate "slave" microcontroller which would only perform the sampling and save data to a buffer (an Arduino board or anything similiar could be a good candidate for the job). The main processing unit can then read chunks of the continuous waveform and process it using the convenience of linux operating system. However, in our case the continuouty and filter behavior proved to be satisfactory to be able to identify important and significant loads reliably.

7.3 Higher frequency sampling

One of the limitations that we faced is the use of current transformers. Current transformers act as low pass filters and rarely work in higher frequency ranges that could potentially contain important and unique features for other electrical appliances. For instance hall effect sensors can easily go to 20kHz[14].

Chapter 8

Conclusion

We developed and implemented a system which works in practice on a generic household with a variety of different electrical appliances, that include both non-linear and linear purely resistive loads of different characteristics. We identified features and implemented algorithms that can automatically detect these features to identify these appliances. These features are very simple and do not require large training sets of labeled data that many neural network approaches do. We also created an auxiliary system to record data of individual appliances that can also be used to polish the classifier for new appliances. This system can help both consumers and utility companies. Consumers will be more educated about their energy usage patterns and can schedule certain big loads at different times of the day to reduce energy costs from utility companies. Utility companies would be able to better manage the demand of the grid as more consumers would be able to switch their behaviors to accommodate the grid.

Appendix A

Firmware Code

This is used to isolate two cores from any system tasks

```
sudo cset shield -c 1-2
sudo cset shield -k on
```

This is used to run the sampling service using the "shielded" cores

```
[Unit]
Description=Sampling Service
After=multi-user.target

[Service]
User=root
Type=idle
Restart=always
WorkingDirectory=/home/pi/FFT-Harmonic-Extraction/fft
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=99
ExecStart=/usr/bin/cset shield -e
```

```
    /home/pi/FFT-Harmonic-Extraction/fft/fft_sampling

[Install]

WantedBy=multi-user.target
```

---

in etc/fstab we created a ramdisk to save sampling results to RAM to minimize I/O Read/Writes

---

```
tmpfs /var/sample_tmp tmpfs nodev,nosuid,size=10M 0 0
```

Appendix B

Sampling Code

This function is used to save sampled waveform to a file

```
void *save_data_to_file(void* argument) {
    int *data = (int*)argument;
    // Creating filename
    char buf[100];
    int timestamp = (int)time(NULL);
    snprintf(buf, sizeof(buf), "/var/sample_tmp/%d-buffer.csv",
        timestamp);
    // Counting length
    int length = 0;
    for (int i = 0; ;i++) {
        int value = data[i];
   if (value < 0) {
       length = i;
       break;
   }
    }
```

```
    // Writing to file
    FILE *f = fopen(buf,"w");
    for (int i = 0; i < length; i++) {
   fprintf(f,"%d,\n",data[i]);
    }
    fclose(f);
    int max_size = 1 << SIZE_LIMIT;
    for (int i = 0; i < max_size; i++) {
   data[i]=0;
    }
    return NULL;
}
```

This function samples the waveform and saves it to file

```
# sample_data function.

# channels - binary channel vector that specifies which channels to
    sample.
# For example, if we want sample channel 0 only:
# channels = [1,0,0,0,0,0,0,0]
# while if we wanted to sample channel 0 and channel 5:
# channels = [1,0,0,0,0,1,0,0]
sample *sample_data() {
   int fd=1;
   fcntl(stdout, F_SETFL, O_NONBLOCK);

   sample *s = sample_allocate();
```

```
 clock_t start = clock();
 float diff = 0.0;
int i = 0;
int data;
int index = 0;
int j = 0;
pthread_t my_thread;
while (1) {
    if (diff >= 1.0) {
        s->channel_data[index][i] = -1;
       int *data = s->channel_data[index];
       pthread_create(&my_thread, NULL, save_data_to_file, data);
       pthread_detach(my_thread);
       index = (index + 1) % 8;
       diff  = 0.0;
       start = clock();
       i=0;
    }
    data = sample_channel(0);
    s->channel_data[index][i] = data;
    i++;
    data = sample_channel(1);
    s->channel_data[index][i] = data;
    i++;
    data = sample_channel(3);
    s->channel_data[index][i] = data;
    i++;
```

```
        data = sample_channel(4);
        s->channel_data[index][i] = data;
        i++;
         clock_t current = clock();
         diff = ((float)(current - start) / CLOCKS_PER_SEC ); // Update
             time
    }


    sample_deallocate(s);
}
```

Appendix C

Filtering Code

```
import glob
import csv
import time
import math
import os
import array
import struct
import gc
import zlib
import requests
import json
import MySQLdb
import pandas as pd
import numpy as np
from scipy.signal import butter, lfilter, sosfilt, cheby2, sosfilt_zi,
    sosfiltfilt, resample
```

```
db = MySQLdb.connect(host="localhost", # your host
                     user="root",      # username
                     passwd="password",   # password
                     db="seads")  # name of the database
cur = db.cursor()



def butter_bandpass_filter(lowcut, highcut, fs, order):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    sos = butter(order, [low, high], btype='band',output='sos')
    zi = sosfilt_zi(sos)
    return sos,zi

def read_csv(filename):
    data = []
    with open(filename) as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')
        for row in spamreader:
            value = int(row[0])
            data.append(value)
    return np.array(data)

def create_filters():
    fs=3550
    filters = []
```

```
    initial_conditions = []
    harmonic_frequencies = [60, 120, 180, 240, 300, 360, 420, 480, 540,
        600, 660, 720, 780, 840, 900, 960]
    for harmonic in harmonic_frequencies:
        lowcut = harmonic-10
        highcut = harmonic+10
        filter, zi = butter_bandpass_filter(lowcut, highcut, fs, order=6)
        filters.append(filter)
        initial_conditions.append(zi)
    return filters, initial_conditions


def find_first_peak(voltage):
    index = 0
    max = voltage[index]
    for i in range(0,50):
        if voltage[i] > max:
            max = voltage[i]
            index = i
    return index


def find_last_peak(voltage):
    length = len(voltage)
    index = length-50
    max = voltage[index]
    for i in range(length-50,length):
        if voltage[i] > max:
            max = voltage[i]
```

```
            index = i
    return index


scale_factors = [60000.0, 100.0, 10000.0, 10, 5000.0, 1.0, 1000.0, 1.0,
    1000.0, 1.0, 100.0, 0.1, 10.0, 0.1, 10.0, 0.1, 10.0]
directory = "/var/sample_tmp/"
power_correction_factor = 0.00456601002818
filters1, ics1= create_filters()
filters2, ics2= create_filters()
filters3, ics3= create_filters()

filters_for_channels = [filters1, filters2, filters3]
ics_for_channels = [ics1, ics2, ics3]



def calculate_power(voltage, current):
    first_peak = find_first_peak(voltage)
    last_peak = find_last_peak(voltage)
    length = last_peak - first_peak
    sum = 0.0
    for i in range(first_peak, last_peak):
        c_value = current[i]
        v_value = voltage[i]
        power = c_value*v_value
        sum += power
```

```
    average_power = sum / length * power_correction_factor
    return abs(average_power)

def calculate_energy(waveform):
    sum = 0.0
    length = len(waveform)
    for data in waveform:
        sum += data*data
    return sum

def get_correction_factor():
    adc_max_value = 4096.0;
    referential_voltage = 3.282;
    ct_conversion_factor = 60.0600600601;
    voltage_correction_factor = 0.09487951807;
    current_conversion_factor = referential_voltage * 2 / 1.4142135624;
    current_factor =
        (referential_voltage*ct_conversion_factor)/adc_max_value
    total_factor = current_factor*voltage_correction_factor
    return total_factor

def compute_harmonic_energy(fs, window,current_waveform):
    harmonic_frequencies = [60, 120, 180, 240, 300, 360, 420, 480, 540,
        600, 660, 720, 780, 840, 900, 960]
    harmonic_data = {}
    for harmonic in harmonic_frequencies:
        lowcut = harmonic-10
```

```
        highcut = harmonic+10
        y = butter_bandpass_filter(current_waveform, lowcut, highcut,
            fs, order=6)
        energy_data = calculate_energy_chunks(window,y)
        harmonic_data[harmonic] = energy_data
    return harmonic_data


def get_all_csv_files():
    filenames = []
    os.chdir(directory)
    for filename in glob.glob("*.csv"):
        filenames.append(filename)
    filenames.sort()
    return filenames




def get_data():
    filenames= get_all_csv_files()
    if not filenames:
        return None, None

    oldest_file = filenames[0]
    #print("Oldest file: {}".format(oldest_file))

    parts = oldest_file.split('-')
```

```
    timestamp = int(parts[0])
    current_timestamp = int(time.time())

    diff = abs(current_timestamp-timestamp)
    #print("Diff: {}".format(diff))
    if diff < 4:
        return None, None

    channels = [[],[],[],[],[]]

    try:
        csv_data = read_csv(oldest_file)
    except:
        print("read_csv error")
        return None,None

    i = 0

    for data in csv_data:
        index = i % 4
        channels[index].append(data)
        i+=1

    os.remove(oldest_file)
    return channels, timestamp

def filter_waveform(waveform, sos, ic=None):
```

```
    x = np.array(waveform)
    y, zo = sosfilt(sos, x, zi=ic)
    return y, zo

def get_hex_data(array):
    python_array = []
    for value in array:
        int_value = int(math.ceil(value))
        if int_value < 0 or int_value > 32700:
            int_value = 0
        python_array.append(int_value)

    b = bytes()
    b = b.join((struct.pack('!h', val) for val in python_array))
    hex_string = b.encode('hex')
    #print("Original")
    #print(hex_string)
    compressed_data = zlib.compress(hex_string)
    compressed_hex = compressed_data.encode('hex')
    return compressed_hex

def save_data(hex_data, timestamp):
    device_id = 'FILTER'
    try:
        sql = "INSERT INTO SeadsData (device_id, unix_timestamp, data)
            VALUES ('{id}', {timestamp}, '{data}')".format(id=device_id,
            timestamp=timestamp, data=hex_data)
```

```
            cur.execute(sql)
            db.commit()
        except (MySQLdb.Error, MySQLdb.Warning) as e:
            print('Database error: {}'.format(e))

def calculate_energy_for_channel(current, filters, ics):
    energy_data = []
    for i in range(0, len(filters)):
        filt = filters[i]
        ic = ics[i]

        filtered,zo = filter_waveform(current, filt, ic)
        ics[i]=zo
        factor = scale_factors[i]
        energy = calculate_energy(filtered)/factor
        #print("Energy at {} Hz is {}".format((i+1)*60, energy))
        energy_data.append(energy)
    return energy_data

while True:
    channels, timestamp = get_data()
    if not channels:
        #print("No Data to read")
        time.sleep(1)
        continue
    current_channels = [np.array(channels[1]), np.array(channels[2]),
        np.array(channels[3])]
```

```
voltage = np.array(channels[0]) - 2048
current_channels[0] = (current_channels[0] - 2074)
current_channels[1] = (current_channels[1] - 2065) * 3.75
current_channels[2] = (current_channels[2] - 2058) * 3.75
power_data = []
if len(voltage) < 3000:
    for i in range(0,4):
        channels[i] = resample(channels[i], 3550)
    print("Incorrect length! Resampling")
power_1 = calculate_power(voltage, current_channels[0])
power_2 = calculate_power(voltage, current_channels[1])
power_3 = calculate_power(voltage, current_channels[2])
#print("Power1 is {}".format(power_1))
#print("Power2 is {}".format(power_2))
#print("Power3 is {}".format(power_3))

power_energy_data = [power_1, power_2, power_3]

for i in range(0, 3):
    current = current_channels[i]
    filters = filters_for_channels[i]
    ics = ics_for_channels[i]
    energy_data = calculate_energy_for_channel(current, filters, ics)
    power_energy_data = power_energy_data + energy_data

hex_data = get_hex_data(power_energy_data)
save_data(hex_data, timestamp)
```

Appendix D

Classification Code

```
# Where f is 120Hz energy signal
def identify_fridge_1(f):
    z = np.zeros(len(f))
    for i in range(0, len(f)):
        data = f[i]
        if data > 100:
            z[i] = 1
    return z
```

```
# Where f is medfilted 540Hz energy signal
def identify_fridge_2(f):
    max = np.max(f) + 1;
    f_inv = (f - max) * -1;
    peaks2 = find_peaks(f_inv, prominence=100)[0]

    x = np.linspace(start = 0, stop = len(f), num=len(f))
    threshold = np.interp(x, peaks2, f[peaks2]+20)
```

```
    z = np.zeros(len(f))

    for i in range(0, len(f)):
        data = f[i]
        t = threshold[i]

        if data > t:
            z[i] = 1

    return z, threshold
```

```
def identify_microwave(h_180, h_300):
    filtered_signal = h_180 - h_300
    filtered_signal[filtered_signal < 100] = 0

    z = np.zeros(len(filtered_signal))

    intervals = np.where(filtered_signal == 0)[0]

    for i in range(0, len(intervals)-1):
        curr_index = intervals[i]
        next_index = intervals[i+1]

        interval_length = next_index - curr_index

        if interval_length > 30:
```

```
            z[curr_index:next_index] = 1
    return z
```

```
# Dishwasher has very pronounced 120Hz harmonics
def identify_dishwasher(h_120):
    filtered_signal = medfilt(h_120, 1501)

    z = np.zeros(len(filtered_signal))

    for i in range(0, len(z)):
        curr = filtered_signal[i]

        if curr > 500:
            z[i] = 1
    return z, filtered_signal
```

```
# Where p2 is the power singal in the second leg of 120V
def identify_dryer(p2):
    power_candidates = np.zeros(len(p2))
    z = np.zeros(len(p2))

    for i in range(0, len(power_candidates)):
        curr = p2[i]

        # Look for all the loads that draw ~ 20 amps
        if curr > 2400:
            power_candidates[i] = 1
```

```
    nz = np.nonzero(power_candidates)[0]

    for i in range(0, len(nz)-1):
        curr_index = nz[i]
        next_index = nz[i+1]

        diff = next_index - curr_index

        if diff < 120:
            z[curr_index:next_index] = 1
    return z, power_candidates
```

```
def identify_clothes_washer(h_180):
    filtered_signal = h_180
    filtered_signal[filtered_signal < 20] = 0

    z = np.zeros(len(filtered_signal))

    nz = np.nonzero(filtered_signal)[0]

    for i in range(0, len(nz)-1):
        curr_index = nz[i]
        next_index = nz[i+1]

        diff = next_index - curr_index
```

```
        if diff < 300:
            z[curr_index:next_index] = 1

    intervals = np.where(z == 0)[0]

    for i in range(0, len(intervals)-1):
        curr_index = intervals[i]
        next_index = intervals[i+1]

        interval_length = next_index - curr_index

        if interval_length < 1200:
            z[curr_index:next_index] = 0
        if interval_length > 4800:
            z[curr_index:next_index] = 0
    return z
```

```
def identify_water_pump(h_180):
    filtered_signal = h_180

    filtered_signal[filtered_signal < 10] = 0
    filtered_signal[filtered_signal > 17] = 0

    z = np.zeros(len(filtered_signal))

    for i in range(0, len(z)):
        curr = filtered_signal[i]
```

```
        if curr > 0:
            z[i] = 1
    return z
```

```
# 540, 660, 780 Hz is a good candidate for classification
def identify_convection_oven(h_780):
    filtered_signal = h_780

    filtered_signal[filtered_signal<200] = 0

    z = np.zeros(len(filtered_signal))

    intervals = np.where(filtered_signal == 0)[0]

    for i in range(0, len(intervals)-1):
        curr_index = intervals[i]
        next_index = intervals[i+1]

        interval_length = next_index - curr_index

        if interval_length > 200:
            z[curr_index:next_index] = 1
    return z
```

```
# Identify cooktop by moving standard deviation

# Need to filter our dryer, washing machine
```

```
def identify_cooktop(p2):
    moving_std = np.array([power_2[i:i+100].std() for i in
        range(len(power_2)-100)])
    moving_std[moving_std < 70] = 0

    intervals = np.where(moving_std == 0)[0]

    z = np.zeros(len(p2))

    for i in range(0, len(intervals)-1):
        curr_index = intervals[i]
        next_index = intervals[i+1]

        interval_length = next_index - curr_index

        if interval_length < 1200:
            z[curr_index:next_index] = 0
        else:
            z[curr_index:next_index] = 1
    return z

def filter_out_dryer_and_washer(cooktop, dryer, washer):
    cooktop = cooktop - dryer
    cooktop = cooktop - washer
    cooktop[cooktop < 0] = 0
```

```
    intervals = np.where(cooktop == 0)[0]

    for i in range(0, len(intervals)-1):
        curr_index = intervals[i]
        next_index = intervals[i+1]

        interval_length = next_index - curr_index

        if interval_length < 1200:
            cooktop[curr_index:next_index] = 0
    return cooktop
```

```
def identify_iron(power_1, power_2):
    filtered_240_v = filter_240_v(power_1, power_2)
    difference = get_difference(filtered_240_v, 3)
    z = identify_appliance(15, 1400, 1500, difference)
    return z, difference

def identify_coffee_maker(power_1, power_2):
    filtered_240_v = filter_240_v(power_1, power_2)
    difference = get_difference(filtered_240_v, 3)
    z = identify_appliance(15, 900, 1100, difference)
    return z, difference

def get_difference(points, W):
    length = int(len(points))
    diff = np.zeros(length)
```

```
    for i in range(0, length-W):
        first = points[i]
        second = points[i+W]
        d = second - first
        diff[i] = d
    return diff

def filter_240_v(power1,power2):
    z = np.zeros(len(power1))

    for i in range(0, len(z)):
        p1 = power1[i]
        p2 = power2[i]

        if p2 < p1:
            z[i] = p1 - p2
        else:
            z[i] = p1
    return z

def identify_appliance(cycle_width, lower_threshold, upper_threshold,
    difference_signal):
    filtered_signal = difference_signal
    filtered_signal[abs(filtered_signal) < lower_threshold] = 0
    filtered_signal[abs(filtered_signal) > upper_threshold] = 0

    z = np.zeros(len(filtered_signal))
```

```
    nz = np.nonzero(filtered_signal)[0]

    for i in range(0, len(nz)-1):
        curr_index = nz[i]
        next_index = nz[i+1]

        interval_length = next_index - curr_index

        if interval_length == 1:
            continue
        else:
            current_delta = filtered_signal[curr_index]
            next_delta = filtered_signal[next_index]

            if current_delta > 0 and next_delta < 0:
                if interval_length < 15:
                    z[curr_index:next_index] = 1
    return z
```

Bibliography

[1] Ali Adabi, Pavlo Manovi, and Patrick Mantey. Seads: A modifiable platform for real time monitoring of residential appliance energy consumption. In *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, pages 1–4. IEEE, 2015.

[2] Ali Adabi, Pavlo Manovi, and Patrick Mantey. Cost-effective instrumentation via nilm to support a residential energy management system. In *2016 IEEE International Conference on Consumer Electronics (ICCE)*, pages 107–110. IEEE, 2016.

[3] Liang Du, Dawei He, Ronald G Harley, and Thomas G Habetler. Electric load classification by binary voltage–current trajectory mapping. *IEEE Transactions on Smart Grid*, 7(1):358–365, 2015.

[4] George William Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, 1992.

[5] Taha Hassan, Fahad Javed, and Naveed Arshad. An empirical investigation of vi trajectory based load signatures for non-intrusive load monitoring. *IEEE Transactions on Smart Grid*, 5(2):870–878, 2013.

[6] J Zico Kolter and Matthew J Johnson. Redd: A public data set for energy disaggregation research. In *Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA*, volume 25, pages 59–62, 2011.

[7] Chen Kunjin, Zhang Yu, Hu Jun, Hang Fan, and He Jinliang. Scale and context-aware convolutional non-intrusive load monitoring. unpublished, 2019.

[8] Stephen Makonin, Fred Popowich, Ivan V Bajić, Bob Gill, and Lyn Bartram. Exploiting hmm sparsity to perform online real-time nonintrusive load monitoring. *IEEE Transactions on smart grid*, 7(6):2575–2585, 2015.

[9] Stephen Makonin, Fred Popowich, Lyn Bartram, Bob Gill, and Ivan V Bajić. Ampds: A public dataset for load disaggregation and eco-feedback research. In *2013 IEEE Electrical Power & Energy Conference*, pages 1–6. IEEE, 2013.

[10] P Mantey. Eigenvalue sensitivity and state-variable selection. *IEEE transactions on automatic control*, 13(3):263–269, 1968.

[11] Lukas Mauch and Bin Yang. A new approach for supervised power disaggregation by using a deep recurrent lstm network. In *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 63–67. IEEE, 2015.

[12] Mohamed Nait Meziane, Philippe Ravier, Guy Lamarque, Jean-Charles Le Bunetel, and Yves Raingeaud. High accuracy event detection for non-intrusive load monitoring. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2452–2456. IEEE, 2017.

[13] Thomas Picon, Mohamed Nait Meziane, Philippe Ravier, Guy Lamarque, Clarisse Novello, Jean-Charles Le Bunetel, and Yves Raingeaud. Cooll: Controlled on/off loads library, a public dataset of high-sampled electrical signals for appliance identification. *arXiv preprint arXiv:1611.05803*, 2016.

[14] Edward Ramsden. *Hall-effect sensors: theory and application*. Elsevier, 2011.

[15] Sergio Rapuano and Fred J Harris. An introduction to fft and time domain windows. *IEEE instrumentation & measurement magazine*, 10(6):32–44, 2007.

[16] Changho Shin, Sunghwan Joo, Jaeryun Yim, Hyoseop Lee, Taesup Moon, and Wonjong Rhee. Subtask gated networks for non-intrusive load monitoring. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1150–1157, 2019.

[17] Raymond S Turgel. Digital wattmeter using a sampling method. *IEEE Transactions on Instrumentation and Measurement*, 23(4):337–341, 1974.

[18] Electronic Tutorials. Harmonics. `https://www.electronics-tutorials.ws/accircuits/harmonics.html`. Accessed: 2019-08-18.

[19] He Wen, Zhaosheng Teng, SiYu Guo, JingXun Wang, BuMing Yang, Yi Wang, and Tao Chen. Hanning self-convolution window and its application to harmonic analysis. *Science in China Series E: Technological Sciences*, 52(2):467–476, 2009.

[20] Wikipedia contributors. Harmonics (electrical power) — Wikipedia, the free encyclopedia, 2019. [Online; accessed 18-August-2019].

[21] Wikipedia contributors. Split-phase electric power — Wikipedia, the free encyclopedia, 2019. [Online; accessed 18-August-2019].