Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering

André Nydegger Wermundsen

Identification and Classification of Electrical Loads in a Norwegian Household Using Energy Disaggregation Methods of Non-Intrusive Load Monitoring.

Master's thesis in Cybernetics and Robotics
Supervisor: Assoc. Prof. Frank Ove Westad
December 2018

NTNU
Norwegian University of Science and Technology

André Nydegger Wermundsen

Identification and Classification of Electrical Loads in a Norwegian Household Using Energy Disaggregation Methods of Non-Intrusive Load Monitoring.

Master's thesis in Cybernetics and Robotics
Supervisor: Assoc. Prof. Frank Ove Westad
December 2018

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering

Problem Description

Smart meters installed in Norwegian homes enable Norwegian consumers to monitor their households' total real-time electrical usage. Through Non-Intrusive Load Monitoring (NILM), it would be possible to detail how much electricity is consumed by each electric appliance in a household, thus permitting consumers to adopt a more energy-efficient behavior. However, the most common smart meter in Norway, the Kaifa Smart Meter, delivers consumption data only every two seconds. The methods used in the field of NILM usually demand data at a higher frequency than two seconds in order to be able to provide reliable appliance-specific electricity consumption data. To what extent is it possible to gain reliable consumption data with the smart meters available in Norwegian households? In this project, a data collecting experiment in a Norwegian household will be conducted to assess the usability of NILM in this context. In the experiment, individual appliance loads will be identified and classified using traditional NILM methods to disaggregate the total energy consumption of a household.

Acknowledgment

I want to thank the following people for their help during this project. Frank Westad for his guidance and great insights in big data. Joar Harkestad for introducing me to NILM. Ørjan Svendsen at Hark Technologies for enthusiastically sharing his knowledge of the Kaifa smart meter. Hallgeir Horne for the multiple phone calls discussing NILM. Victoria Susanne Nydegger Schrøder and Thea Wehler Knudtzon for proofreading. Last, but not least, I would like to thank my family for being supportive, especially my mother for letting me (intrusively) monitor her electric consumption.

A.N.W

Abstract

The purpose of this thesis is to establish the level of benefit brought to the consumers by the introduction of smart meters. An experiment was designed to collect total consumption data and individual appliance data in a non-lab environment for training purposes. The data collected is formatted to fit the standard set by the evaluation tool NILM-Eval and evaluated with three algorithms: Weiss, Baranski and Voss and Parson. A detailed guide for setting up the experiment and data structure is provided to aid further research. Results show that the possibilities for a specified electricity bill are present but in need of further studies and a substantial premade signature database to enable consumers to fully benefit from NILM.

Sammendrag

Formålet med denne avhandlingen er å fastslå fordelene til forbrukerne ved innføring av smart strømmålere. Et eksperiment ble utformet for å samle inn totalt forbruksdata samt individuell apparatdata i et ikke-laboratoriemiljø til opplæringsformål. Dataene som er samlet inn er formatert til å passe standarden satt av evalueringsverktøyet NILM-Eval og evaluert med tre algoritmer: Weiss, Baranski og Voss og Parson. En detaljert veiledning for oppsett av eksperiment og datastruktur er gitt for å støtte videre forskning. Resultatene viser at mulighetene for en spesifisert strømregning er tilstede, men det er behov for videre studier og en omfattende forhåndslagret signatur-database for å gjøre det mulig for forbrukerne å dra full nytte av NILM.

Preface

The master thesis in front of you is the accumulation of my integrated Master of Science in Engineering Cybernetics at the Department of Engineering Cybernetics at Norwegian University of Science and Technology (NTNU). The idea for this thesis was conceived over a cup of coffee with Joar Hark, CEO in the startup Hark Technologies. Joar informed me about the law changes[1] made by the Norwegian Government resulting in the introduction of smart meters and the possible benefits they bring to the consumer and the environment. Hark Technologies are developing the EcoMonitor, a device to automatically collect the consumption data from Norwegian smart meters, making it easily available for the consumer. As a result, I decided to look into the the possibility to provide consumers with an appliance-specified electricity bill by utilizing the previously unavailable data now accessible through smart meters and the EcoMonitor.

The work was conducted mostly individually and has taken me on a journey through all prior studies at NTNU, new studies in the field of Electric Power Engineering and previously unfamiliar programming languages.

As a reader, it is helpful to have a certain level of scientific knowledge, but with the thesis's focus on consumers, I have tried to present the content in an accessible manner.

André Nydegger Wermundsen
Department of Engineering Cybernetics
18th of December 2018

Contents

Problem Description **1**

Acknowledgment **3**

Abstract **i**

Sammendrag **i**

Preface **ii**

Table of Contents **v**

List of Tables **vii**

List of Figures **x**

Abbreviations **xi**

1 Introduction **1**
1.1 Previous Work 2
1.2 Problem formulation 3
1.2.1 Objectives 3
1.2.2 Norwegian Smart Meters 3
1.3 Outline 4

2 Non-Intrusive Load Monitoring **7**
2.1 History of NILM 7
2.1.1 The Essentials of NILM 7
2.1.2 Evolution of NILM 7
2.2 The Main Steps 8
2.3 Device Signatures 10
2.3.1 Advances in the Field 10

2.3.2 Appliance Categories . . . 12
2.4 Evaluation framework and Disaggregation Algorithms . . . 13
2.4.1 Weiss' Algorithm . . . 14
2.4.2 Parson's Algorithm . . . 14
2.4.3 Baranski's Algorithm . . . 14
2.5 Accuracy Metrics . . . 14

3 Experimental Design **17**
3.1 Norwegian Household Experiment . . . 17
3.1.1 Initial Thoughts . . . 17
3.1.2 Choosing a Norwegian Household . . . 17
3.1.3 Choosing appliances . . . 18
3.2 Data . . . 19
3.2.1 Collecting Data . . . 19
3.3 Hardware . . . 21
3.3.1 Kaifa Smart Meter . . . 21
3.3.2 Plugwise System . . . 21
3.3.3 Hark Technologies EcoMonitor . . . 21
3.3.4 Raspberry Pi b+ 1.2v with microSD cars, WiFi USB dongle and power supply . . . 21
3.4 Software . . . 23
3.4.1 Plugwise Source . . . 23
3.4.2 Raspberian Jessie . . . 23
3.4.3 Plugwise-2-py . . . 23
3.4.4 Node-RED . . . 25
3.4.5 Matlab . . . 25
3.4.6 Data formatting to fit NILM-Eval . . . 25

4 Implementation **29**
4.1 Kaifa Smart Meter and EcoMonitor . . . 29
4.2 Plugwise . . . 29
4.2.1 Source . . . 29
4.2.2 Pictures of Installation . . . 30
4.3 Raspberry Pi . . . 30
4.3.1 Raspberian Jessie . . . 31
4.3.2 Plugwise-2-py . . . 31
4.3.3 Node-RED . . . 32
4.4 Data Formatting . . . 32
4.5 NILM-Eval . . . 33

5 Results **35**
5.1 Evaluation of the experiment . . . 35
5.2 Usability of the Weiss algorithm . . . 35
5.2.1 Finding the Signatures . . . 36
5.2.2 Fridge and TV . . . 36
5.2.3 Parameter Sensitivity . . . 39

5.2.4 Overall Performance . . . 39
5.3 Baranski and Voss . . . 39
5.3.1 Parameter Sensitivity . . . 40
5.3.2 Overall Performance . . . 41
5.4 Parson . . . 41
5.4.1 Fridge . . . 43
5.4.2 Water Heater . . . 44
5.4.3 Microwave oven . . . 46
5.4.4 Electric Heater Terrace . . . 48
5.4.5 Parameter Sensitivity . . . 50
5.4.6 Overall Performance . . . 52

6 Discussion **55**
6.1 Evaluating the Experiment as a whole . . . 55
6.2 Three phases and type of electric distribution to households . . . 55
6.2.1 Zero volts measured on phase 2 voltage . . . 56
6.2.2 Separating phase current loads . . . 56
6.3 Weiss . . . 58
6.4 Baranski . . . 58
6.5 Parson . . . 58
6.5.1 Training . . . 59
6.6 Tuning . . . 59

7 Conclusion **63**
7.1 Conclusions . . . 63

Bibliography **65**

Appendix A : Data collecting software installation guide **69**

A Data collecting software installation guide **71**

Appendix B : Data formatting **77**

B Data formatting **79**
B.1 Matlab Script for Data Formatting . . . 87

List of Tables

1.1 Statutory rules for data measurements in Norwegian smart meters 4
1.2 Accessible active power data from the three different smart meters in Norway. 4

2.1 Algorithms used to evaluate NILM potential by the author 13

3.1 Possible and chosen appliances to monitor and their class. 19
3.2 Accessible active power data from the three different smart meters in Norway. 21
3.3 Smart Meter data as downloaded from EcoMonitor 27
3.4 NILM-Eval data structure 28

5.1 Weiss signatures 36
5.2 Barinski: 20 clusters 40
5.3 Barinski 20 FSM for 20 clusters 41
5.4 Barinski 50 clusters 42
5.5 Barinski 20 FSM for 50 clusters 43
5.6 Parson algorithm results 53

A.1 Table of Smart Plug details in experiment 71

List of Figures

1.1 Possible consumer savings. 2
1.2 Gantt chart for project. 5

2.1 Load Disaggregation 8
2.2 NILM steps. 8
2.3 Power signatures. 10
2.4 2D signature space. 11
2.5 3D signature space. 11
2.6 Electric kettle plot 12
2.7 Washing Machine plot 13
2.8 Laptop plot from the ECO dataset 13

3.1 Apartment floor plan 18
3.2 Previously used appliance meters. 20
3.3 The Plugwise Circle system. 22
3.4 Hark Technologies EcoMonitor 22
3.5 System design overview 24
3.6 Data flow in the system. 26

4.1 Plugwise Source 30
4.2 Installation of hardware 31
4.3 Plugwise-2-py web interface 32
4.4 Node-RED interface 33

5.1 Weiss: Fridge consumption detection 1 37
5.2 Weiss: TV consumption detection 1 37
5.3 Weiss: TV consumption detection 1 38
5.4 Weiss: TV consumption detection 2 38
5.5 Electric Heater Salong consumption 39
5.6 Parson: Fridge consumption and detection 1 44
5.7 Parson: Fridge consumption and detection 2 44

5.8 Parson: Actual fridge consumption and aggregated data 45
5.9 Parson: Identified fridge consumption and aggregated data 45
5.10 Parson: Water Heater consumption and detection 1 46
5.11 Parson: Water Heater consumption and detection 2 46
5.12 Parson: Water Heater consumption and detection 3 47
5.13 Parson: Water Heater consumption and detection 4 47
5.14 Parson: Microwave Oven and detection 1 48
5.15 Parson: Microwave Oven consumption and detection 2 48
5.16 Parson: Actual Microwave Oven consumption and aggregated data 49
5.17 Parson: Identified Microwave Oven consumption and aggregated data . . 49
5.18 Parson: Electric Heater Terrace consumption and detection 1 50
5.19 Parson: Electric Heater Terrace consumption and detection 2 50
5.20 Parson: Electric Heater Terrace consumption and detection 3 51
5.21 Parson: Electric Heater Terrace consumption and detection 4 51

6.1 TT-System 56
6.2 Voltage phases 57
6.3 Phases and circuits 57
6.4 Current phases superimposed 60
6.5 TN-System 61
6.6 Current phases from a different network than TT 62

Abbreviations

ALM	=	Appliance Load Monitoring
ILM	=	Intrusive Load Monitoring
NILM	=	Non-Intrusive Load Monitoring
HAN	=	Home Area Network

Chapter 1

Introduction

The Norwegian government recently passed a law [1] declaring that Norwegian electricity service providers are to install smart meters for all their consumers within 2019. The three main reasons behind the law-passing were that *"...it benefits the customers, society at large and the environment."*[2] . Service providers will also receive the consumer's consumption on an hourly basis, giving them valuable insight into real-time grid load.

The main benefit provided by smart meters is considered to be the consumer's possibilities in understanding their consumption through the available real-time consumption data. This way, consumers will be more aware of their energy usage, and hopefully, make their habits more energy saving. Earlier, the electric consumption was self-reported six times a year by the consumer. With the installed smart meters, electricity providers receive an hourly report stating consumers current consumption giving them the advantage to quickly, and more correctly, create load forecasts for the grid. Also, with the hourly reports, the service providers will be able to determine which customers are contributing to big surges throughout the day. Surges typically appear in the morning and late afternoon, when people are using multiple appliances to cook, heat their houses and charge their electric cars. The grid has to be dimensioned to handle the most significant surge, even though this capacity is only needed a small percentage of the day making it unnecessary powerful and expensive. To get fairer pricing, service providers are discussing introducing rush-hour prices at the times of the day with high grid demand [3].

Many consumers consider the installation of smart meters by law negatively. With their consumption data being delivered to service providers hourly, they feel a breach of privacy at the same time as the hourly changing of kW prices seems unfair compared today's practice. After the law-passing, the focus on wards will lie on the advantages smart meters bring to the consumer. Real-time consumption data gives an excellent way for consumers to learn more about their consumption, hence balancing out their electricity and acting more environmentally.

The smart meters deliver the total electricity consumption, but by solely looking at the aggregated use, it can be difficult for the average consumer to understand which part of their consumption is leading to a higher electricity bill. Research indicates that giving

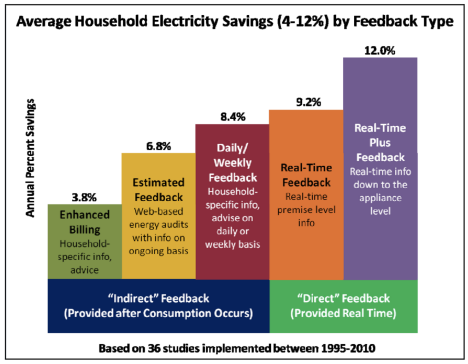


Figure 1.1: Consumer savings potential possible with added feedback according to [4]. The three left columns describe regular NILM with different channels of feedback, with real-time feedback on the two right columns being more advanced methods or intrusive solutions.

appliance-by-appliance feedback can induce electricity savings by around 12 percent in an average household [4]. One way to do this is to use NILM.

1.1 Previous Work

NILM is a technique for disaggregating the total energy load of a household into appliance-specific categories and has been studied since its invention in the eighties by Hart et al, at the Electric Power Research Institute Massachusetts Institute of Technology [5]. Hart et al. showed that by mapping out a few appliance load characteristics, also called signatures, one could deduct which appliances were responsible for the accumulation. After this discovery there has been much research conducted on the topic and several new methods has risen in the field, both supervised as Hart et al. presented and unsupervised, where no appliance load characteristics, are needed. One thing they all have in common is three main steps; data acquisition, extracting features from the data and classifying loads. In NILM, the method should be non-intrusive, and the electric consumption data is therefore collected at the service entry, in our case the Norwegian Smart Meter. Different methods extract features in different ways, but the goal is always to determine when an appliance is turned ON or OFF and to find the most describing features in the data during that period. These features are used to classify the appliance that caused the step change in total consumption. To successfully map features, information about the individual appliances can either be estimated from their specifications or by individually monitoring their consumption (intrusive monitoring).

In his thesis[6], Horne did not have the needed equipment available in time and therefore used downsampled data from an existing public data set to the following sampling frequencies. Today, smart meters are widely installed in Norway, and an experiment with actual smart meter data can be conducted. Previously data collecting done in the field of NILM such as the ECO[7] and REDD[8] has not gone into detail about how their research was set up making it harder to replicate for future studies.

1.2 Problem formulation

Although there has been done much research in the field of NILM, not much information is available on how to best set up a system to collect and store the consumption data in an intrusive fashion. A feasibility study into the possibilities of understanding Norwegian households electric loads through NILM, with the delivered smart meter aggregated data, will hence be conducted focusing on the most installed smart meter, the Kaifa. To evaluate the NILM potential, the evaluation framework NILM-eval[9] created by Distributed Systems Group[10] at ETH Zurich, will be used.

1.2.1 Objectives

The primary objectives of this master thesis are:

- Assess the feasibility of using NILM to provide appliance spesific consumtion data to Norwegian households.
- Design and execute an experiment to collect both smart meter data and individual appliance data in a Norwegian household.
- Analyze the collected data and assess its potential for NILM.
- Identify and classify individual home appliances using Non-Intrusive Load Monitoring methods on aggregated data provided by the Kaifa Smart Meter.
- Map the performance of different NILM approaches and identify shortcomings and possible solutions.

Secondary objectives include:

- Produce a Data Collection Software Installation Guide to aid future experiments on NILM.

1.2.2 Norwegian Smart Meters

According to Norwegian law, the smart meters installed in Norway has to [1]:

- "...store measurement values with a recording frequency of maximum 60 minutes, and be adjustable down to a recording frequency of minimum 15 minutes."
- "...have a standardized interface that facilitates communication with external equipment based on open standards."
- "...be able to connect and communicate with other smart meters."
- "...ensure that stored data is not lost in case of power failure."
- "...be able to break and limit power withdraw at the point of delivery, excluding transformer-based facilities."

- "...be able to send and receive information regarding power prices and tariffs as well as be able to transfer control and ground fault signals."
- "...provide security against abuse of data and unwanted access to control functions."
- "...register flow of active and reactive power in both directions."

The consumers in Norway will get one of three different smart meters installed depending on which one their service provider decided to buy, e.i. depending on what part of the country they live. One common thing for all the smart meters is that they have to deliver data within the decided intervals ruled by Norwegian law, see figure 1.1. In the case of NILM, higher data frequenzy equals better posibillity for identification and classification. The most significant difference between the selected smart meters is that they deliver data at slightly different intervals, see figure 1.2. One of the benefits of getting the smart meter installed is the access to the households consumption data, this difference in data frequency between the different areas in Norway can be seen as unfair.

Frequenzy	**Data**
Every 2.5 seconds	Active power, import and export
Every 10 seconds	Reactive power, import and export Current, all phases Voltage, all phases
Every hour	Active energy, import and export Reactive energy, import and export Time and date

Table 1.1: Statutory rules for data measurements in Norwegian smart meters.

Smart Meter	**Active power obtained from HAN**
Nuri Telecom LtD (Kaifa Smart Meter)	Every 2 seconds
Kampstrup Smart Meter	Every 2.5 seconds
Aidon Smart Meter	Every 2 to 3 seconds

Table 1.2: Accessible active power data from the three different smart meters in Norway.

1.3 Outline

This thesis is organized in many ways the same as NILM is conducted, Data acquisition, feature extraction, and appliance identification. First, the second chapter will examine the research previously conducted in the field of NILM. Throughout the chapter, one will gain a deeper understanding of feature extraction, appliance identification, and the different approaches to successful disaggregation of smart meter data. Chapter three describes the practical data acquisition, how to conduct a household experiment and the methods used

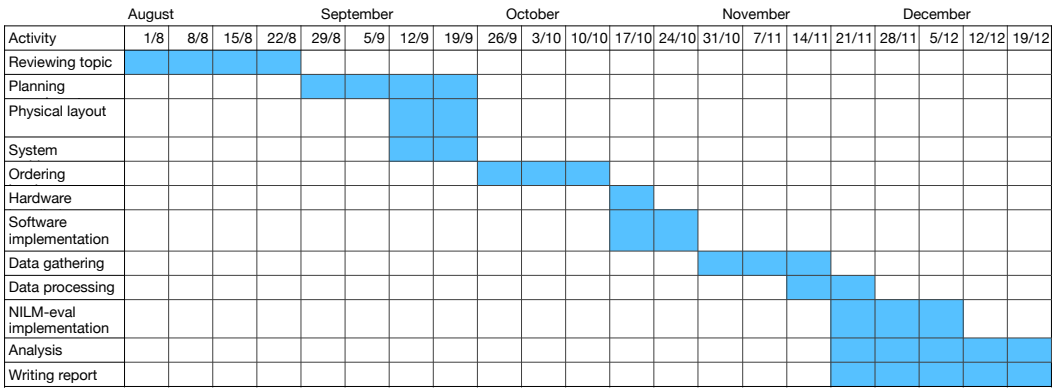


Figure 1.2: Gantt chart for project.

to collect and process the data using theory from chapter two. Chapter four explains the implementation of the experiment and the challenges that was faced, hopefully aiding future experiments in the field. The results of the experiment, the data and the appliance identification are presented in chapter five before they are discussed in chapter six. A conclusion to this thesis's objectives is then given in chapter seven.

Chapter 2

Non-Intrusive Load Monitoring

This chapter introduces NILM, beginning with the initial approach and ending with today's most used algorithms and their metrics.

2.1 History of NILM

2.1.1 The Essentials of NILM

The first mention of NILM was made at MIT in the 1980s by a research group led by George Hart. They envisioned that it would be possible to use the known characteristics of electrical appliances to disaggregate the total electric consumption of a household successfully. The idea was that by looking at the changes in total load, these step changes would contain some characteristic values depending on the appliance changing state. NILM can be formulated as

$$P(t) = P_1(t) + P_2(t) + P_3(t) + ...P_n(t)$$

where the total electricity consumption is made up by the n amount of appliances in the household. In total, all of the sockets in a household, including losses in the wiring, would add to the consumption measured at the entry point of the house. By recognizing the changing state of appliances, the goal was to identify the cause of total consumption as shown in figure 2.1.

2.1.2 Evolution of NILM

Even though the idea was conceived decades ago, there was not much research done in the following decades. At the time, smart meters such as the ones we have today were not available, and a device would have to be retrofitted to the existing revenue meters. The computing power needed to process data was also expensive, and the non-intrusive method would, in reality, become dificult to implement. During the past decade, rapid development in machine learning and artificial intelligence, combined with cheap computing

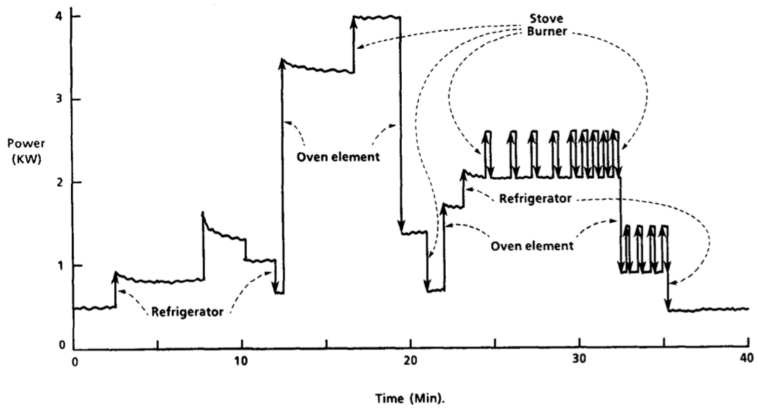


Figure 2.1: Load Disaggregation as visualized by Hart[11] in 1992.

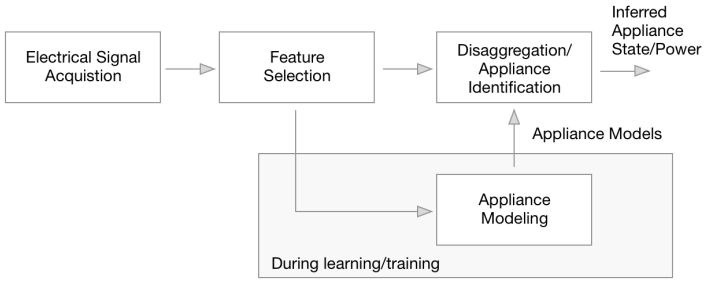


Figure 2.2: NILM steps as used by Wong [13]

power and available smart meters, has made NILM a much more researched field. In their 2014 paper "Non-intrusive load monitoring and supplementary techniques for home energy management"[12], Bahman Naghibi and Sara Deilamis proposed new learning methods based on machine learning, revealing a new potential of the field.

2.2 The Main Steps

Despite different new methods of NILM, most approaches are still based on the original work[5] by Hart et al.. The main stages can be divided into three phases: Data acquisition, feature extraction, and appliance identification.

Data acquisition

Obtaining aggregated data today is most easily done through smart meters. The downside of using smart meters is that the available data is decided by the manufacturer or state,

which in some cases can limit the NILM possibilities. E.g., the XXXXXX has the capability of delivering other measurements such as phase shifts between current and voltage per phase[6]. These measurements are not available for the consumer. A positive example is the Kaifa, which delivers data at a higher frequency than Kampstrup and Aidon (as shown in table 1.2) due the inner workings making it difficult to measure correctly at a 2.5 interval. In this case, an exception made by the Norwegian Government[6]. The upside of using smart meters is, of course, the smart meters availability, as they are to be installed in all Norwegian households.

Feature extraction

Which features that needs to be extracted from the aggregated data differs in different approaches. While most look at active or reactive power, there are also methods using higher harmonics such as phase shifts caused by inductive components in the appliances. It usually comes down to the amount of data available.

The next step is detecting an event, such as the change of state in an appliance. The features can be extracted in different states and are classified as:

- Steady state features

  Steady state is the time when an appliance is operating normally. The variations in real and reactive power are typically used to recognize a change of state.

- Transient state features

  Features extracted when the load is changing. Many inductive appliances have transient states when turned ON and OFF, and these fast-changing loads will be easier to sort out from the total consumption, but also require a higher sampling rate.

- Non-traditional features

  Features that are not directly connected to electric consumption. This could be the number of people in the household, time of day or probability caused by the other currently used appliances.

A challenge in event detection is the many changes that can occur in the aggregated data due to instability in the grid or a load shift caused by a misbehaving appliance in steady state. Another problem is that with low-resolution data, the event detection suffers. If an event occurs between the measuring intervals, the event could be only partially caught or even completely missed. Therefore it is also important to know if the event is caused in the change of state, a steady state or a transient stage.

Appliance identification

The last step is to map the features found and identify the appliance which caused it. Different approaches have different ways of doing this and will be discussed further in this chapter.

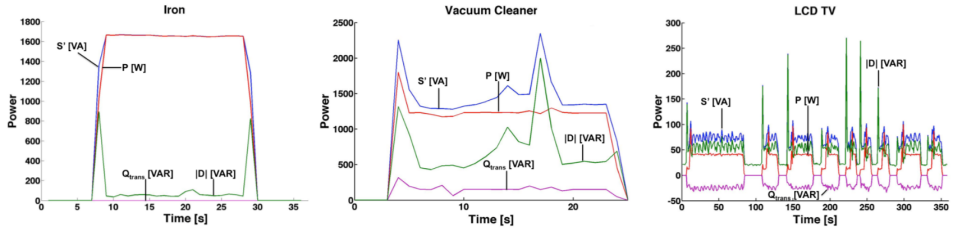


Figure 2.3: Power signatures for an iron (restive), vacuum cleaner (inductive) and LCD TV (capacitive). Illustration borrowed from Weiss [14]

2.3 Device Signatures

The signature in the aggregated data, often called an edge, is the step change when a load is added or subtracted. These appear when either the state or load changes. The change differs depending on the nature of the device and can be divided into three categories: resistive (Watt), inductive (volts-amps-reactive, or VAR) and capacitive (Also VAR, but opposite polarity of inductive). E.g., an incandescent light bulb is completely resistive because it draws current and voltage in phase to heat the wire filament that immediately dissipates light and heat. A vacuum cleaner is inductive because its motor demanding a big current to create a magnetic field during startup. In contrast to resistive loads, this magnetic field resists changes in current, resulting in a 90-degree phase shift between the current and voltage. Lastly, a capacitive load such as a laptop charger contains capacitors which can store a charge making it withstand voltage changes resulting putting it in an opposite 90-degree phase shift. Figure 2.3 illustrates the different power signatures.

The initial approach made by Hart et al. was to use the real (P) and reactive (Q) power in the aggregated data [5]. An edge detector would find the steady-state changes (steps) and plot these in a two-dimensional plane ΔP and Δ Q as shown in figure 2.4. By clustering these together, the appliances could be separated, and their state could be noted. The signature space de-complicated the complex data into easily separatable clusters dynamically, where negative responses would be matched or paired with their opposite positive responses (for two state appliances). New, unmapped appliances, would then either show up in existing clusters, or create new clusters that could be mapped at a later time. To calculate their consumption, a premade table of each appliance would contain the consumption details.

2.3.1 Advances in the Field

In the approach made by Hart et al., classifying other devices than the big appliances such as refrigerators, washing machines and ovens, proved challenging, but not impossible. A shortcoming was that some appliances would create similar signatures, making it difficult to separate them in individual clusters. This problem is addressed in newer approaches where a new dimension has been introduced to further isolate the signatures in higher dimensional space, see figure 2.5. Another predicament is that even though this approach is non-intrusive, it still requires the household appliance's loads to be appropriately mapped

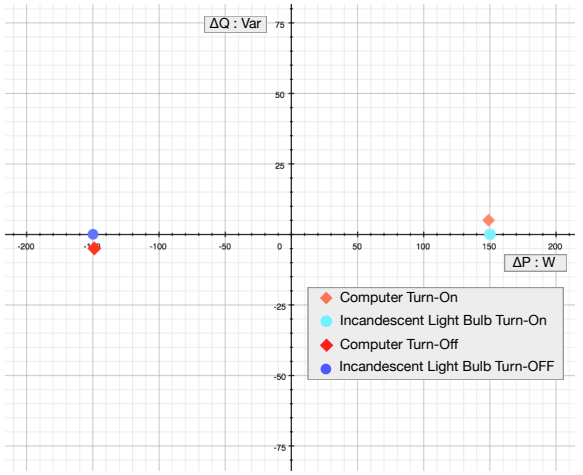


Figure 2.4: Signatures from a computer and incandescent light bulb illustrated in 2D space. Notice how the incandescent light bulb draws no reactive power due to being purely resistive.

beforehand. This could be difficult to do when a device has several operating states, e.i an oven, creating numerous signatures in need of clustered in separate clusters, despite only being one appliance. The newer articles by Hart, [11] and [15], tries to address these problem, as well as [16] who introduced a new dimension,[16] who uses clustering through advanced machine learning and [14] who isolates phases.

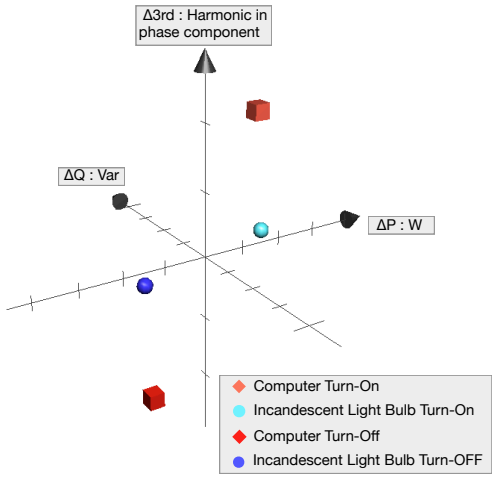


Figure 2.5: Signatures from a computer and incandescent light bulb illustrated in 3D space. Notice how the the two appliances are easily seperatable when adding a new dimension.

2.3.2 Appliance Categories

When mapping events to appliances in standard NILM, the signatures have to be known. For some appliances, finding the signatures is easier than for others depending on their operation principle and load properties. By looking at the appliances' operational states, Hart et al. defined four different categories: USE THIS [17]

1. **ON/OFF**

   Appliances that have two distinct operating states, ON and OFF. They either draw power at a set level (e.g. a 60 watt light bulb) or nothing. A change of steady state in this category will result in the same changing values every time, providing an easily detectable signature as seen in figure 2.6.

2. **Finite state machine (FSM)**

   While the "ON/Off" category only contains one discrete operational state, the FSM-category contains appliances which have multiple discrete operational states, such as a stove or a washing machine, see figure 2.7. These various power loads will essentially be different "ON/OFF" appliances, but it is essential for the algorithms to distinguish these two categories so to not misclassify the multiple states as separate appliances.

3. **Continuous state machine (CSM)**

   Continuous state machines are devices that due to their almost step-less states has no fixed pattern to recognize. E.g., a battery charger, adjustable lights, and power drills, see figure 2.8

4. **Permanent consumer**

   The fourth category captures devices that by design always stay on, such as hard-wired fire alarms and burglar alarms. These devices are still drawing power due to their design. Instead of these being detected, they will typically add to the zero-level of the household electrical consumption.

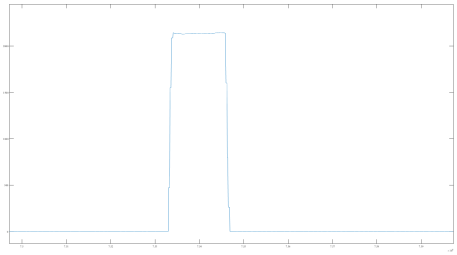


Figure 2.6: Class 1: Electric kettle consumption during one run.

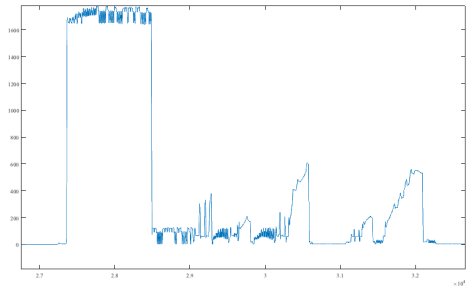


Figure 2.7: Class 2: Washing Machine consumption during one cycle.

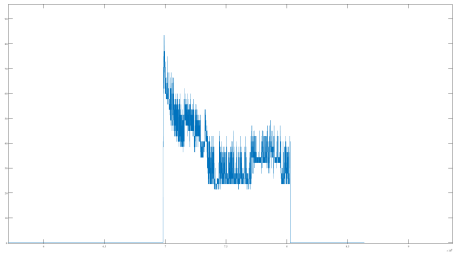


Figure 2.8: Class 3: Laptop consumption while charging. Data borrowed from the ECO dataset[7].

2.4 Evaluation framework and Disaggregation Algorithms

Today there are many approaches to disaggregate total electrical consumption data. While many build upon the initial concept laid out by Hart et. al., some interesting machine learning methods are used as well. Many of these produce good results, the problem is to compare them in a good way. Usually, they use different measures of accuracy, and each offers many different parameters for tuning. To ease the process of evaluating these algorithms against each other, two frameworks, the NILM-eval[9] and NILMTK[18], have been developed during earlier research. Neither of the evaluation tools had been updated the last couple of years at the time of writing, but due to Hallgeir Horne's previous choice of using NILM-Eval, this Matlab toolkit was chosen for this thesis as well. NILM-eval features four separate algorithms: Weiss, Barinski, Parson and Kolter. Due to missing dependencies, Kolter does not work and only the remaining three will be evaluated.

Authors	**Learning**	**Granularity**	**Main features**
Weiss et al.	Supervised	1 second	Real power and reactive power
Parson et al.	Semi-supervised	1 minute	Real power
Baranski and Voss	Unsupervised	1 second	Real power

Table 2.1: Algorithms used to evaluate NILM potential by the author

2.4.1 Weiss' Algorithm

Weiss' [14] algorithm is based on the approach Hart proposed in the 1980s, where a signature describing active and reactive power is needed per appliance. As previously mentioned, the signature Hart created was made up by the appliance's real and reactive power when changing state. The events are then mapped on a two-dimensional plane and clustered together. Instead of using two-dimensions, Weiss uses three: real power, reactive power, and distortive power. Distortive power is a calculated component of reactive power found by looking at the phase shift created by the load, and Weiss takes advantage of the phases splitting each event into a detailed phase-view that further increase its precision. After finding these events, they get stored in a signature database where they can be matched up to future events making this approach supervised.

2.4.2 Parson's Algorithm

Parson's algorithm[19] uses Hidden Markov Models (HMM) to disaggregate the household appliances one by one. HMM [20] observes a series of observations and find the most probable state transitions. The key that a state is viewed as hidden for the observer, in this case, the state of an appliance getting turned on or off being buried in the total consumption data. During training, the algorithm finds periods where the pattern of the appliance is apparent and uses this pattern to update a general model of the appliance. The general model contains the means of individual appliance consumption at its states, e.g. how much power a fridge usually consumes. After training, it removes devices, one by one, from the aggregated data and continues the search for others. Due to its training model, Parson's algorithm is a semi-supervised method. The method stands out from the Weiss and Barinski with its default granuality of 1/60Hz, meaning it only needs to know the aggregated consumption delivered from Kaifa every 60 seconds.

2.4.3 Baranski's Algorithm

Baranski's[21] algorithm is designed to perform NILM on consumption data that is not as detailed as the previous algorithms. Baranski's algorithm takes in what it calls "rough data" and " It is designed to detect frequently occurring patterns from the load trace of the active power consumption without any a priori knowledge concerning special appliances."[22]. With the algorithm focusing on frequently reoccurring patterns, and appliances creating significantly big loads, smaller appliances will not be prioritized. A discovered appliance will be defined as a final state machine with a set number of states, where the states are assumed to always repeat in the same sequence within the pattern. In their paper[21], the authors conclude that "Typical patterns of ON-OFF appliances or finite state machines with less than about five different states could be detected without any apriori knowledge."

2.5 Accuracy Metrics

Several different metrics measure the success of an algorithm. Although it is hard to compare different approaches, there are a few metrics available cross-algorithm that are good performance metrics used to evaluate machine learning which NILM-Eval offers.

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.1}$$

Precision measures the number of correct classifications done by the algorithm. A true positive is a correctly identified change of state, event, in data and a false positive is when the thought edge is not present. Precision is a useful metric for what works in the approach, but in reality, it only tells half the story. E.g., if the water heater turns on ten times a day, just one correctly identified edge will produce a precision score of 100%, granted no false positives exist, possibly deceiving the algorithm's machine learning. The precision alone is hence not suitable as a sole parameter to evaluate the performance of the algorithm.

Addressing the problem of precision, recall is introduced. Recall tells the other half of the story revealing the poor performance not seen by precision. In the case of the water heater, precision rates the algorithm 100% successful while recall will rate it 1/10 = 10 percent.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2.2}$$

By combining precision and recall in what is called an F-score, a more nuanced metric appears.

$$F\text{-}score = 2 * \frac{precision * recall}{precision + recall} \tag{2.3}$$

F-score weights the two previously mentioned metrics equally by creating a harmonic mean between them, punishing extreme values. E.g., if either one of them is extremely low, the F-score will reflect it.

Chapter 3

Experimental Design

This project aims to assess the feasibility of using NILM to provide real-time consumption data to Norwegian households through mandatory state-approved smart meters. Since being imposed by Norwegian law, smart meters are now widely installed in households in Norway. This makes it possible to conduct an experiment with actual smart meter data, as Horne[6] was not able to do at the time. With the previous work in the field lacking a detailed guide for how to set up data acquisition, this chapter will provide in-depth information about the planning of a data gathering experiment to aid future research.

3.1 Norwegian Household Experiment

When designing an experiment, there are many possible pitfalls. Determining the data available, defining a typical household and deciding which hardware is available within the budget are important information needed to base hardware and software choices.

3.1.1 Initial Thoughts

First and foremost, the quality of the data collected is of outmost importance. The quality depends on several factors, and in this project, the aggregated data is only available through the confinements of the Kaifa smart meter. The remaining choices will have to be made based on this in order to assess the potential of NILM in a Norwegian household. What data needs to be collected and what hardware does one need to collect it? Which electric appliances does a Norwegian household usually contain?

3.1.2 Choosing a Norwegian Household

In previous research[7][8][23], houses occupied by small families or couples with heating and cooking equipment typical for their country were used. For this experiment, the authors mothers household was used, with two people occupying it during the whole time period. The apartment is approximately $115m^2$ with the floor plan showed in figure 3.1.

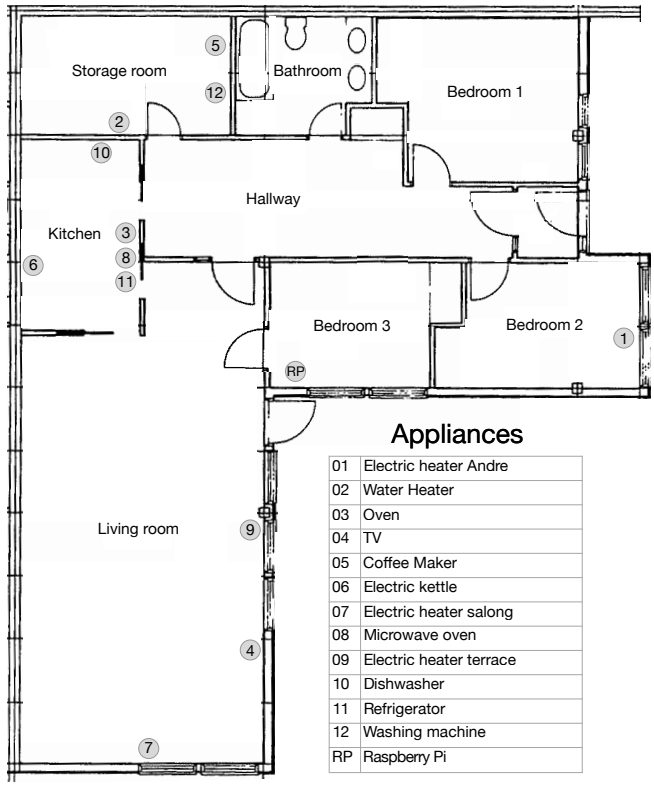


Figure 3.1: Floor plan of the apartment where the experiment was conducted with appliances and Raspberry Pi marked.

3.1.3 Choosing appliances

For consumers, the biggest difference in consumption can be made by changing their habits regarding the appliances used daily, especially the devices creating large loads. Defining a Norwegian household was done by consulting the author's peers, and it became clear that the appliances shown in table 3.1 are the most common.

As mentioned in chapter two, some of these are more difficult to classify because they have continuous states or are always ON. Some are also hard to measure because of physical restrictions. To reduce electricity consumption and lowering the electricity cost of a household, the most important appliances to monitor will as mentioned be the largest consumers in the household. The biggest restrictions in this experiment were the physical constraints for some appliances. The kitchen stovetop demands 25A of current and uses a different plug than other appliances, hence it will not be included. Heated floors are directly connected to the wiring and hard to measure without special equipment and installation by authorized workers. The author's mother did not use a laundry dryer or a PC, and her laptop is rarely used, these will hence be excluded. Not measuring the laptop is a drawback to the experiment since its charger is the only capacitive load in the

Appliance	Class	Chosen for experiment
Kitchen oven	3	Yes
Kitchen stove top	2	No
Microwave	3	Yes
Refrigerator	1	Yes
Dishwasher	2	Yes
Electric Kettle	1	Yes
Coffee Maker	1	Yes
3 Electric heaters	3	Yes
Heated floors	3	No
Water heater	1	Yes
Washing machine	2	Yes
Dryer	2	No
TV	3	Yes
Stereo	3	No
Lights	1	No

Table 3.1: Possible and chosen appliances to monitor and their class.

household. On the upside, there are almost no capacitive loads in a regular household. The appliances chosen to monitor were therefore the ones shown in table 3.1:

3.2 Data

3.2.1 Collecting Data

There are two types of data necessary, aggregated data and individual appliance data for training and verification purposes depending on the algorithm used.

Total aggregated data

All Norwegian smart meters features a HAN-port where consumers can connect and read their consumption data. Connecting, downloading and formatting this data is complicated for the users, creating the need for a dedicated device. This thesis is in part written for Hark Technologies, a company in the early stages of producing such a device. This experiment will hence use Hark Technologies's second prototype of the EcoMonitor, see figure 3.4 to collect the household aggregated data. EcoMonitor formats and relays the aggregated data to a web server available for customers through a website.

Individual Appliance Data

The Kaifa smart meter relays the data every 2 seconds. To achieve the best possible results in NILM, the training data should also be sampled every 2 seconds. On the Norwegian

marked today, there are several different possibilities for monitoring the electricity consumption of an individual appliance. Smart plugs such as the Fibaro Wall plug 2[24], TP-Link WiFi Smart Plug[25], D-Link DSP-W115[26], Eve energy[27], and Belkin WeMo Insight[28] are consumer based devices with its main focus on delivering services for automatization and remote triggering of devices through smartphones. Even though they deliver load monitoring for the connected appliances to some degree, their sampling frequency is far lower than the needed 0.5Hz.

In the 2017 Ph.D. *Thesis Disaggregation of Domestic Smart Meter Energy Data*[29] the author compiled the data collection hardware used in earlier experiments, see figure 3.2. Due to the Enmetric PowerPort being discontinued and the IBM nPlug/jPlug, eGauge meters and 'Watts Up?' meters being very intrusive and expensive to install, the Plugwise Circle was chosen for this experiment. One should note that there are some disadvantages when deciding to use smart plugs instead of more intrusive systems. One is the physical restriction mentioned, resulting in the kitchen stove top getting left out of the experiment. They also have a maximum current measurement capability of around 16A and more noise in the readings (the noise of Plugwise Circle is 5% $\pm$ 0,5 W. [30]). Fortunately, with the stove already excluded, the remaining appliances all fit well within the current range. However, the biggest drawback is the lacking measurement of reactive power, as will be discussed in chapter five.

Model	Manufacturer	Price (£)	Used to record	Sample period (secs)	Network connection
EcoManager Transmitter Plug[a]	EDF & Current Cost	15	UK-DALE	6	433 MHz TRX
IAM[b]	Current Cost	13	-	6	433 MHz TX
Kill A Watt[c]	P3 International	16	-	N/A	none
Circle[d]	Plugwise	28	Tracebase	1	ZigBee mesh
HomeMatic Radio Plug[e]	eQ-3	39	-	5 (best[f])	868 MHz TRX
PowerPort[g]	Enmetric	[n]	REDD	1	IEEE 802.15.4 wireless
nPlug[h]	IBM Research India	105	iAWE	1	Wi-Fi
PowerScout[i]	DENT Instuments	[n]	AMPds	60	serial or Ethernet
eGauge meters[j]	eGauge	350	Dataport	1	Ethernet or PLC or Wi-Fi
'Watts Up?' meters[k]	'Watts Up?'	100	Kelly 2011	1	Ethernet
WiFiPlug[l]	WiFi Plug	45	-	[n]	Wi-Fi
PowerBlade[m]	EECS at Uni. of Michigan	7·7	-	<1	BLE

Figure 3.2: Previously used appliance meters, borrowed from [29].

3.3 Hardware

3.3.1 Kaifa Smart Meter

The Kaifa smart meter used in the experiment is one of three different smart meters installed in Norway. In comparison to the other two, the Kampstrup and Aidon, the Kaifa offers a better data resolution as seen again in figure 3.2. The choice of smart meter for this thesis was based on this fact, to assess the possibilities of NILM on the best data available.

Smart Meter	Active power obtained from HAN
Nuri Telecom LtD (Kaifa Smart Meter)	Every 2 seconds
Kampstrup Smart Meter	Every 2.5 seconds
Aidon Smart Meter	Every 2 to 3 seconds

Table 3.2: Accessible active power data from the three different smart meters in Norway.

3.3.2 Plugwise System

The Plugwise Circle system consists of two types of smart plugs, the Circle and Circle+, and a USB-stick named Stick, displayed in figure 3.3. The Circle+ has all the same features as Circle but also includes an internal clock that is used to synchronize all of the Circles in the household in a mesh network over a Zigbee wireless protocol. The Stick is used for communication between the mesh network and a computer to control, monitor, visualize and schedule the Circles power states (ON/OFF) and their consumption. The smart plugs also have built-in memory to buffer the loads for re-transmitting should the Zigbee connection drop. Although this function is not essential for gathering the data in the case of the thesis, it could aid in the initial implementation described in chapter four.

The Plugwise system by standard does not provide the necessary sampling frequenzy[31] and a workaround needed to resolve this problem will be laid out later in this chapter.

3.3.3 Hark Technologies EcoMonitor

The Hark Technologies EcoMonitor is connected to the household's smart meter HAN-port with a cat5 cable and is powered from an external power source. The communication between the EcoMonitor and the consumer is through WiFi. In the chosen household, the Kaifa smart meter was installed out of the household's WiFi reach, Setting up a shared internet connection from a smartphone in the proximity of the EcoMonitor solved this problem, adding the need for a smartphone in the system. This kind of issue is something that might affect the commercialization of the technology and should be addressed.

3.3.4 Raspberry Pi b+ 1.2v with microSD cars, WiFi USB dongle and power supply

To communicate with the Plugwise system, and to store the consumption data collected, a computer had to stay online during the experiment. Because of its small footprint, many

Figure 3.3: The Plugwise Circle+, Circle and Stick.

Figure 3.4: The Hark Technologies EcoMonitor.

possible configurations, and easy setup, the single board computer Raspberry Pi b+ 1.2 was selected. One important thing to note is that the microSD card used with the Raspberry Pi should be of high quality with fast read and write speed. Data will be processed at least every 2 seconds, and the cheaper, slower cards will be a bottleneck in the system. It is also imperative to use a power supply that delivers the correct voltage and current[32] to make sure the Raspberry Pi does not underperform. A Wifi dongle is not necessary but was added to extract the collected data easily. The following hardware is necessary for the experiment. Figure 3.5 shows the components related to each other.

- 1 Kaifa Smart Meter
- 1 Hark Technologies EcoMonitor
- 1 Plugwise Circle+
- 11 Plugwise Circle
- 1 Plugwise Stick
- 1 Raspberry Pi b+ with microSD cars, WiFi USB dongle, and power supply

3.4 Software

With its default 10 second resolution, the Plugwise system does not provide data matching the Kaifa smart meter, and a more custom approach had to be designed. Most of the software used in this experiment is the publicly available software Plugwise-2-Py[33], a hack created to communicate directly to the Plugwise system without using the Plugwise provided software. With additional software included, this section will lay out the programs needed to run on the Raspberry Pi.

3.4.1 Plugwise Source

Plugwise Source is the software package that comes with the Plugwise system. Source offers useful insight into the electrical consumption with adjustable graphs and estimations of the individual appliances electrical cost, based on a given rate. All connected devices can also be turned ON and OFF either manually or by time schedules. Unfortunately, the sampling frequency is by the standard set to every 10 seconds, significantly lower than the Kaifa Smart Meter. Hence, Plugwise Source was only used for the initial setup.

3.4.2 Raspberian Jessie

Raspberry Pi B+ can run several different operating systems based on Linux. For this experiment, a light operating system with desktop-capabilities was wanted to comfortably install software and being able to connect through VNC[34] remotely. Raspberian Jessie[35] offered all the mentioned features and was deemed a good fit for the job with its stable Debian Linux base.

3.4.3 Plugwise-2-py

Plugwise-2-py is built on the reverse engineering of the Plugwise system. By intersecting the network packages, data being sent in-between the system components has been recorded and mapped out into commands and data. With this information, one can control and read the Circle+ and Circles in different ways than the Source software allows. By repeatedly asking the smart plugs what their current electrical load is, the data could be polled at a faster interval than every 10 seconds. When asked how much the sampling can be increased by, the customer support at Plugwise answered that even though the Plugwise smart plugs can send the actual power consumption every second, but is not designed for it and the functionality is therefore not guaranteed. A value will always be sent if it is polled every second, but it could in some cases be the the sample of the previous second. In other

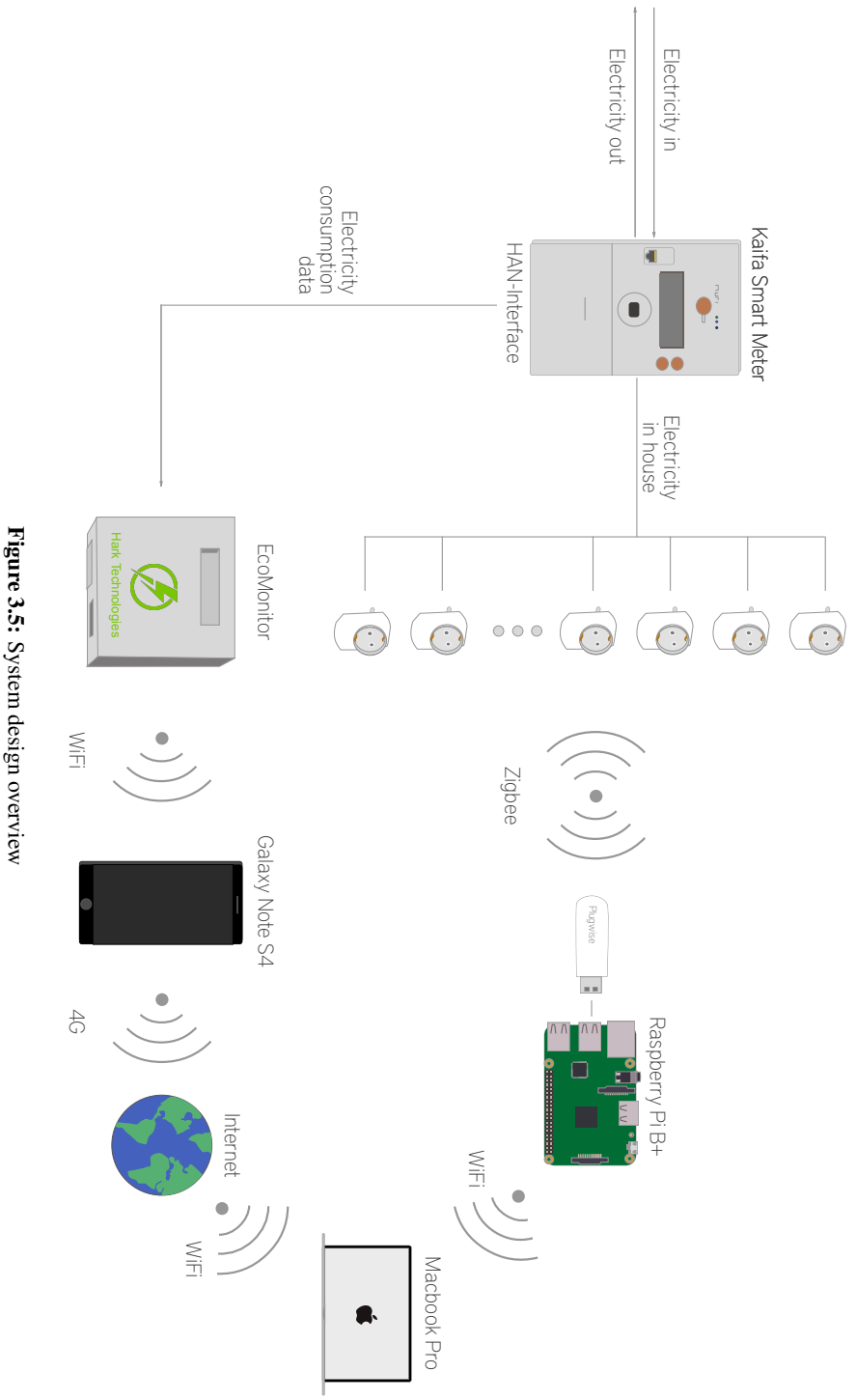


Figure 3.5: System design overview

words, by polling more data one could, in the worst case, get duplicate messages if the smart plug does not have new data. On the basis that this Plugwise setup worked in the Trackbase experiment 3.2 and in other experiments [36] it was also chosen for this thesis.

3.4.4 Node-RED

Plugwise-2-py does not store any data, and since collecting and storing data is the purpose of this experiments system, additional software is needed. *"Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways"*[37]. The connection possibilities to, among others, Domoticz is widely used to control and store historical data in the Plugwise-2-py community. This lightweight home automation system was therefore chosen to relay data within the Raspberry Pi's software modules.

3.4.5 Matlab

For formatting raw data and utilizing the NILM-eval evaluation kit, the tool Matlab was used. As mentioned in chapter 2, a second evaluation kit was available[18], but for this report, the NILM-Eval software is used to assess the available algorithms. Hence, the main software components of the system are:

- Plugwise source
- Raspberian Jessie
- Plugwise-2-py
- Node-Red
- Matlab

3.4.6 Data formatting to fit NILM-Eval

Hark Technologies informed the author that the data downloaded from their EcoMonitor will appear as in table 3.3. NILM-eval requiers the data to be formattet to fit their structure, see table 3.4. Preparing the data is hence also a task need to be done. For this, python and matlab will be used. The flow of data in the experiment is as shown in figure 3.6

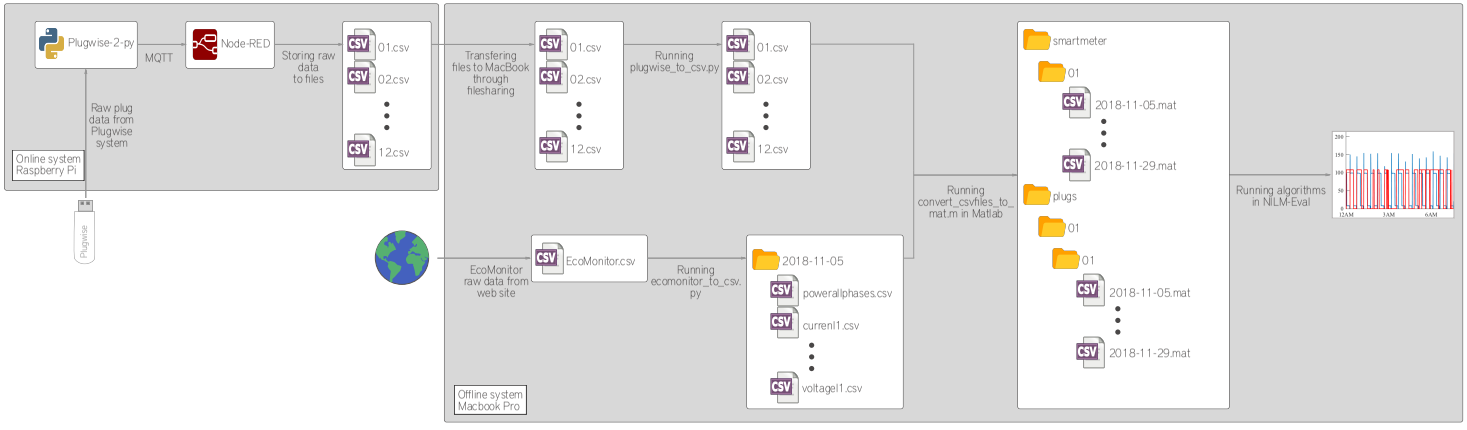


Figure 3.6: The data flow in the system. Raw plug-data is sent from the Plugwise system into the online system of the Raspberry Pi and stored in CSV-files before needing to be manually processed. The raw smart meter data is collected from the EcoMonitor website and then manually processed. Flow with Matlab is not illustrated, this is described in Appendix C.

Name in EcoMonitor data	Description	Unit	Sampling frequency
sID	EcoMonitor ID	-	Every 2 seconds
DTM	Timestamp	-	Every 2 seconds
Pi	Active power imported	KW	Every 2 seconds
Pe	Active power exported	KW	Every 10 seconds
Qi	Reactive power imported	kVAr	Every 10 seconds
Qe	Reactive power exported	kVAr	Every 10 seconds
I1	Current phase 1	Ampere	Every 10 seconds
I2	Current phase 2	Ampere	Every 10 seconds
I3	Current phase 3	Ampere	Every 10 seconds
U1	Voltage phase 1	Volt	Every 10 seconds
U2	Voltage phase 2	Volt	Every 10 seconds
U3	Voltage phase 3	Volt	Every 10 seconds
Ai	Tota active energy imported	KWh	Every hour
Ae	Total active energy exported	KWh	Every hour
Ri	Total reactive energy imported	kVArh	Every hour
Re	Total reactive energy exported	kVArh	Every hour

Table 3.3: Data downloaded from EcoMonitor website

Name	Description	Available	Rate	Comment
powerallphases	Sum of real power over all phases	Yes	2s	
powerl1	Real power phase 1	Yes	10s	P1 = I1 * U1
powerl2	Real power phase 2	Yes	10s	P1 = I2 * U2
powerl3	Real power phase 3	Yes	10s	P1 = I3 * U3
currentneutral	Neutral current	No		
currentl1	Current phase 1	Yes	10s	
currentl2	Current phase 2	Yes	10s	
currentl3	Current phase 3	Yes	10s	
voltagel1	Voltage phase 1	Yes	10s	
voltagel2	Voltage phase 2	Yes	10s	
voltagel3	Voltage phase 3	Yes	10s	
phaseanglevoltagel2l1	Phase shift between voltage on phase 2 and 1	[1]		Knowledge of voltage star
phaseanglevoltagel3l1	Phase shift between voltage on phase 3 and 1	[1]		Knowledge of voltage star
phaseanglecurrentvoltagel1	Phase shift between current/voltage on phase 1	No		
phaseanglecurrentvoltagel2	Phase shift between current/voltage on phase 2	No		
phaseanglecurrentvoltagel3	Phase shift between current/voltage on phase 3	No		

Table 3.4: Demanded NILM-Eval data structure with added availability of data from the Kaifa smart meter. [1]: Assuming no errors in the electrical system, the phase voltage is always separated 120 degrees.

Chapter 4

Implementation

This chapter will take the reader through the implementation of the planned experiment covered in chapter 3. Installing the Plugwise-2-py software on the Raspberry Pi, and getting it to work with the Plugwise hardware, proved more difficult than anticipated. The data will go through multiple steps, and with one step failing, much debugging will result. Much time went into error searching both in code and in hardware before getting everything interacting together. There are some online resources to find answers for the Plugwise-2-py software, such as the forum `https://www.domoticz.com/forum/` and of course the software's GitHub repository `https://github.com/SevenW/Plugwise-2-py`, but it could be made easier for future experiments. In appendix A you will find a step-by-step guide explaining how to set up the software. This chapter will go briefly through the highlights of each hardware and software module.

4.1 Kaifa Smart Meter and EcoMonitor

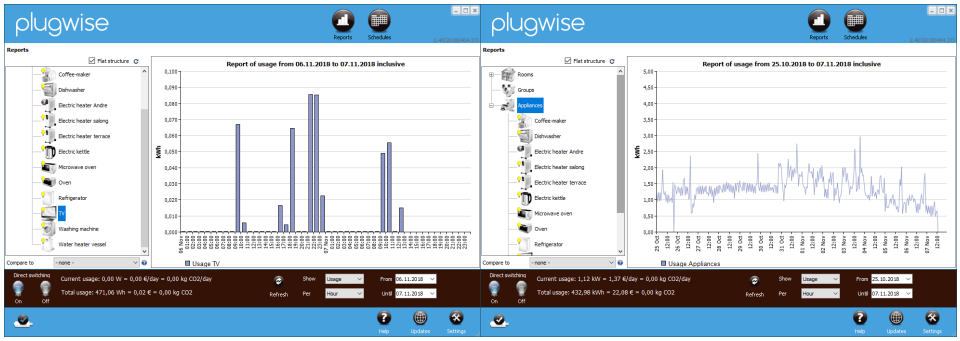
The EcoMonitor was connected through an ethernet cable to the Kaifa HAN-port, see figure 4.2a. The smartphone with a shared network from its 4G cellular connection was placed in close proximity.

4.2 Plugwise

4.2.1 Source

The smart plugs were initialized in a network created through a wizard with appliance names and locations. During this process the smart plugs were connected to an extension cord and labeled so to be completely configured before intrusively installing them. The set up system is shown in screenshots in figure 4.2.

(a) Settings for groups **(b)** Settings for appliances



(c) Report for TV on a given day **(d)** Report for usage

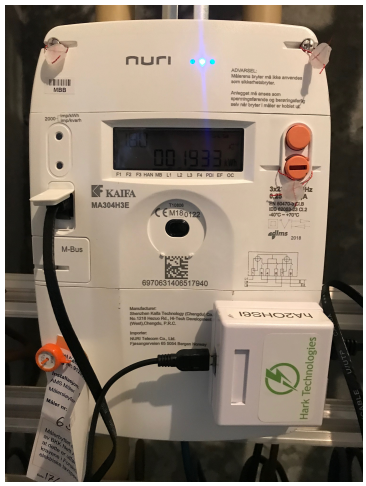
Figure 4.1: Screenshots from Plugwise Source software

4.2.2 Pictures of Installation

Installing the smart plugs went swiftly on the most part, as they are quick to plug in between the wall socket and appliance cord. Some installations proved to be more difficult than expected with the wall mounted microwave and oven having to be partly disassembled, as seen in figure 4.2b and 4.2d.

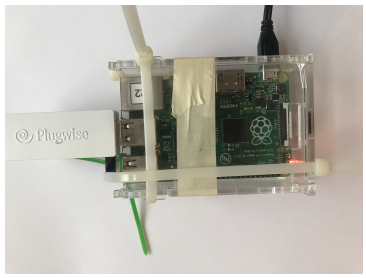
4.3 Raspberry Pi

The Raspberry Pi was connected to power supply, a WiFi dongle and the Plugwise Stick, see figure 4.2c The power supply was a certified microUSB power supply that delivered the needed voltage. This was important so that the Raspberry Pi's performance was not throttled by insufficient voltage.



(a) Smart meter with EcoMonitor connected.

(b) Removing microwave and oven to install smart plugs.



(c) Raspberry Pi B+ in case with Plugwise Stick connected in the top USB-socket

(d) Plugwise Circle connected to an electric kettle.

Figure 4.2: Installing the hardware

4.3.1 Raspberian Jessie

The newest versions at the time of writing, had problems connecting Plugwise-2-py to node-RED. The complications are most likely due to updated packages handling web sockets, which the Plugwise-2-py does not support. To get around this, the Raspberian Jessie version from 2017-07-05 was chosen and flashed onto the microSD card with Etcher[38].

4.3.2 Plugwise-2-py

Installing the Plugwise-2-py python package is done by following the instructions in the GitHub repo. Be sure to install it in the same path as described, making the installation much easier. The rate of data pulling was changed to every two seconds to match data provided by the Kaifa smart meter.

Mosquitto

When the active power has been received in Plugwise-2-py, it needed to be stored. The software already includes *Message Queuing Telemetry Transport* (MQTT)[39], a protocol used for sending sensor data in a publish-subscribe-based fashion, and the library *Mosquitto* was hence installed to broadcast messages in the local network.

Web Interface

When the Mosquitto software is installed and the Plugwise-2-py is running, the web interface, see figure 4.3, will show the real-time active power used by each appliance. This feature of the Plugwise-2-py software was helpful to see when the beginning data stream is working.

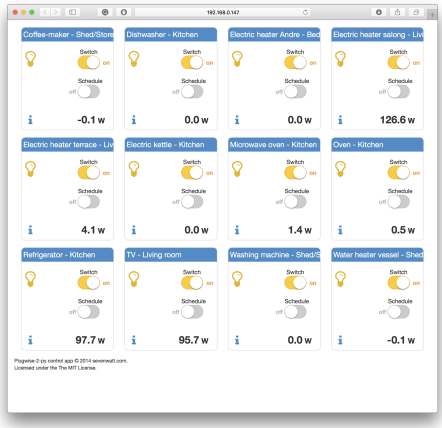


Figure 4.3: Screenshot of the Plugwise-2-py web interface

4.3.3 Node-RED

The also included interface to Domoticz[40] was at the beginning of the project viewed as a viable way of collecting the smart plug data, but showed to be insufficient because of its long log intervals of 5 minutes. A direct save-to-file function in Node-RED was used instead, to store the smart plug data. Note that the connection to Domoticz in Node-RED was kept and is visible in the screenshot in figure 4.4. This was to quickly be able to check the data whilst collecting and storing it in separate CSV-files.

4.4 Data Formatting

In this experiment, the data was formatted to fit the NILM-Eval framework which in turn is based on the ECO dataset[7]. The structure for the dataset is found in both NILM-Eval[9] and Appendix B and is brushed upon in figure 3.6 in the last chapter. The formatting itself

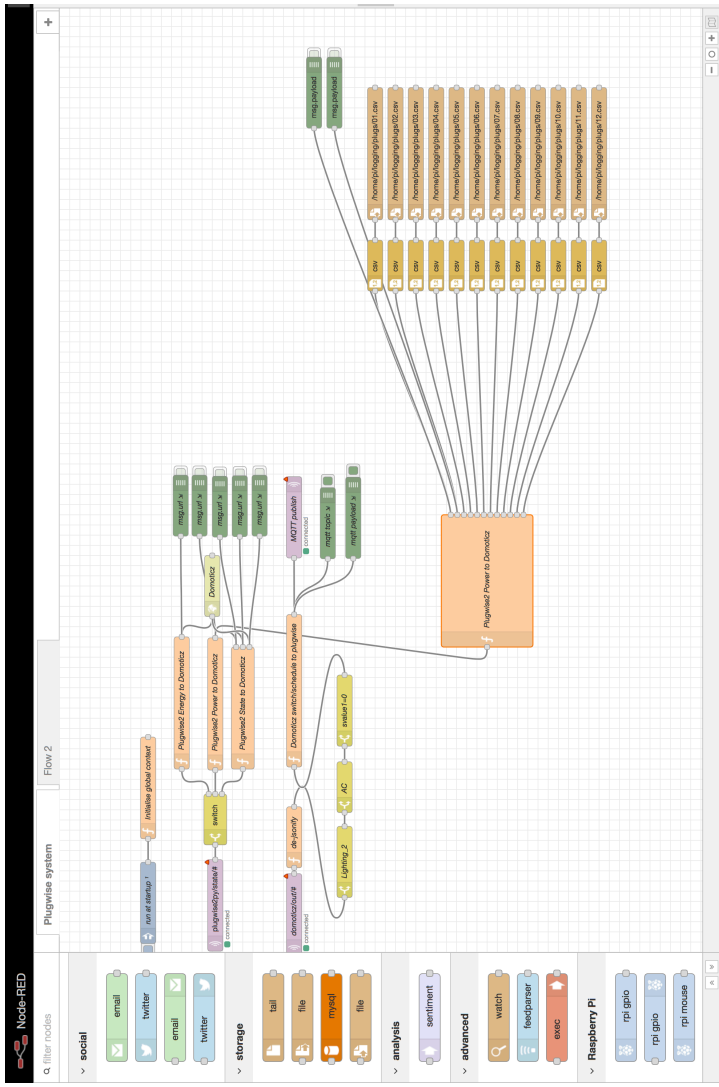


Figure 4.4: Screenshot of the setup in Node-RED. Orange rectangles are function blocks written in javascript to filter process-data before storing it in separate CSV-files.

was done in three different steps: smart plug, smart meter, and combining those two to fit the Matlab structure.

4.5 NILM-Eval

After copying the formatted files to NILM-Eval, a few config-files declaring the name of appliances, thresholds, phases, and households was tailored for the gathered data. These

files and other alterations are covered in the guide and will be discussed in the discussion.

Chapter 5

Results

This chapter presents the results of the experiment, the data collected, and finally, the identification and classification of appliances with the algorithms as presented forth in chapter two.

5.1 Evaluation of the experiment

The system was implemented as planned and proved successful after the challenges described in chapter 4 and Appendix A were solved. The data collected with the Kaifa smart meter was not as expected. The voltage on phase 2 was measured as zero during the whole experiment, and a solution was implemented by creating U2 based on U1 and U3. This will be discussed in chapter 6.

The algorithms were trained on the first week of data and tested on the following 18 days. The data collected was dense with only a few hours of aggregated data missing in the training period. The algorithm tuning was done to the best effort with the time available but could be improved.

5.2 Usability of the Weiss algorithm

The Weiss algorithm proved difficult to use due to a lack of generalizations in the code. E.g., the names of each appliance had to be manually written in different Matlab-files to calculate and plot their consumption, and several errors occurred concerning matrices and lists not being presumed sizes. The metrics seemed to suffer from these errors as well, reporting high F-scores without evidence of this in the plots. This was not a problem when testing the algorithm on the ECO dataset[7], showing that work needs to be done on improving the NILM-Eval software when using new data. Therefore, the results of NILM with the Weiss algorithm fell victim to poor implementation and deliver limited information of its usability. Fortunately, some interesting plots and aspects were uncovered. These will be presented in this section with a focus on visual inspection.

5.2.1 Finding the Signatures

The first step in Weiss is to train on the plug data to find signatures. In this run, a threshold of 5W in the plug data was set as the limit to classify switching events. Weiss does not read the reactive data directly from data but calculates it using the phase shift caused by appliance loads. These being unavailable, the signatures created were one-dimensional, as seen in 5.1. What is worth noticing is that the signatures are reasonably separated on the one dimension available.

Appliance	Active power	Reactive power	Phase
Dishwasher	-2132.45	0.00	1
Dishwasher	-2137.94	0.00	1
Washing Machine	-605.98	0.00	1
Washing Machine	-712.11	0.00	1
Water Heater	-1920.23	0.00	2
Water Heater	1910.25	0.00	2
Oven	-1985.49	0.00	2
Oven	2085.68	0.00	2
TV	-44.44	0.00	2
TV	85.89	0.00	2
Electric Kettle	-2071.35	0.00	2
Electric Kettle	-1246.56	0.00	2
Electric Kettle	2110.84	0.00	2
Microwave Oven	-1714.72	0.00	2
Microwave Oven	1780.44	0.00	2
Electric Heater Bedroom	-948.47	0.00	3
Electric Heater Bedroom	957.78	0.00	3
Coffee Maker	-1647.46	0.00	3
Coffee Maker	1420.00	0.00	3
Electric Heater Salong	-956.54	0.00	3
Electric Heater Salong	956.72	0.00	3
Electric Heater Terrace	-954.25	0.00	3
Electric Heater Terrace	953.84	0.00	3
Fridge	-70.47	0.00	3
Fridge	104.69	0.00	3

Table 5.1: Signature database made by Weiss.

5.2.2 Fridge and TV

Although the algorithm could not be used to evaluate all the appliances, the Fridge and TV produced plotted results. The Fridge is a Class 1 appliance with a clear pattern, and the TV is Class 3 without a clear pattern, these provide a level of insight.

Fridge

The Fridge, seen in figure 5.1 and 5.2, is quite well captured by the Weiss algorithm. Although the detection is shifted or missing in some cases, the algorithm is able to capture a fair amount of events. This could be due to the Fridge always being ON, making it harder for it to completely miss an event such as in the case of the TV.

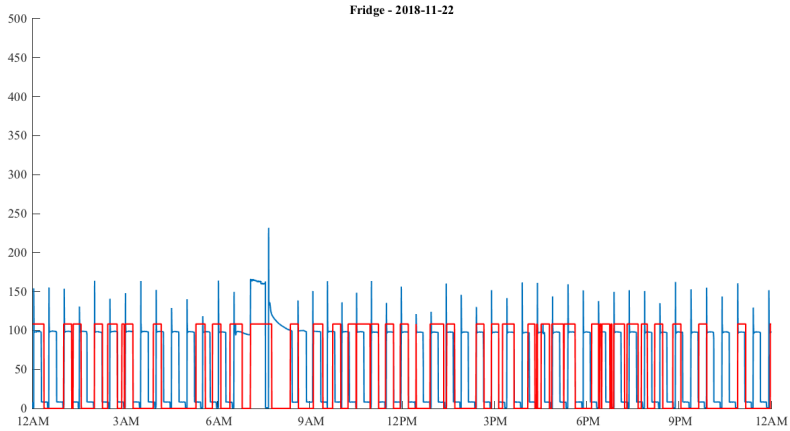


Figure 5.1: Weiss: Fridge consumption detected on day 18, showed against the plug data.

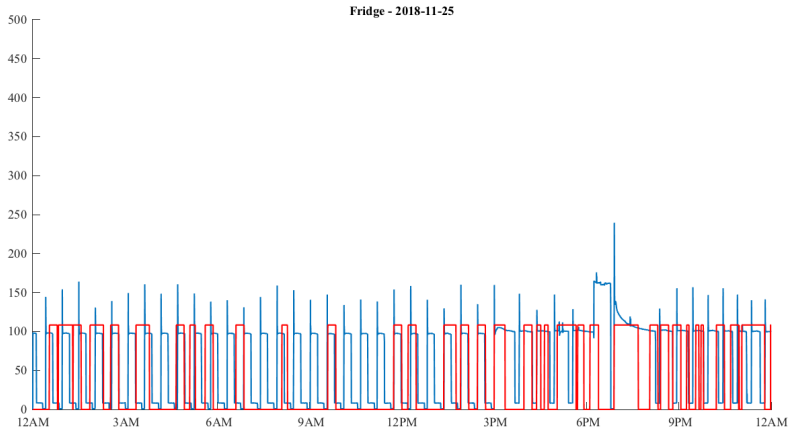


Figure 5.2: Weiss: Fridge consumption detected on day 21, showed against the plug data.

TV

The identification of the TV's events was poor, see figure 5.4. While the state change from OFF to ON is sometimes detected, there are far more false positives. This is not very surprising due to the TV drawing a much smaller active power load than other appliances connected, such as the three Electric Heaters which are also Class 3. The variations in the three Electric Heaters in its ON-state are smaller than its initial power change when turned ON, see figure 5.5. This might a contributing factor to why the identification of a smaller device, such as the TV, might get influenced.

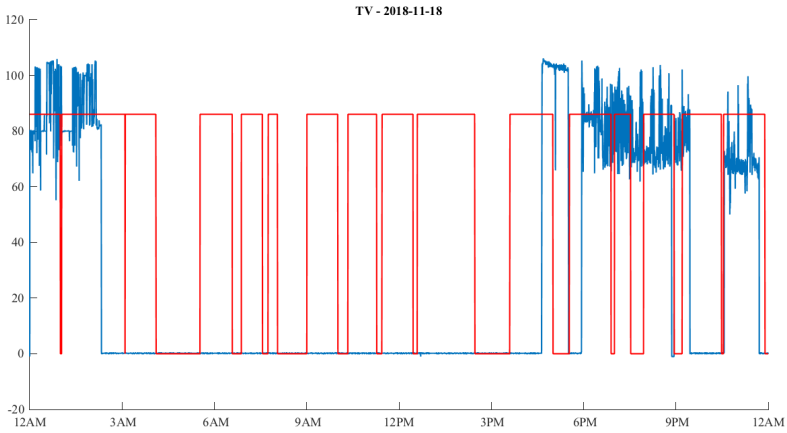


Figure 5.3: Weiss: TV consumption detected on day 14, showed against the plug data.

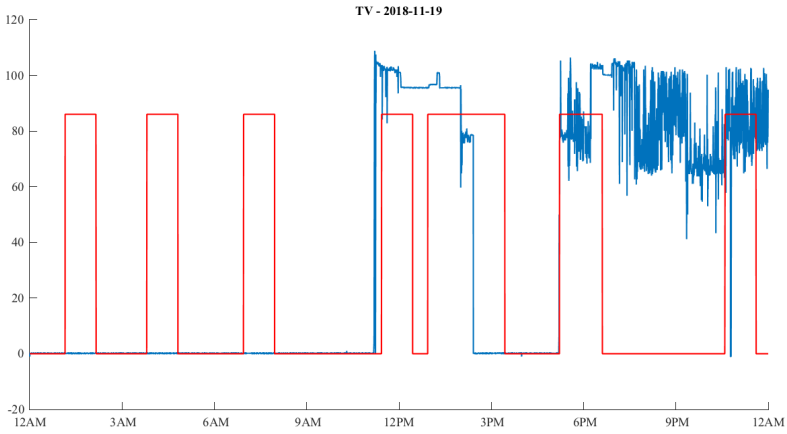


Figure 5.4: Weiss: TV consumption detected on day 15, showed against the plug data.

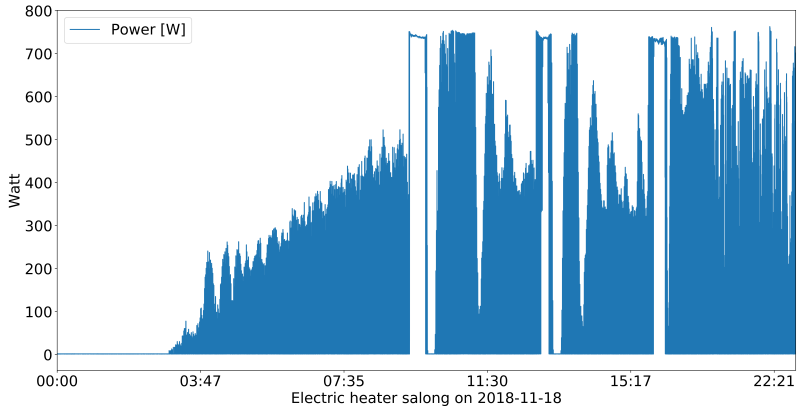


Figure 5.5: Electric Heater Salong consumption during day 14. One of three electric heaters monitored in the experiment.

5.2.3 Parameter Sensitivity

Weiss contains many different parameters to tune. The most important ones to start with are the evaluation thresholds (To not let ground errors interfere.) and event thresholds found in the configuration files. When running individual test to identify appliances, the parameters shown below proved decisive.

- r : Scaling parameter indicating the maximum distance of one event to the nearest signature.
- osc : Scaling parameter for oscillations
- maxEventDuration : The maximal duration an event lasts

5.2.4 Overall Performance

Overall, the Weiss algorithm shows promise due to its reasonably spaced signatures mapped. The results shown for the Fridge indicates that it is possible to achieve NILM for Class 1 appliances. The Class 3 appliance TV will have to be tuned more to conclude the possibilities.

5.3 Baranski and Voss

The approach in Baranski is to cluster frequently occurring emerging patterns in the aggregated data. With its design, the smaller appliances will not be prioritized. This is apparent in the results as almost all the clusters found contained the three Electric Heaters. The load caused by the heaters are both heavy and frequently occurring, leaving the infrequent

and rare appliances in the shadow. To get around this problem, the number of clusters was increased from the recommended 20 to 50, and the threshold of the events was adjusted in the interval [50W,500W] before setting it to 150W. The results were still mostly unchanged with the electric heaters taking up all the space as seen in table 5.4. The FSMs are consequently only electric heaters, as seen in table 5.5.

A theory to why the Electric Heater Salong appears more often is that because the Electric Heater Terrace is placed under a wall with more windows (see floor plan 3.1) it is therefore constantly running on maximum capacity, compared to Electric Heater Salong which covers a smaller area with with new, recently installed windows. The FSMs are overlapping, not knowing which of the electric heaters are turned ON or OFF. The confusion is understandable since both of them are same make and model, and they could all be classified as one oven in a heating category on the bill. The appliances are therefore not seperatable.

Cluster	ΔP	Size	App. 1	%	App. 2	%
C1	-743	13698	Electric Heater Salong	0.61	Electric Heater Terrace	0.02
C2	752	9687	Electric Heater Salong	0.66	Electric Heater Terrace	0.01
C3	-744	4145	Electric Heater Salong	0.51	Electric Heater Terrace	0.05
C4	748	3910	Electric Heater Salong	0.79	Electric Heater Terrace	0.01
C5	719	3141	Electric Heater Salong	0.58	Electric Heater Terrace	0.01
C6	-641	2589	Electric Heater Salong	0.61	Electric Heater Terrace	0.01
C7	77	2353	Electric Heater Salong	0.71	Electric Heater Terrace	0.02
C8	-998	2191	Electric Heater Terrace	0.59	Electric Heater Salong	0.03
C9	649	1927	Electric Heater Salong	0.63	Electric Heater Terrace	0.01
C10	-65	1624	Electric Heater Salong	0.68	Electric Heater Terrace	0.04
C11	1731	1294	Electric Heater Salong	0.20	Electric Heater Terrace	0.10
C12	1024	1186	Electric Heater Terrace	0.33	Electric Heater Salong	0.09
C13	1032	1044	Electric Heater Salong	0.13	Electric Heater Terrace	0.06
C14	1710	983	Electric Heater Terrace	0.19	Electric Heater Salong	0.08
C15	1719	964	Electric Heater Salong	0.16	Electric Heater Terrace	0.15
C16	-1702	644	Electric Heater Salong	0.17	Electric Heater Terrace	0.03
C17	-1013	617	Electric Heater Terrace	0.12	Electric Heater Salong	0.11
C18	-1687	589	Electric Heater Salong	0.12	Electric Heater Terrace	0.03
C19	-1632	571	Electric Heater Salong	0.14	Electric Heater Terrace	0.02
C20	-2072	526	Electric Heater Salong	0.17	Water Heater	0.11

Table 5.2: Barinski clusters when run with 20 clusters and a threshold of 150W.

5.3.1 Parameter Sensitivity

In this experiment, the weights were not adjusted, but set to 1, due to the long algorithm runtime. These weights adjust the length of the switching event, the electricity consumption boost, and a more thorough test need to be conducted to map the full potential of the algorithm. The maximum number of states per appliance was also set to 2, as recom-

FSM	ΔP (C1)	ΔP(C1)	Appliance
1	748	-744	Electric Heater Salong
2	1710	-1702	Electric Heater Terrace or Electric Heater Salong
3	1719	-1702	Electric Heater Terrace or Electric Heater Salong
4	1024	-1013	Electric Heater Terrace or Electric Heater Salong
5	1710	-1687	Electric Heater Terrace or Electric Heater Salong
6	649	-641	Electric Heater Salong
7	1032	-1013	Electric Heater Terrace or Electric Heater Salong
8	1719	-1687	Electric Heater Terrace or Electric Heater Salong
9	1731	-1702	Electric Heater Terrace or Electric Heater Salong
10	1731	-1687	Electric Heater Terrace or Electric Heater Salong
11	1024	-998	Electric Heater Terrace
13	1032	-998	Electric Heater Terrace or Electric Heater Salong
14	1710	-1632	Electric Heater Terrace or Electric Heater Salong
15	1719	-1632	Electric Heater Terrace or Electric Heater Salong
16	1731	-1632	Electric Heater Terrace or Electric Heater Salong
17	752	-743	Electric Heater Salong
18	752	-744	Electric Heater Salong
19	719	-641	Electric Heater Salong
20	77	-65	Electric Heater Salong

Table 5.3: Barinski 20 FSM for 20 clusters: Here it is clear that the mapped appliances are wildly crowded in the same clusters. Number 12 was not produced by the algorithm for unknown reasons.

mended, meaning that each appliance only has two states. This choice also contributes to the multiple clusters of individual electric heaters.

The most important parameters to tune were found to be:

- numOfClusters : The number of clusters.
- threshold : The threshold for detecting events.
- maxNumOfStates : Number of states possible per appliance.

5.3.2 Overall Performance

Overall, the Barinski algorithm shows little promise in detecting other appliances than the electric heaters with the tuning used in this experiment. The FSMs found could not be used to classify a spesific appliance, only This will be discussed in chapter 6.

5.4 Parson

With the key feature being recognizing patterns, and previous work including a fridge with freezer and a microwave oven, these appliances were chosen for testing. While these are appliances which have clear patterns of state changes throughout the day, one would

Cluster	ΔP	Size	App. 1	%	App. 2	%
C1	-751	8339	Electric Heater Salong	0.63	Electric Heater Terrace	0.01
C2	755	6308	Electric Heater Salong	0.68	Electric Heater Terrace	0.01
C3	736	4770	Electric Heater Salong	0.63	Electric Heater Terrace	0.00
C4	-729	4357	Electric Heater Salong	0.63	Electric Heater Terrace	0.00
C5	-678	1785	Electric Heater Salong	0.62	Electric Heater Terrace	0.01
C6	692	1728	Electric Heater Salong	0.59	Electric Heater Terrace	0.01
C7	-1004	1714	Electric Heater Terrace	0.62	Electric Heater Salong	0.03
C8	-616	1409	Electric Heater Salong	0.62	Electric Heater Terrace	0.02
C9	-741	1298	Electric Heater Salong	0.57	Electric Heater Terrace	0.01
C10	632	1271	Electric Heater Salong	0.60	Electric Heater Terrace	0.00
C11	747	1210	Electric Heater Salong	0.75	Electric Heater Terrace	0.01
C12	39	1087	Electric Heater Salong	0.82	Electric Heater Terrace	0.01
C13	749	1067	Electric Heater Salong	0.82	Electric Heater Terrace	0.00
C14	-751	1008	Electric Heater Salong	0.46	Electric Heater Terrace	0.08
C15	750	982	Electric Heater Salong	0.81	Electric Heater Terrace	0.02
C16	755	973	Electric Heater Salong	0.72	Electric Heater Terrace	0.05
C17	-744	880	Electric Heater Salong	0.52	Electric Heater Terrace	0.01
C18	-747	873	Electric Heater Salong	0.51	Electric Heater Terrace	0.05
C19	-29	789	Electric Heater Salong	0.83	Electric Heater Terrace	0.02
C20	1016	734	Electric Heater Terrace	0.39	Electric Heater Salong	0.08
C21	48	705	Electric Heater Salong	0.55	Electric Heater Terrace	0.02
C22	1740	643	Electric Heater Salong	0.18	Electric Heater Terrace	0.11
C23	1735	551	Electric Heater Terrace	0.16	Electric Heater Salong	0.15
C24	-1009	549	Electric Heater Terrace	0.53	Electric Heater Salong	0.05
C25	1024	471	Electric Heater Salong	0.14	Electric Heater Terrace	0.09
C26	1007	465	Electric Heater Salong	0.11	Electric Heater Terrace	0.08
C27	-898	464	Electric Heater Terrace	0.35	Electric Heater Salong	0.09
C28	2237	463	Electric Heater Salong	0.22	Water Heater	0.08
C29	-126	420	Electric Heater Salong	0.52	Electric Heater Terrace	0.07
C30	-1720	418	Electric Heater Salong	0.16	Electric Heater Terrace	0.03
C31	247	407	Electric Heater Salong	0.67	Electric Heater Terrace	0.02
C32	-73	406	Electric Heater Salong	0.56	Electric Heater Terrace	0.04
C33	951	390	Electric Heater Terrace	0.27	Electric Heater Salong	0.19
C34	-1716	371	Electric Heater Salong	0.18	Electric Heater Terrace	0.03
C35	1011	346	Electric Heater Salong	0.13	Electric Heater Terrace	0.05
C36	1735	341	Electric Heater Terrace	0.21	Electric Heater Salong	0.06
C37	-1743	338	Electric Heater Salong	0.17	Water Heater	0.15
C38	1663	325	Electric Heater Salong	0.22	Electric Heater Terrace	0.12
C39	1731	324	Electric Heater Terrace	0.23	Water Heater	0.01
C40	1734	321	Electric Heater Salong	0.18	Electric Heater Terrace	0.13
C41	487	304	Electric Heater Salong	0.68	Electric Heater Terrace	0.03
C42	1734	303	Electric Heater Salong	0.16	Electric Heater Terrace	0.15
C43	-1010	292	Electric Heater Terrace	0.18	Electric Heater Salong	0.08
C44	-1712	280	Electric Heater Salong	0.08	Electric Heater Terrace	0.04
C45	-1705	257	Electric Heater Salong	0.15	Electric Heater Terrace	0.03
C46	-1690	244	Electric Heater Salong	0.15	Electric Heater Terrace	0.03
C47	-1026	223	Electric Heater Salong	0.12	Electric Heater Terrace	0.05
C48	-988	180	Electric Heater Salong	0.13	Water Heater	0.01
C49	-2708	170	Electric Heater Salong	0.11	Electric Heater Terrace	0.03
C50	-2186	130	Electric Heater Salong	0.26	Dishwasher	0.06

Table 5.4: Barinski clusters when run with 50 clusters and a threshold of 150W.

FSM	ΔP (C1)	ΔP(C1)	Appliance
1	750	-751	Electric Heater Salong
2	1011	-1010	Electric Heater Terrace or Electric Heater Salong
3	1007	-1009	Electric Heater Salong or Electric Heater Terrace
4	749	-751	Electric Heater Salong
5	1735	-1743	Electric Heater Terrace or Electric Heater Salong or Water Heater
6	1734	-1743	Electric Heater Terrace or Electric Heater Salong or Water Heater
7	1011	-1009	Electric Heater Terrace or Electric Heater Salong
8	1734	-1743	Electric Heater Salong or Electric Heater Terrace
9	1024	-1026	Electric Heater Salong or Electric Heater Terrace
10	1007	-1010	Electric Heater Salong or Electric Heater Terrace
11	755	-751	Electric Heater Salong
12	747	-747	Electric Heater Salong
13	749	-747	Electric Heater Salong
14	750	-747	Electric Heater Salong
15	1740	-1743	Electric Heater Terrace or Electric Heater Salong or Water Heater
17	1731	-1720	Electric Heater Terrace or Electric Heater Salong
18	1735	-1743	Electric Heater Terrace or Electric Heater Salong or Water Heater
19	747	-741	Electric Heater Salong
20	747	-751	Electric Heater Salong

Table 5.5: Barinski 20 FSM for 50 clusters: Here it is clear that the mapped appliances are crowded in the same clusters. Number 16 was not produced by the algorithm for unknown reasons.

also want to identify other appliances as well. The author, therefore, looked at the Fridge (Inductive, Class 1), Water Heater (Resistive, Class 1), Microwave Oven(inductive, Class 3) and Electric Heater Terrace (Resistive, Class 3).

As explained in chapter 3, the Parson algorithm needs to know the mean consumption of the appliance being considered. There are three ways of finding these, manually, by training on aggregated data or by training on plug data. NILM-Eval does not produce plotted results when training, but provide a list of means and variances found to describe the pattern best. These means and variances did not always give the best result when testing and proved difficult to understand. The results below are therefore produced by first manually inspecting the plug data for means and variances, then trained, before combined to find the best results.

In general, the metrics are a good place to start for measuring the result of a test, but they do not tell the whole story. As seen in the figures below, the F-score might catch the total consumption to a degree, but the misclassified state changes are concerning.

5.4.1 Fridge

As seen in figure 5.6, the Fridge consumption is mostly correct, but the events are not covered. In some cases, the algorithm overfits and end up saying that the Fridge switch states multiple time during only one state, as seen in figure 5.7.

The actual Fridge pattern from plug data mapped against aggregated consumption. The pattern is clearly repetitive and should be simple for the algorithm to map according to its approach, but this is not the case as shown in figure 5.8 and 5.9.

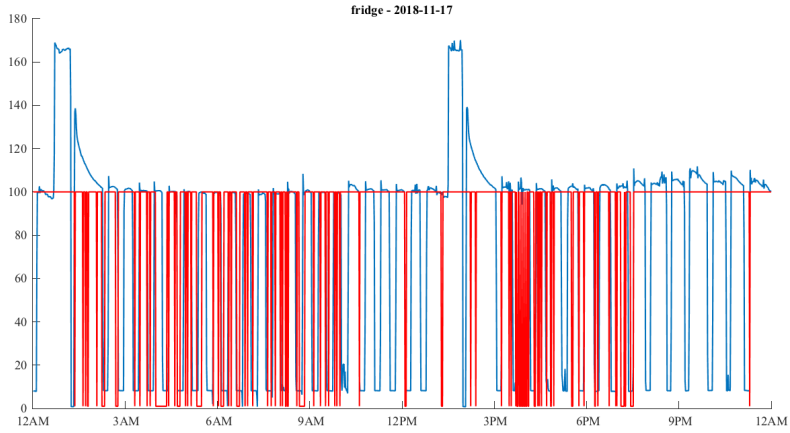


Figure 5.6: Parson: Fridge consumption found on day 13.

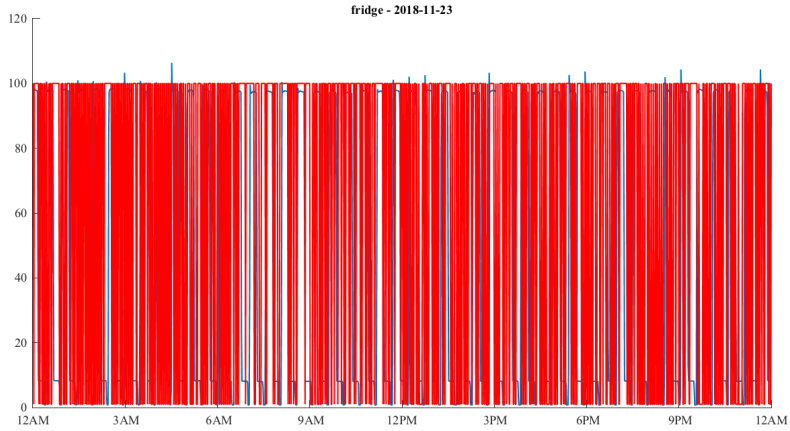


Figure 5.7: Parson: Fridge consumption found on day 19.

5.4.2 Water Heater

The Water Heater has a somewhat clean state change as seen in figure 5.18. The consumption mean and variance was set to 1900W after inspecting the raw data and consulting the

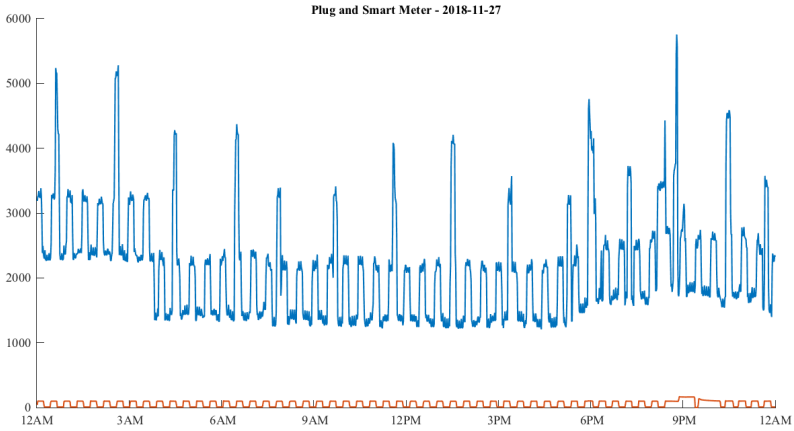


Figure 5.8: Parson: Actual consumption by Fridge against aggregated data found on day 23.

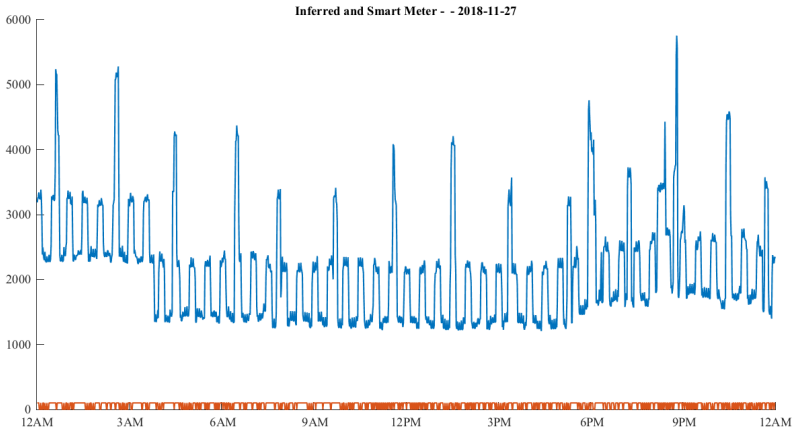


Figure 5.9: Parson: Identified Fridge consumption against aggregated data found on day 23.

algorithm training. When using a granularity of 60, the whole ON-state is mostly down sampled to approximately 10 data points, possibly creating problems. The probabilities in the Hidden Markov Model must be well tuned as well for the Parson to discover the pattern.

As seen in figure 5.18, Parson shows promise at the beginning of the run, detecting the water heater in both short and longer periods of consumption, but fails to classify 10 out of 17 events while misclassifying one. The following day, seen in figure 5.19, the same results can be found. Here, Parson works well picking up the rater long semi-steady state between approximately 1 PM and 6 PM. Later in the experiment on day 20, seen in figure

5.20, Parson mostly produce false positives. On day 21, seen in figure 5.21, Parson not only misclassifies several events not caused by the water heater but also combine six state changes as one long event.

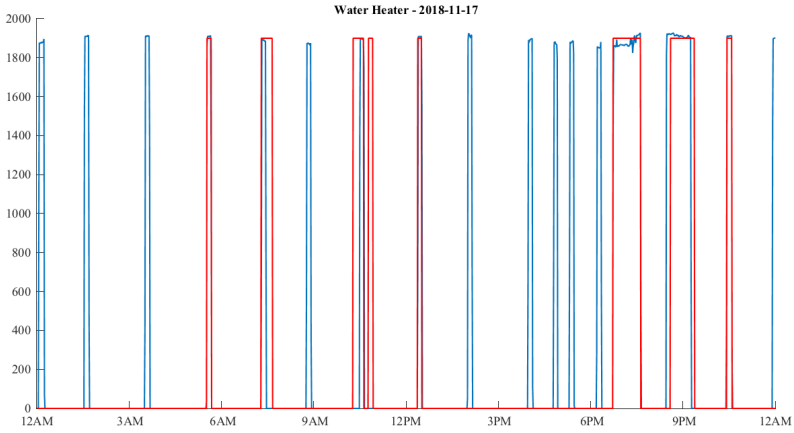


Figure 5.10: Parson Water Heater consumption and detection on day 13.

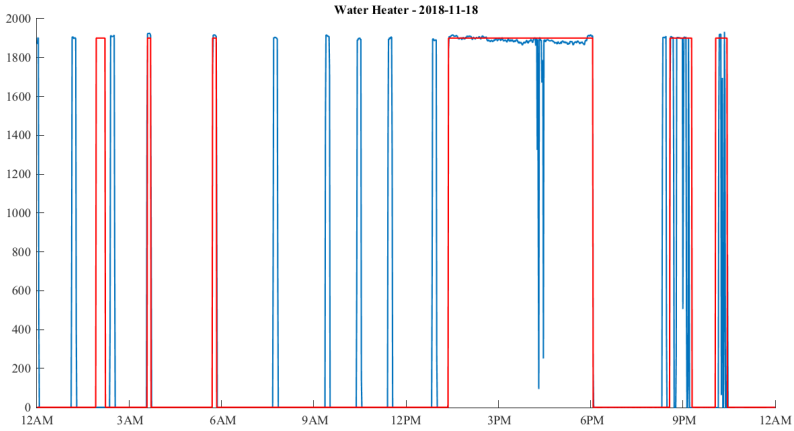


Figure 5.11: Parson: Water Heater consumption and detection on day 14.

5.4.3 Microwave oven

In contrast to the Fridge, the Microwave Oven's load is irregular, and the duration of the ON-state is short. The Microwave Oven has a significant load depending on the settings

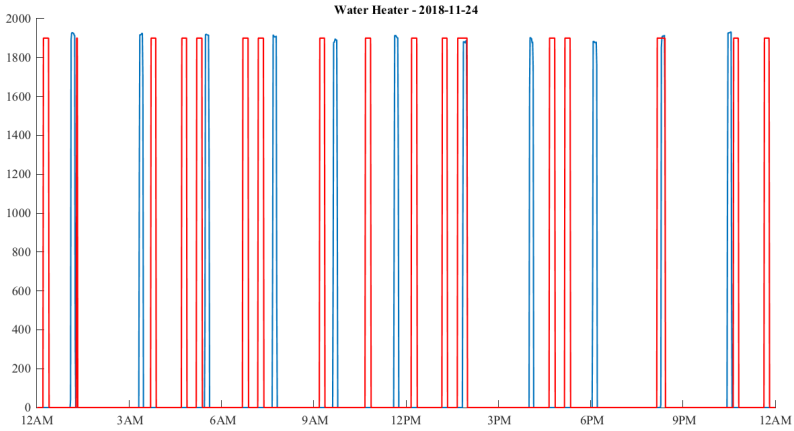


Figure 5.12: Parson: Water Heater consumption and detection on day 20.

Figure 5.13: Parson: Water Heater consumption and detection on day 21

chosen by the user, with an additional heating element for alternative heating. This places it in between Class 2 and 3, and the many operational modes create a larger variance, as seen in figure 5.15. Training on its pattern is hence hard, as seen in the misclassifications in figure 5.14.

The pattern is not repetitive will be hard for the algorithm to map, as seen in figure 5.16.

As seen in figure 5.17, Parson both miss the actual events and misclassifies two other events.

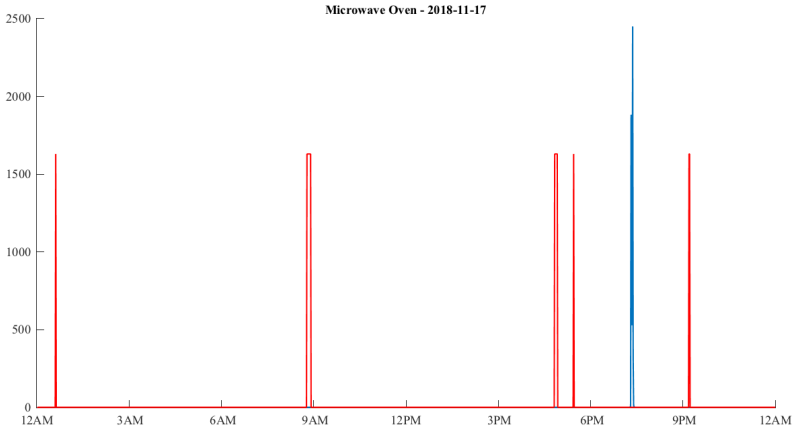


Figure 5.14: Parson: Microwave Oven consumption and detection on day 13.

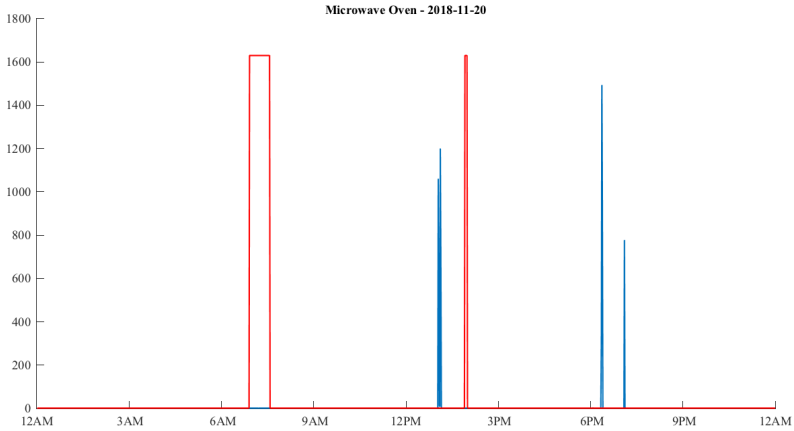


Figure 5.15: Parson: Microwave Oven consumption and detection on day 16.

5.4.4 Electric Heater Terrace

Using Parson for multi-state appliances in Class 2 is difficult, resulting in appliances in Class 3, such as the Electric Heater Terrace, being more difficult. The Electric Heater Terrace itself has many different states due to the thermostat producing different gains in consumption and is essentially controlled by the weather. In the first seven days of the data, the time that is used for training, the weather was warmer than the rest of the period resulting in the Electric Heater Terrace not drawing as much power as possible. In figure 5.18 the reader can see the consumption throughout day 12 when the weather was warmer. Here, the plug-training suggested a mean of 560W for the ON-state, which can

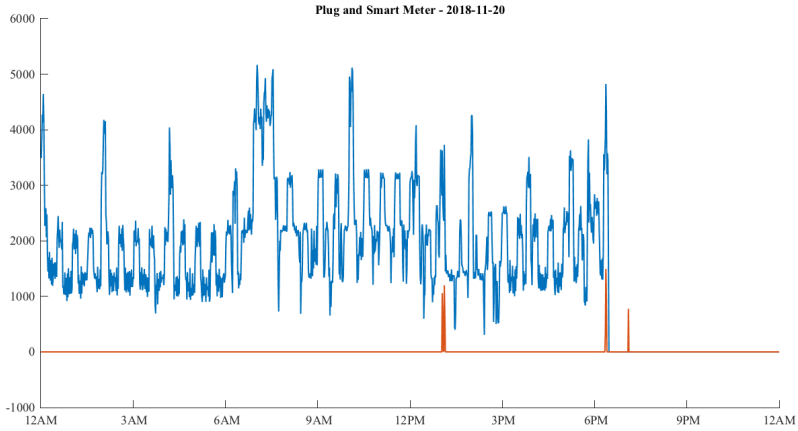


Figure 5.16: Parson: The actual Microwave Oven pattern from plug data mapped against aggregated consumption on day 16.

Figure 5.17: Parson: Identified Microwave Oven pattern and aggregated data on day 16.

be seen to fit rather well in general but increasing false state changes. In figure 5.20, the consumption during the colder day 19, the mean is not fitting the consumption anymore. Figure 5.19 and figure 5.21 shows the same two days with the mean for the ON-state set to 1000W. Here, the state changes are covered to a higher degree, but the consumption is most likely overestimated due to the high variance. When the mean for ON-state is set to 1000W, Parson still does not classify the events correctly, though it has less misclassified step changes.

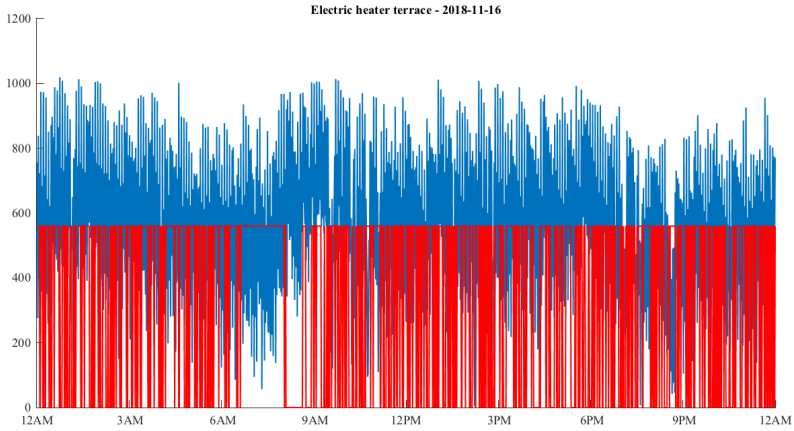


Figure 5.18: Parson: Electric Heater Terrace consumption and detection on day 12 with a ON-state mean of 560W.

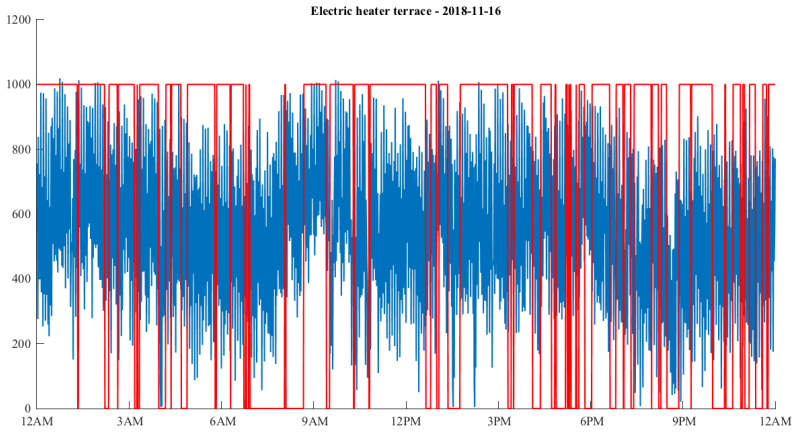


Figure 5.19: Parson: Electric Heater Terrace consumption and detection on day 12 with a ON-state mean of 1000W.

5.4.5 Parameter Sensitivity

The most important parameters to tune in the Parson algorithm were found to be:

- meanOn : An appliance's mean of Gaussian distribution describing the change in power when switched ON
- meanOff : An appliance's mean of Gaussian distribution describing the change in power when switched OFF

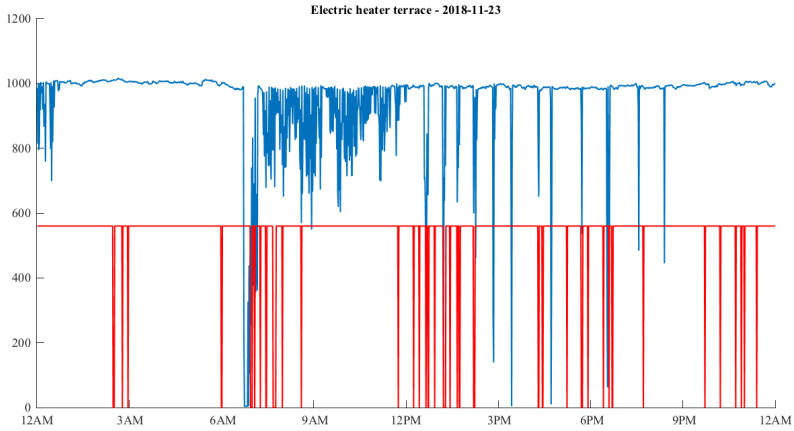


Figure 5.20: Parson: Electric Heater Terrace consumption and detection on day 19 with a ON-state mean of 560W.

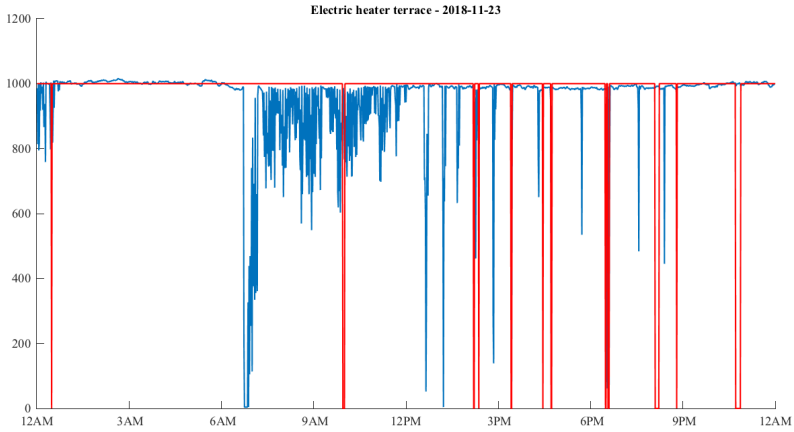


Figure 5.21: Parson: Electric Heater Terrace consumption and detection on day 19 with a ON-state mean of 1000W.

- varOn : An appliance's variance of Gaussian distribution describing the change in power consumption when switched ON
- varOff : An appliance's variance of Gaussian distribution describing the change in power consumption when switched OFF
- transOn : Transition probability from 'OFF' state to 'ON' state
- transOff: Transition probability from 'ON' state to 'OFF' state

- likThresh: Likelihood threshold

5.4.6 Overall Performance

Parson worked as expected with a heavy reliance on consistent patterns and constant means. When appliances had a considerable variation, the chance of wrongly estimating the consumption increased. Still, the algorithm's F-score is quite high, as seen in table 5.6. The Electric Heater has the highest score due to it being ON during most of the experiment.

Metrics and parameters	Fridge	Water Heater	Microwave Oven	Electric Oven Terrance (low mean)	Electric Oven Terrance (high mean)
F-Score	0.6602	0.4252	NaN	0.8278	0.8690
Precision	0.5786	0.3590	0	0.8005	0.8135
Recall	0.7685	0.5215	0	0.8571	0.9326
TPR	0.7685	0.5215	0	0.8571	0.9326
FPR	0.7117	0.1248	0	0.9278	0.9285
meanON [W]	100	1900	1630	560	1000
meanOFF [W]	1	0	1.35	0	0
varOn [W]	100	55	1000	350	350
varOff [W]	5	0	2	4	4

Table 5.6: Parson algorithm results.

Chapter 6

Discussion

There are several possible pitfalls related to all steps in conducting a feasibility study as done in this project. while it was possible to solve most of them during the implementation, while it was possible to solve most of them during the implementation. In this chapter, these challenges, along with the results found, will be discussed.

6.1 Evaluating the Experiment as a whole

The design and implementation of the experiment can overall be deemed successful, due to system functioning and the data being collected from the Kaifa smart meter and the individual appliances, but some aspects should be mentioned.

The data collected from the appliances was not as consistent as planned. Plugwise-2-Py was set to poll data every two seconds, but the frequency of data received ranged between two and seven seconds. The assumption that the data would be frequent enough due to previous research using the same hardware might not have been correct, or there might have been interference due to the WiFi dongle being connected directly beside the Plugwise Stick. Another possibility is that the previous researchers might have smoothed the data where the consumption was not read, something the author did in this experiment.

During the planning phase, it was much more time consuming than anticipated to find suitable hardware, ordering the products and shipping them. This delay resulted in a shorter analysis time making it challenging to tune the algorithms properly. The alterations that needed to be done in NILM-Eval for Weiss was affected as well.

6.2 Three phases and type of electric distribution to households

Much work was needed to understand the data and the reason it appeared as it did. The initial assumption was that Kaifa would deliver data according to the specifications set by

the Norwegian government. Although Kaifa did provide the data available, a challenge appeared in the way the household was connected to the grid and what data was readable.

Depending on the city's electric grid, there are different distribution systems for electricity. In the conducted experiment, the household was connected to a TT-system, see figure 6.1, complicating the usability of NILM in two ways: missing data on phase two voltage and separating phase current loads.

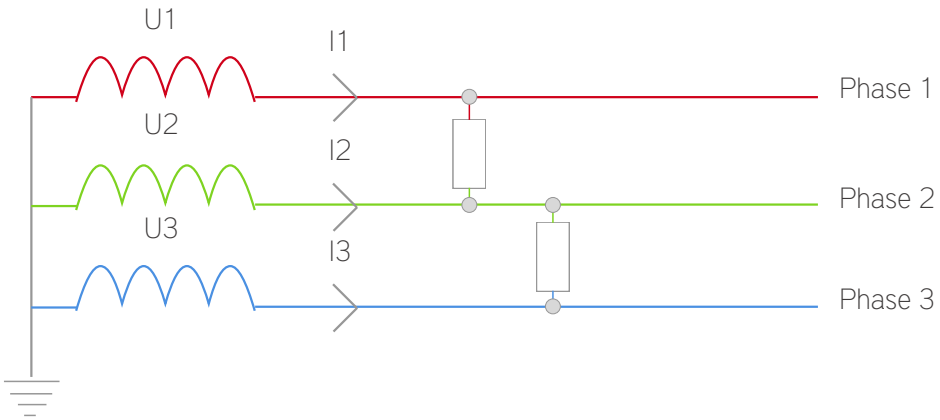


Figure 6.1: TT-System, same as used in household with load measured between two phases.

6.2.1 Zero volts measured on phase 2 voltage

Figure 6.2 shows how the voltage is set up. Here, the voltage on phase 1 and phase 3 deliver the normal 230V, with phase 2 being used as a neutral with 0V measured. Phase 2 is always zero because phase 1 and phase 3 are measured on opposite sides of phase 2, making it appear as a having no voltage. The missing data was solved in the experiment by giving U2 the average value of U1 and U3, which might not be the recommended way of solving it.

6.2.2 Separating phase current loads

With its design, the TT-system rises a new problem in the data, separating the appliance current loads. Typically, an appliance's current load would show on one phase, but in the case of this experiment, the appliance current load is observed in two, see figure 6.4. This is because the circuits in the household are connected between two phases in a TT-system, and with the Kaifa smart meter seemingly always showing the data as positive, the current load caused by one appliance is added to two phases. Figure 6.3 illustrates this in the household used in the experiment. The author's hypothesis is that if the household is connected to a TN- or TS-system, better results might be achievable. The reason behind this hypothesis is that if the household is connected to a distribution system where the phases are connected to a neutral line or ground, as shown in figure 6.5, the same would be

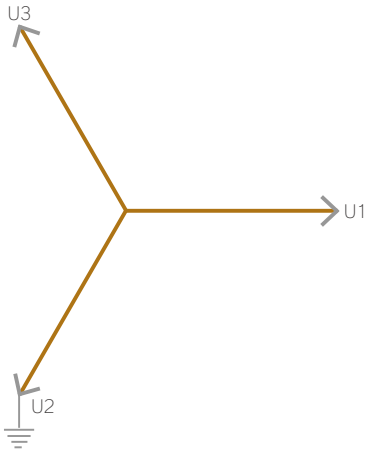


Figure 6.2: Voltage phases as read by Kaifa

true for the circuits in the household and the smart meter would then measure individual current loads on the phases.

The hypothesis was developed during the final days of the research period, leaving insufficient time to gather more data. However, the author did share this hypothesis in an online forum for Norwegian home automation[41], the author got in contact with a person from the electric service provider industry who had collected data from his smart meter on a different system than TT. As hypothesized, the smart meter data received indicates that each phase is measured individually, seen in figure 6.6.

The difference between the systems and the data which is then available raise an interesting question: Is the NILM potential for customers with a TN- or a TS-system larger than for customers with a TT-system? This is something worth looking into in future work.

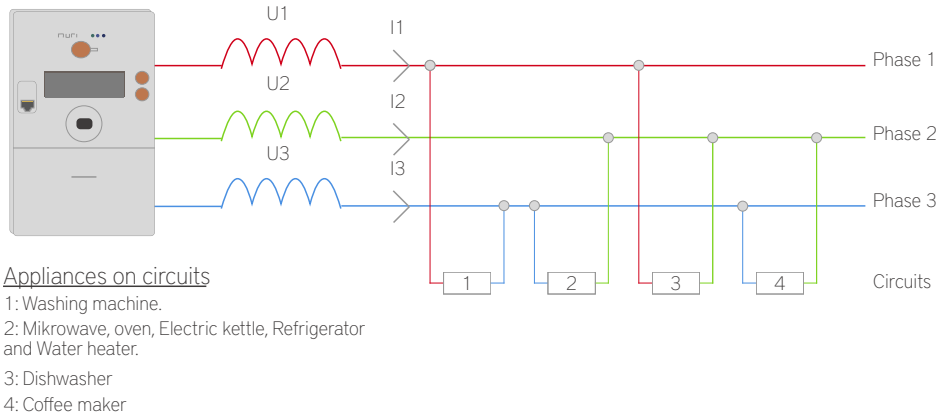


Figure 6.3: Circuits loads on phases in the experiment.

6.3 Weiss

Since the Weiss algorithm is based on active, reactive and distortive power, the data available from the Kaifa proved to be a problem. The active power is available every two seconds, but the phase-specific current and voltage usage are only readable every ten seconds. To account for this, the data was formatted so that the nine missing values in-between are copies of the last know value. The resolution is then in reality between 0.1hZ and 1hZ depending on when the event occurs.

In the supervised approach of Weiss, knowing the phase loads is part of creating signatures and event detection. With the voltage on phase 2(U2) always being zero as mentioned earlier, the resulting power on phase 2 would be calculated to zero (Power = current phase 2 + voltage phase 2). A simplified solution was to give U2 the average value between U1 and U3 (U2=(U1+U3)/2). While this temporarily satisfied the Weiss algorithm implementation, a second challenge arose.

There were in many errors when running Weiss with the data collected due to appliance-specified if-statements in the code. The calculated consumption varies vastly depending on the chosen appliance as some are set by a constant number and others calculated by Weiss, others were not supported due to them not being recognized as an appliance. With these challenges, the Weiss algorithm was not successfully evaluated, and a generalized version should be implemented in the NILM-Eval framework to properly test the Norwegian NILM potential. Most importantly, without measuring the current-voltage phase shift, the Weiss algorithm will still not be able to calculate reactive power by solely looking at its phase. If a new version is to be implemented, the possibility of somehow using the total reactive power read by Norwegian smart meters should be looked into.

6.4 Baranski

The results in chapter five showed that it could be difficult to find appliances such as the washing machines, coffee makers and other devices that are run occasionally, due to the electric heaters which are constantly ON. This problem could potentially be solved by more tuning and by filtering out the electric heaters after discovery, leaving the Baranski algorithm to look for remaining appliances.

6.5 Parson

In general, Parson shows some promise when calculating the consumption, but it falsely believes the states are changing when they are not. E.g., the fridge is changing states, turns ON and OFF, multiple times while it is actually in one state. The F-score is slightly fooled by the success of this, even though it only gets close to calculating the daily consumption by saying its always on. In this experiment, the plug training conducted by Parson did not always get close to the Gaussian spreads seen in the data, and further work and tuning could improve the results of the algorithm.

6.5.1 Training

The training period of this test was the first week, with the following 18 days as the test period. With more data, the training done by the algorithms might improve, and catch the more general signatures and means of the appliances, making them more robust.

6.6 Tuning

Due to the short time left in the research period, the tuning was not conducted as planned. Time permitting, a sensitivity analysis would have been done to map the important parameters properly. This should be done in future work before dismissing any of the tested approaches in this thesis.

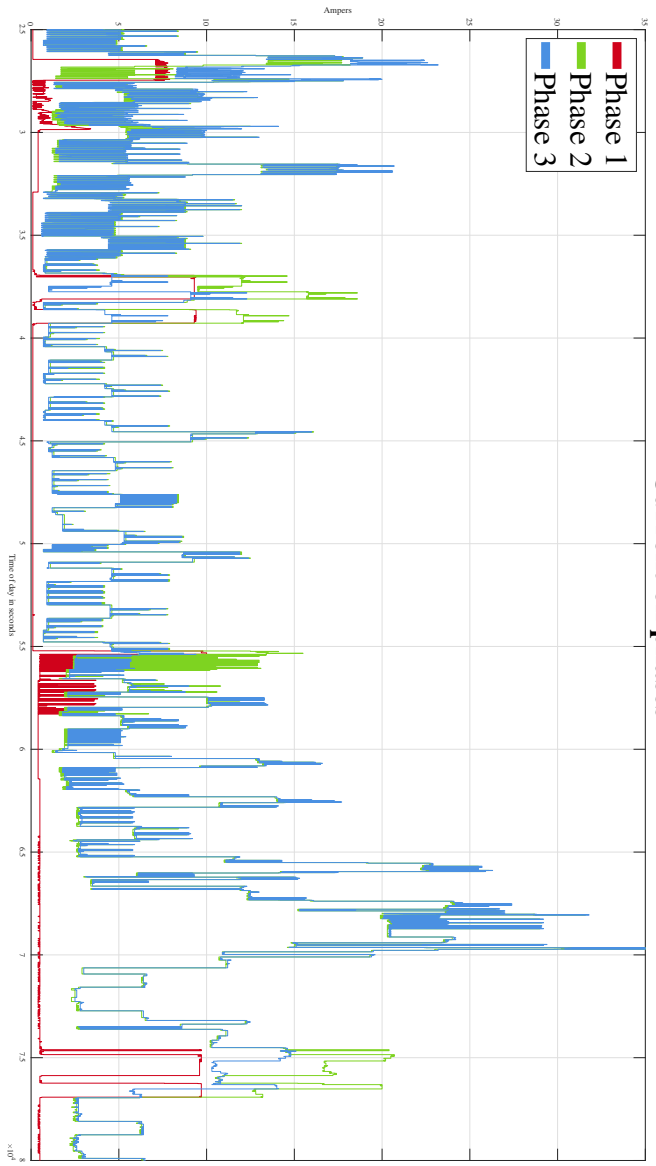


Figure 6.4: Current phases superimposed. Appliance loads are here shown to always affect two phases with the synchronous step changes.

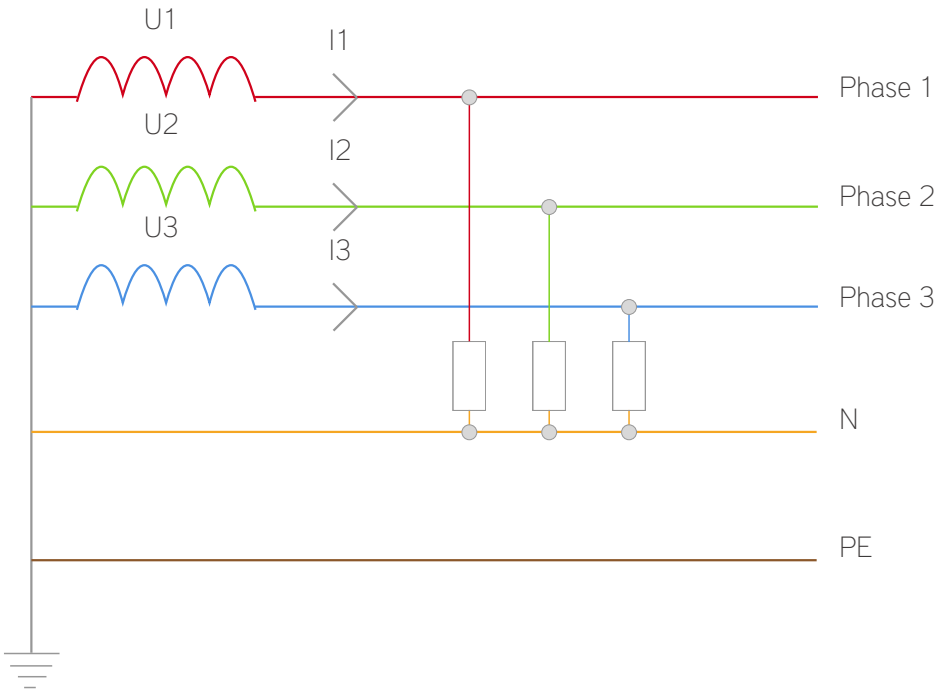


Figure 6.5: TN-System, a system where loads are measured between one phase and neutral.

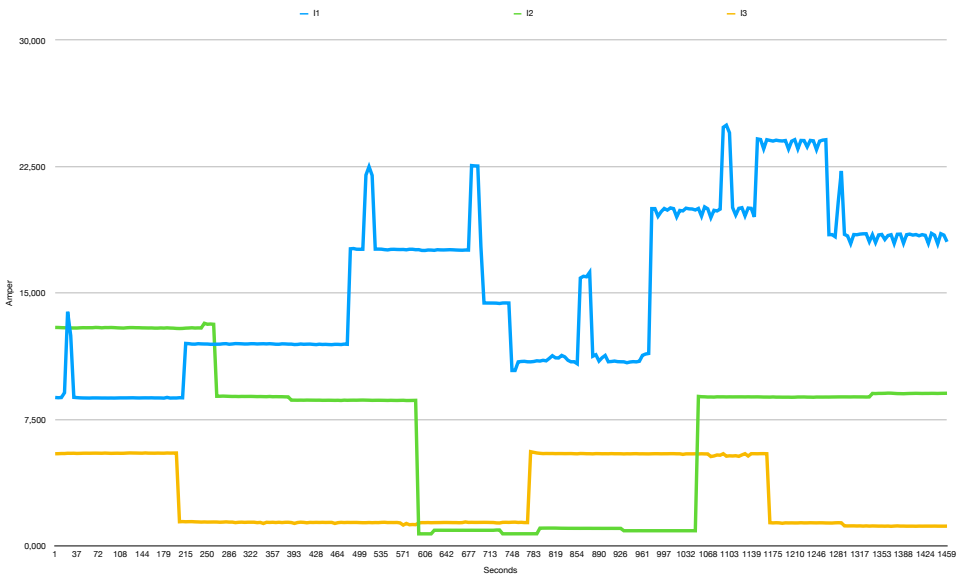


Figure 6.6: Current phases data from a different network than TT. The data given was not a long period, but the current phases seem to be measured individually as they are appearing independent of each other.

Chapter 7

Conclusion

7.1 Conclusions

The potential of NILM for Norwegian households using today's standard algorithms is present, but weak. This project's results show that contrarily to initial assumptions, the biggest challenge is not the low frequency of the smart meter data collected, but the lack of individual phase measurements and the electric heaters. To improve the results of this project, more tuning could be done, but the best solution would be to design a new NILM algorithm specially tailored for Norwegian smart meter data. To enable consumers to fully benefit from NILM, a substantial premade signature database is needed.

Bibliography

[1] Ministry of Petroleum and Energy. For-2017-12-14-2019, 2017. https://lovdata.no/forskrift/1999-03-11-301/\T1\textsection4-2.

[2] Lyse: Reasons for installing smart meters. https://www.lysenett.no/AMS-english/, 2018. Accessed: 2018-12-08.

[3] Surge pricing. https://www.distriktsenergi.no/artikler/2018/6/12/smarte-strommalere-gir-mer-rettferdig-strompris/, 2018. Accessed: 2018-11-27.

[4] John A. "Skip" Laitner Karen Ehrhardt-Martinez, Kat A. Donnelly. Advanced metering initiatives and residential feedback programs: A meta-review for household electricity-saving opportunities. https://aceee.org/research-report/e105, 2010. Accessed: 2018-10-09.

[5] Jr.Fred C. Schweppe George W. HartEdward C. Kern. U.s. patent 4,858,141. https://patents.google.com/patent/US4858141, 1986. Accessed: 2018-10-09.

[6] Hallgeir Horne. *Non-Intrusive Load Monitoring with Norwegian Smart Meters*. PhD thesis, Norwegian University of Science and Technology, 2018.

[7] Eco dataset. https://www.vs.inf.ethz.ch/res/show.html?what=eco-data. Accessed: 2018-12-10.

[8] J Zico Kolter and Matthew J Johnson. Redd: A public data set for energy disaggregation research. *Artif. Intell.*, 25, 01 2011.

[9] ETH Zurich. Nilm-eval: An evaluation framework for non-intrusive load monitoring algorithms. https://github.com/beckel/nilm-eval, 2015. Accessed: 2018-11-27.

[10] Distributed systems group. http://vs.inf.ethz.ch, 2014. Accessed: 2018-11-27.

[11] George Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80: 1870 – 1891, 01 1993. doi: 10.1109/5.192069.

[12] B. Naghibi and S. Deilami. Non-intrusive load monitoring and supplementary techniques for home energy management. In *2014 Australasian Universities Power Engineering Conference (AUPEC)*, pages 1–5, Sept 2014. doi: 10.1109/AUPEC.2014.6966647.

[13] YUNG FEI WONG. A non-intrusive load monitoring framework for robust real-time disaggregation of smart meter data, Jan 2018. URL `https://monash.figshare.com/articles/A_Non-Intrusive_Load_Monitoring_Framework_for_Robust_Real-Time_Disaggregation_of_Smart_Meter_Data/5798574/1`.

[14] Markus Weiss, Thorsten Staake, and Friedemann Mattern. Leveraging smart meter data to recognize home appliances. *2012 IEEE International Conference on Pervasive Computing and Communications, PerCom 2012*, 03 2012. doi: 10.1109/PerCom.2012.6199866.

[15] George Hart and Anastasios T. Bouloutas. Correcting dependent errors in sequences generated by finite-state processes. *IEEE Transactions on Information Theory*, 39: 1249–1260, 07 1993. doi: 10.1109/18.243442.

[16] Wesley A. Souza, Fernando P. Marafão, Eduardo V. Liberado, Marcelo G. Simões, and Luiz C. P. Da Silva. A nilm dataset for cognitive meters based on conservative power theory and pattern recognition techniques. *Journal of Control, Automation and Electrical Systems*, 29(6):742–755, Dec 2018. ISSN 2195-3899. doi: 10.1007/s40313-018-0417-4. URL `https://doi.org/10.1007/s40313-018-0417-4`.

[17] Ahmed Zoha, Alexander Gluhak, Muhammad Imran, and Sutharshan Rajasegarar. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. *Sensors (Basel, Switzerland)*, 12:16838–16866, 12 2012. doi: 10.3390/s121216838.

[18] Nilmtk. `https://github.com/nilmtk/nilmtk`. Accessed: 2018-11-14.

[19] Oliver Parson, Mark James Weal, and Alex Rogers. A scalable non-intrusive load monitoring system for fridge-freezer energy efficiency estimation. 2014.

[20] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. *IJPRAI*, 15:9–42, 02 2001. doi: 10.1142/S0218001401000836.

[21] M Baranski and J Voss. Genetic algorithm for pattern detection in nialm systems. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 4:3462 – 3468 vol.4, 11 2004. doi: 10.1109/ICSMC.2004.1400878.

[22] M Baranski and J Voss. Genetic algorithm for pattern detection in nialm systems. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 4:3462 Introduction, 11 2004. doi: 10.1109/ICSMC.2004.1400878.

[23] Lucas Pereira, Filipe Quintal, Rodolfo Gonçalves, and Nuno Nunes. Sustdata: A public dataset for ict4s electric energy research. *ICT for Sustainability 2014, ICT4S 2014*, 07 2014.

[24] Fibaro wall plug 2. https://www.fibaro.com/us/products/wall-plug/. Accessed: 2018-12-14.

[25] Tp-link wifi smart plug. https://www.tp-link.com/us/products/details/cat-5516_HS100.html. Accessed: 2018-12-14.

[26] D-link dsp-w115. https://eu.dlink.com/uk/en/products/dsp-w115-mydlink-wifi-smart-plug. Accessed: 2018-12-14.

[27] Eve energy. https://www.evehome.com/en/eve-energy. Accessed: 2018-12-14.

[28] Belkin wemo insight. https://www.belkin.com/us/p/P-F7C029/. Accessed: 2018-12-14.

[29] Jack (Daniel) Kelly. *Disaggregation of Domestic Smart Meter Energy Data*. PhD thesis, University of London and the Diploma of Imperial College, 2017.

[30] Plugwise. Plugwise circle. https://www.plugwise.com/documents/plugwise/product_documents/EN-plugwise-circlef-ts.pdf, 2018. Accessed: 2018-12-08.

[31] Plugwise datasheet. https://www.plugwise.com/documents/plugwise/product_documents/EN-plugwise-circlef-ts.pdf. Accessed: 2018-12-14.

[32] Raspberry pi power requirements. https://www.raspberrypi.org/documentation/faqs/#pi-power, 2018. Accessed: 2018-11-13.

[33] SevenWatt. Plugwise-2-py. https://github.com/SevenW/Plugwise-2-py. Accessed: 2018-11-13.

[34] Vnc. https://www.realvnc.com/en/, 2018. Accessed: 2018-11-13.

[35] Rasbian jessie. http://downloads.raspberrypi.org/raspbian/images/raspbian-2017-07-05/, 2017. Accessed: 2018-11-13.

[36] Lucas Pereira and andNuno Nunes Miguel Ribeiro. Engineering and deploying a hardware and software platform to collect and label non-intrusive load monitoring datasets. 2017.

[37] Node-red. https://nodered.org. Accessed: 2018-11-14.

[38] Etcher. https://www.balena.io/etcher/, 2018. Accessed: 2018-11-13.

[39] Mqtt. http://mqtt.org. Accessed: 2018-11-14.

[40] Domoticz. https://nodered.orghttps://www.domoticz.com/wiki/Main_Page. Accessed: 2018-11-14.

[41] Hjemmeautomasjon. www.hjemmeautomasjon.no. Accessed: 2018-12-18.

Appendix

Appendix A

Data collecting software installation guide

No previous research described the experiment design in detail. To aid in research dissemination, this step-by-step guide covering how to install the data collecting system used in this thesis was written. All files and code made by the author can be found on the github repo `https://github.com/powermundsen/NILM`. There are often many ways of installing and running software. The steps described in this appendix are the ones that worked for the author, and hopefully, they will work for the reader as well. Good Luck!

1. Set up the Plugwise system on a Windows machine to make sure the plugs and the Plugwise Stick is working correctly.

2. Make a spreadsheet of the MAC-addresses and corresponding given plug numbers, names, locations, type and a short version of the MAC, as shown in table A.1.

MAC	Plug number	Name	Location	Type	Short MAC
000D6F0003BD6969	01	Electric heater Andre	Bedroom	Circle	BD6969
000D6F0003BD8974	02	Water Heater	Shed/Storeroom	Circle	BD8974
000D6F0002C0DDDB	03	Oven	Kitchen	Circle	C0DDDB
000D6F000416DD83	04	TV	Livingroom	Circle	16DD83
000D6F0003BD5FE2	05	Coffee Maker	Shed/Storeroom	Circle	BD5FE2
000D6F0003561FC7	06	Electric kettle	Kitchen	Circle	561FC7
000D6F0004B1EC41	07	Electric heater salong	Livingroom	Circle	B1EC41
000D6F0002C0E746	08	Microwave oven	Kitchen	Circle	C0E746
000D6F0000469B3E	09	Electric heater terrace	Livingroom	Circle+	469B3E
000D6F0004B613A8	10	Dishwasher	Kitchen	Circle	B613A8
000D6F0004A29FA3	11	Refrigerator	Kitchen	Circle	A29FA3
000D6F0002C0EFF3	12	Washing machine	Shed/Storeroom	Circle	C0EFF3

Table A.1: Smart Plug details in experiment.

3. Use a Raspberry Pi with a correct power supply to make sure it runs as it should. If a power supply with lower power is used, the CPU will be throttled.

4. Use a high class SD card that will not throttle the Raspberry Pi's performance.
5. Download an operating system for the Raspberry Pi. In my case i used the Raspian Stretch Desktop (`https://www.raspberrypi.org/downloads/raspbian/`)
6. Flash the SD-card with the OS just downloaded using Etcher (`https://www.balena.io/etcher/`)
7. Plug the SD-card into the Raspberry Pi and connect a keyboard, mouse, Ethernet internet or WiFi dongle and a display for initial setup.
8. In the desktop menu, open "Raspberry Pi Configuration". Under "Interfaces", enable SSH and VNC for best remote access.You should also change the default password to a more secure one.
9. Download and install RealVNC (`https://www.realvnc.com/en/connect/download/vnc/`). Open RealVNC and configure a remote connection to the Raspberry Pi. Since I have not configured a static IP-address I made a free account and made the Raspberry Pi accessible through their cloud service.
10. To connect to the Raspberry Pi within your local network, go to terminal and type one of the two commands:

    ```
    $ ssh pi@<raspberry ip>
    $ ssh pi@raspberrypi.local
    ```

    Then log in with your password.
11. To enable file sharing on the Raspberry Pi, follow this guide (`http://raspberrypituts.com/access-raspberry-pi-files-in-your-os-x-finder/`)

    ```
    $ sudo apt-get install netatalk
    $ ifconfig
    $ open afp://192.168.0.10
    ```
12. Plugwise-2-py (`http://github.com/SevenW/Plugwise-2-py`)
13. Installing the software. To make it easier for yourself, install it in the /home/pi folder.

    ```
    $ sudo python get-pip.py
    $ git clone https://github.com/SevenW/Plugwise-2-py.git
    $ cd Plugwise-2-py
    $ sudo pip install .
    ```
14. Moving the config-files. From Plugwise-2-py folder, copy the config-files to the main folder:

    ```
    $ cp -n config-default/pw-hostconfig.json config/
    $ cp -n config-default/pw-control.json config/
    $ cp -n config-default/pw-conf.json config/
    ```

15. Configuring pw-hostconfig.json:

    - Open the config/pw-hostconfig.json and check if it has the correct paths.
    - If Plugwise-2-py was installed in the /home/pi the file should be contain the right path:

```
{"permanent_path":"/home/pi/datalog","tmp_path":"/tmp","log_path":"/home/pi/pwlog",
"serial":"/dev/ttyUSB0","log_format":"epoch","mqtt_ip":"127.0.0.1","mqtt_port":"1883"
```

    Note that:

    - "Serial" might not point to the Plugwise stick in all cases. Check therefore if this is correct for you.
    - To enable MQTT messaging later the "log_format", "mqtt_ip" and "mqtt_port" is added.
    - Editing JSON files is error-prone. Use a JSON Validator such as http://jsonlint.com/ to check the config files.

16. Configuring pw-conf.json:

    - Here you add the plugs values created in step 1.
    - Be sure to use the JSON Validator!
    - Example data:

```
{"static": [
{"mac":"000D6F0003BD5FE2","category":"Coffee-maker","name":"Coffee-maker",
"loginterval":"60","always_on":"False","production":"False","location":"Shed/Storeroom"},
{"mac":"000D6F0004B613A8","category":"Dishwasher","name":"Dishwasher",
"loginterval":"60","always_on":"False","production":"False","location":"Kitchen"},
{"mac":"000D6F0003BD6969","category":"Electric heater","name":"Electric heater Andre",
"loginterval":"60","always_on":"False","production":"False","location":"Bedroom Andre"},
{"mac":"000D6F0004B1EC41","category":"Electric heater","name":"Electric heater salong",
"loginterval":"60","always_on":"False","production":"False","location":"Living room"},
{"mac":"000D6F0000469B3E","category":"Electric heater","name":"Electric heater terrace",
"loginterval":"60","always_on":"False","production":"False","location":"Living room"},
{"mac":"000D6F0003561FC7","category":"Electric kettle","name":"Electric kettle",
"loginterval":"60","always_on":"False","production":"False","location":"Kitchen"},
{"mac":"000D6F0002C0E746","category":"Microwave oven","name":"Microwave oven",
"loginterval":"60","always_on":"False","production":"False","location":"Kitchen"},
{"mac":"000D6F0002C0DDDB","category":"Oven", "name":"Oven",
"loginterval":"60","always_on":"False","production":"False","location":"Kitchen"},
{"mac":"000D6F0004A29FA3","category":"Refrigerator","name":"Refrigerator",
"loginterval":"60","always_on":"False","production":"False","location":"Kitchen"},
{"mac":"000D6F000416DD83","category":"TV","name":"TV",
"loginterval":"60","always_on":"False","production":"False","location":"Living room"},
{"mac":"000D6F0002C0EFF3","category":"Washing machine","name":"Washing machine",
"loginterval":"60","always_on":"False","production":"False","location":"Shed/Storeroom"},
{"mac":"000D6F0003BD8974","category":"Water heater vessel","name":"Water heater vessel",
"loginterval":"60","always_on":"False","production":"False","location":"Shed/Storeroom"}
]}
```

17. Configuring pw-control.json: - Here you also plug in values from step 1. - Be sure to use the JSON Validator! - Example data:

```
{"dynamic": [
{"mac": "000D6F0003BD5FE2", "switch_state": "on", "name":"Coffee-maker",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0004B613A8", "switch_state": "on", "name":"Dishwasher",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0003BD6969", "switch_state": "on", "name":"Electric heater Andre",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0004B1EC41", "switch_state": "on", "name":"Electric heater salong",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0000469B3E", "switch_state": "on", "name":"Electric heater terrace",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0003561FC7", "switch_state": "on", "name":"Electric kettle",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0002C0E746", "switch_state": "on", "name":"Microwave oven",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0002C0DDDB", "switch_state": "on", "name":"Oven",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0004A29FA3", "switch_state": "on", "name":"Refrigerator",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F000416DD83", "switch_state": "on", "name":"TV",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0002C0EFF3", "switch_state": "on", "name":"Washing machine",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"},
{"mac": "000D6F0003BD8974", "switch_state": "on", "name":"Water heater vessel",
"schedule_state": "off", "schedule": "", "savelog": "yes","monitor": "yes"}
], "log_level": "info", "log_comm": "no"}
```


18. Running plugwise-2-py: - To run, type the following in terminal (Plugwise-2-py main directory):

    ```
    $ sudo python Plugwise-2.py
    ```

    - The first time it runs it collects the buffered data from the plugs. This might take minutes, or hours. You can watch the progress by tailing the log:

    ```
    $ tail -f /home/pi/pwlog/pw-logger.log
    ```

    or look at the files updating in the Plugwise-2-py folder.

19. To run the web interface, type in terminal (Plugwise-2-py main directory):

    ```
    $ sudo python Plugwise-2-web.py
    ```

    - You can access the web interface at `http://localhost:8000/pw2py.html`. I got the web interface running, but at this point it didn't update the values. The values showed up after implementing the MQTT service. - Beacuase of the changes in sockets, the smart plug values might not show up in Safari web browser until the protocol version in modified in swutil/HTTPWebSocketsHandler.py: After the line with

    ```
    "_opcode_pong = 0xa"
    ```

    add

    ```
    "protocol_version = 'HTTP/1.1'"
    ```

    and restart Plugwise-2-web.py

20. Check the log-files to see if you receive data - If you receive data and the system is initialized correctly the following log files should be updated:

    - /home/pi/datalog/2018/pwact/pwact-2018-10-25-000D6F0003BD6969.log (One file per plug per day)
    - /home/pi/datalog/2018/pwlog/pw-000D6F0002C0E746.log (One file per device)
    - /home/pi/datalog/pwlastlog.log (Contains the last values for each device)
    - /home/pi/pwlog/pw-logger.log (Logs the logging, should say save log for each device)
    - /home/pi/pwlog/pw-web.log (Logs the web-interface, should be filled with MQTT-messages after step XX)

21. Installing mosquitto

    - The terminal line below did not work for me and I installed using this guide: `http://mosquitto.org/blog/2013/01/mosquitto-debian-repository/`

      ```
      $ sudo apt-get install mosquitto
      ```

22. Installing Domoticz is not necessary to collect the smart meter data. The Domotics installation was a source to great frustration because the message handling in Plugwise-2-py. At the time of writing the thesis, none of the Domoticz versions 4.X worked and it is therefore necessary to Install a 3.X version of the software. The easiest way to find a previous version is to compile and make it yourself from their github repo `https://github.com/domoticz/domoticz`. You might also be able to find an installation file online at `https://egregius.be/2018/previous-domoticz-versions/`. When installed and running, Domoticz can be reached at: http://127.0.0.1:8080

23. Node-RED To send the values to Domoticz, you have to use Node-RED to relay the messages from Plugwise-2-py.

    (a) Install node.js

    `https://www.instructables.com/id/Install-Nodejs-and-Npm-on-Raspberry-Pi/`

    (b) Install Node-RED (urlhttps://nodered.org/docs/getting-started/installation)

    (c) Run Node-RED in terminal:

    ```
    $ node-red
    ```

    (d) Access at `http://<Raspberrypiip>:1880`

    (e) Configure the flow by going to the top right corner and choose "Import". The standard flow is found in /home/pi/Plugwise-2-py/domoticz/plugwise2py-domoticz.nodered. The modified version also saves to file is found here: `https://github.com/powermundsen/thesis`.

(f) Deploy! You should now see the messages being sent in the terminal. To double check you can download a more graphical version like `https://mqttfx.jensd.de`.

(g) To make Node-RED start at boot, do the following (`https://nodered.org/docs/hardware/raspberrypi`)

NOTE: Node-RED warns about errors when you try to deploy, these disappeared when I clicked on the nodes and changed the server address. However, this is not a problem because when set up correctly it will work either way.

NOTE 2: It became clear at a later point that the best way to log the consumption is not in Domoticz, but directly in Node-RED. I expanded the flow above to do this and it is saved as a new flow in the "Master" folder (scripts for .....) .

24. Connecting Domoticz and Node-RED

(a) Follow the guide in /home/pi/Plugwise-2-py/domoticz/README.md

(b) I have not yet gotten this connection to work but I can see that the messages are being sent out on the network.

- To change the sampling time, change the following line in /home/pi/Plugwise-2-py/Plugwise-2.py to the time you want:

```
proceed_at = ref + timedelta(seconds=(2 - ref.second%2), microseconds=-ref.microsecond)
```

NOTE: Even though the polling frequency was changed to 1 second, the values received was usually between 2-3 seconds apart. I think this is because Plugwise-2-py filters out duplicate messages.

- Make a backup of the Raspberry Pi disk:

1. Check disk utility to see which disk the Raspberry Pi SD card has
2. In terminal:

```
$ mkdir raspberry_backup
$ cd raspberry_backup/
$ sudo dd if=/dev/disk3 of=~/Desktop/raspberrypi.dmg
```

- Congratulations, the system should now be functional! When starting up, run the following commands (in separate terminal windows):

```
$ sudo python /home/pi/Plugwise-2-py/Plugwise-2.py
$ sudo python /home/pi/Plugwise-2-py/Plugwise-2-web.py
$ node-red
$ ./domoticz_3.8153_linux_armv7l/domoticz
```

Note: In this experiment, the Raspberry Pi rebooted itself periodically. This was circumvented by executing the commands on the Pi itself through VNC.

Appendix B

Data formatting

Ecomonitor data converter

This program has been written to convert the available consumption data from the NURI smart meter to the preferred structure for NILM-Eval (https://github.com/beckel/nilm-eval (https://github.com/beckel/nilm-eval)). The data is received from the EcoMonitor smart meter data collector made by Hark Tech by downloading a CSV-file from the customer website.

Input: 1 CSV-file from EcoMonitor from the path data/rawdata/smartmeter/ecomonitor.csv.

Output: 7 separated consumption CSV-files in folders based on the date of data collection in the path: /data/powermundsen_data/smartmeter/YYYY-MM-DD

Removing unwanted colums

In [ ]:
```
import csv
import pandas as pd
from pylab import *
import datetime
import time
import os


# Setting some variables
mpl.rcParams['agg.path.chunksize'] = 10000
plt.rcParams["figure.dpi"] = 600
plt.rcParams.update({'font.size': 22})
current_directory = os.getcwd()
household = '01'
plotvalues = {'Pi':'Active power in [W]',
              'Pe':'Active power out [W]',
              'Qi':'Reactive power in [VAr]',
              'Qe':'Reactive power out [VAr]',
              'I1':'Current phase 1 [A]',
              'I2':'Current phase 2 [A]',
              'I3':'Current phase 3 [A]',
              'U1':'Voltage phase 1 [V]',
              'U2':'Voltage phase 2 [V]',
              'U3':'Voltage phase 3 [V]',
              'Ai':'Accumulated active power in [KW]',
              'Ae':'Accumulated active power out [KW]',
              'Ri':'Accumulated reactive power in [VArh]',
              'Re':'Accumulated reactive power out [VArh]'}
units = {'Pi':'W',
 'Pe':'kW',
 'Qi':'VAr',
 'Qe':'VAr',
 'I1':'Amper',
 'I2':'Amper',
 'I3':'Amper',
 'U1':'Volt',
```

```
 'U2':'Volt',
 'U3':'Volt',
 'Ai':'KW',
 'Ae':'KW',
 'Ri':'VArh',
 'Re':'VArh'}
filenames = {'Pi':'powerallphases',
 'I1':'currentl1',
 'I2':'currentl2',
 'I3':'currentl3',
 'U1':'voltagel1',
 'U2':'voltagel2',
 'U3':'voltagel3'}


# Import the CSV
df = pd.read_csv('data/rawdata/smartmeter/ecomonitor.csv')

# Renaming columns
df.rename(columns={'DTM':'time'}, inplace=True)

# Converting from kW to W
df.loc[:,'Pi'] *= 1000
df.loc[:,'Pe'] *= 1000
df.loc[:,'Qi'] *= 1000
df.loc[:,'Qe'] *= 1000
df.loc[:,'Ri'] *= 1000
df.loc[:,'Re'] *= 1000

# Making new columns for date and time
df['year'] = df['time'].str.slice(0,4)
df['month'] = df['time'].str.slice(5,7)
df['day'] = df['time'].str.slice(8,10)
df['time_of_day'] = df['time'].str.slice(11,16)

# Adding seconds count
for index, row in df.iterrows():
    timestamp = row['time']
    time_reduced = timestamp[-8:]
    x = time.strptime(time_reduced.split(',')[0],'%H:%M:%S')
    second = datetime.timedelta(hours=x.tm_hour,minutes=x.tm_min,seconds=x.
tm_sec).total_seconds()
    df.set_value(index,'second',second)

#print(df)

# Finding unique days
unique_years = df.year.unique()
unique_months = df.month.unique()
unique_days = df.day.unique()
print('Unique years: %s, Unique months: %s, Unique days: %s' %(unique_years
, unique_months, unique_days))

print('\t\t Staring day iteration')
for year in unique_years:
    for month in unique_months:
        for day in unique_days:
            print('\t\t\t Working on day %s-%s-%s' %(year, month, day))
```

```
            df_temp = df.loc[(df['day'] == day)] # Locks the rows with this
value
            #print(df_temp)


            # Setting second to be index
            df_temp.set_index("second")
            new_index = pd.Index(arange(1,86401), name="second")
            df_temp = df_temp.set_index("second").reindex(new_index)
            #print(df_temp)



            # Plot graphs and save to
            print('\t\t\t Making daily plots')
            for value in plotvalues:
                df_print = df_temp.dropna(subset=[value])
                plot = df_print.plot(x = 'time_of_day', y = value, label= p
lotvalues[value], figsize=[20,10]);
                plot.set_xlabel('Smart meter measurements %s-%s-%s' %(year,
month, day))
                plot.set_ylabel(units[value])
                fig = plot.get_figure();


                plotpath = current_directory + '/data/plots/smartmeter/'
                if not os.path.exists(plotpath):
                    os.makedirs(plotpath)
                filename = year+'-'+month+'-'+day+':'+ value +'-'+plotvalue
s[value]
                fig.savefig(plotpath + filename +'.png')
                plt.clf()
                plt.close('all')

                # Adding value -1 in power to rows with NaN to please matla
b program
                df_temp[value].fillna('-1', inplace=True)
                print('Check value change on NaN in value %s' %value)
                #print(df_temp)

                # Save to file with a filename that includes: date, value,
household.
                print('\t\t\t Saving to file')

                path = current_directory + '/data/powermundsen_data/smartme
ter/' + household + '/' + year + '-' + month + '-' + day
                if not os.path.exists(path):
                    os.makedirs(path)

                if value in filenames:
                    df_temp.to_csv(os.path.join(path, r'%s' %(filenames[val
ue]) + '.csv'), columns=[value], index=True, header=False)



            # Adding the remaining files the add on uses (These have no val
ue since we dont have the data)
```

```
            print('Making the empty files...')
            path = current_directory + '/data/powermundsen_data/smartmeter/
' + household + '/' + year + '-' + month + '-' + day
            others = ['currentneutral','power11', 'power12', 'power13', 'ph
aseanglevoltage121', 'phaseanglevoltage131', 'phaseanglecurrentvoltage11'
, 'phaseanglecurrentvoltage12', 'phaseanglecurrentvoltage13']
            for element in others:
                df_temp[element] = -1
                df_temp.to_csv(os.path.join(path, r'%s' %element +'.csv'),
columns=[element], index=True, header=False)

            print('\t\t\t Finished processing day %s ' %day)


print('Finished processing all smart meter data')
```

Plugwise data converter

This script has been written to convert the logged consumption data from the Plugwise Circle to the preferred structure for NILM-Eval (https://github.com/beckel/nilm-eval (https://github.com/beckel/nilm-eval)). The data has been received from the Plugwise-2-py MQTT messages in Node-Red and saved in CSV files.

Input: CSV-files from the Plugwise-2-py setup on the raspberry pi. By default, the files are saved to /home/pi/logging/plugs/ with the filename <numberOfPlug.csv>. Example: /home/pi/logging/plugs/01.csv

```
The file contains the columns: MAC-address, Timestamp, Power usage.
```

Output: 1 CSV-file per day with the plug's consumption 8with the columns: Second of the day, Consumption).

Removing unwanted colums

In [ ]:
```
import csv
import pandas as pd
import datetime
import time
from pylab import *
import os
import matplotlib.pyplot as plt


# Adding some variables
# TODO: HOUSEHOLD NEEDS TO BE GENERELIZED
household = '01'
appliances = {
'01':'Electric heater Andre',
'02':'Water Heater',
'03':'Oven',
'04':'TV',
'05':'Coffee Maker',
'06':'Electric kettle',
'07':'Electric heater salong',
'08':'Microwave oven',
'09':'Electric heater terrace',
'10':'Dishwasher',
'11':'Refrigerator',
'12':'Washing machine'}
current_directory = os.getcwd()
mpl.rcParams['agg.path.chunksize'] = 10000
plt.rcParams["figure.dpi"] = 600
plt.rcParams.update({'font.size': 22})


for plug in appliances:
    print('*Starting work on %s - %s' %(plug, appliances[plug]))
    plugnumber = plug
```

```
    # Import the plug CSVs
    print('\t Importing CSV...')

    try:
        df = pd.read_csv('data/rawdata/plugs/'+ plug + '.csv', header=None)
        print('\t Import ok')
    except :
        print('Did not find file %s.csv, this will be excluded' %plug)
        continue

    # Renaming columns
    df.columns = ['mac', 'time', 'power']

    # Saving the plug's mac
    mac = df.at[1,'mac']

    # Removing possible duplicates in the data
    df = df[~df.time.duplicated()]

    # Removing negative-valued noise in data
    df.power[df.power < 0] = 0

    # Making new columns for date and time
    df['year'] = df['time'].str.slice(0,4)
    df['month'] = df['time'].str.slice(5,7)
    df['day'] = df['time'].str.slice(8,10)
    df['time_of_day'] = df['time'].str.slice(11,16)

    # Adding seconds count
    for index, row in df.iterrows():
        timestamp = row['time']
        time_reduced = timestamp[-8:]
        x = time.strptime(time_reduced.split(',')[0],'%H:%M:%S')
        second = datetime.timedelta(hours=x.tm_hour,minutes=x.tm_min,second
s=x.tm_sec).total_seconds()
        df.set_value(index,'second',second)

    # Finding unique days
    unique_years = df.year.unique()
    unique_months = df.month.unique()
    unique_days = df.day.unique()

    print('\t\t Unique years: %s, months: %s, days: %s' %(unique_years, uni
que_months, unique_days))


    print('\t\t Staring day iteration for: %s' %appliances[plugnumber])
    for year in unique_years:
        for month in unique_months:
            for day in unique_days:
                print('\t\t\t Working on day %s' %day)

                df_temp = df.loc[(df['day'] == day)] # Locks the rows with
this value
                #print(df_temp)
```

```
                # Setting second to be index
                df_temp.set_index("second")
                new_index = pd.Index(arange(1,86401), name="second")
                df_temp = df_temp.set_index("second").reindex(new_index)

                # Plot graphs and save to
                print('\t\t\t Making daily plot')
                plot = df_temp.dropna().plot(x = 'time_of_day', y = 'power'
, label='Power [W]', figsize=[20,10]);
                plot.set_xlabel('%s on %s-%s-%s' %(appliances[plugnumber],
year, month, day))
                plot.set_ylabel('Watt')
                fig = plot.get_figure();


                plotpath = current_directory + '/data/plots/plugs/'
                if not os.path.exists(plotpath):
                    os.makedirs(plotpath)
                filename = year+'-'+month+'-'+day+':'+plugnumber+'-'+applia
nces[plugnumber]
                fig.savefig(plotpath + filename +'.png')
                plt.clf()
                plt.close('all')

                # Adding value -1 in power to rows with NaN to please matla
b program
                df_temp.power = df_temp.power.fillna(-1)

                # Save to file with filename that inclueds: Plug number, da
te, household.
                print('\t\t\t Saving to file')
                path = current_directory + '/data/powermundsen_data/plugs/'
+ household + '/' + plugnumber
                if not os.path.exists(path):
                    os.makedirs(path)

                cols_with_data = ['power']
                for col in cols_with_data:
                    df_temp.to_csv(os.path.join(path, r'%s-%s-%s' %(year, m
onth, day) + '.csv'), columns=[col], index=True, header=False)
                print('\t\t\t Finished processing day %s ' %day)


print('Finished processing all plugwise data')
```

B.1 Matlab Script for Data Formatting

```
% This script will create .mat files of the data so that the NILM-eval kit
% can process it. It should be placed and run from the folder above /data/powermundsen_data/

current_directory = pwd;
% Setting paths to data
path_smart_plugs = fullfile(current_directory, '/data/powermundsen_data/plugs/');
path_smart_meter = fullfile(current_directory, '/data/powermundsen_data/smartmeter/');
sp_households = string.empty();
sm_households = string.empty();
plugs = string.empty();
meters = string.empty();


% Finding household folders for plugs
files = dir(path_smart_plugs);
dirFlags = [files.isdir] & ~strcmp({files.name},'.') & ~strcmp({files.name},'..');
subFolders = files(dirFlags);
disp('Households for smart plugs found: ');
for k = 1 : length(subFolders)
  sp_households = [sp_households, subFolders(k).name];
  fprintf('Household #%d = %s\n', k, subFolders(k).name);
end

% Finding plugs
files = dir(strcat(path_smart_plugs, sp_households{1}));
dirFlags = [files.isdir] & ~strcmp({files.name},'.') & ~strcmp({files.name},'..');
subFolders = files(dirFlags);
disp('Smart plugs found: ');
for k = 1 : length(subFolders)
  plugs = [plugs, subFolders(k).name];
  fprintf('Plug #%d = %s\n', k, subFolders(k).name);
end

% Iterate through smart plug folders to make mat-files
disp('Start: Iterate through smart plug folders to make mat-files')
for h = 1 : length(sp_households)
    for p = 1 : length(plugs)

        % Setting variables
        household = char(sp_households(h));
        plug = char(plugs(p));
        csv_files = string.empty();
        fprintf("Working on household %s plug %s \n", household, plug)

        %path = strcat(path_smart_plugs, sp_households(h), '/', plugs(p));
        folder = fullfile(path_smart_plugs, household, plug);

        % Finding dates
        files = dir(folder);
        dirFlags = ~strcmp({files.name},'.') & ~strcmp({files.name},'..');
        subFolders = files(dirFlags);
        for k = 1 : length(subFolders)
            if endsWith(subFolders(k).name, '.csv')
                csv_files = [csv_files, subFolders(k).name];
            end
            %fprintf('Date %s \n', k, subFolders(k).name);
        end
        fprintf('Date found %s \n', csv_files)

        % Importing CSV
        for k = 1: length(csv_files)
            fprintf('%s : Reading CSV file for plug: %s\n', csv_files(k), plug);
            filepath = fullfile(folder, char(csv_files(k)));
            [file_path_function, name, ext] = fileparts(filepath);
            consumption = csvread(filepath,0,1);
```

```
            % Making struct
            % Variables
            date = regexprep(csv_files(k),'[-]','');
            date = regexprep(date, '[.csv]', '');
            struct_filename = char(strcat('Appliance', household, plug, date));
            consum = consumption;

            % Creating main struct
            s = struct();
            % Creating struct for text variables
            j = struct();

            % Adding data to main struct
            s.household = household;
            s.plug = plug;
            s.consumption = consumption;

            % Solving the naming problem
            j = struct(struct_filename, s);

            % Exporting to .mat file
            %name = '2018-11-09';
            name = regexprep(csv_files(k), '[.csv]', '');
            filename = strcat(name, '.mat');
            savepath = strcat(file_path_function, '/', filename);
            save(savepath, '-struct', 'j');
            fprintf('Saved file \n');

            % Showing the contents
            %whos(filepath, '2018-11-09.mat')

        end
    end
end


% Finding household folders for smart meters
folders = dir(path_smart_meter);
dirFlags = [folders.isdir] & ~strcmp({folders.name},'.') & ~strcmp({folders.name},'..');
subFolders = folders(dirFlags);
disp('Households for smart meter found: ');
for k = 1 : length(subFolders)
  sm_households = [sm_households, subFolders(k).name];
  fprintf('Household #%d = %s\n', k, subFolders(k).name);
end

% Finding dates
disp('Iterating through households');
for h = 1 : length(sm_households)
    fprintf('Household %s : Started \n', sm_households(h));
    household = subFolders(k).name;
    folder = fullfile(path_smart_meter, household);
    sm_folders = dir(folder);
    dirFlags = [sm_folders.isdir] & ~strcmp({sm_folders.name},'.') & ~strcmp({sm_folders.name},'..');
    subFolders = sm_folders(dirFlags);
    fprintf('Household %s : Dates found \n', household);
    fprintf('%s, ', subFolders.name);

    % For each folder in subFolders
    for day = 1 : length(subFolders)
        fprintf('\n Household %s : Reading files for day %s \n', household, subFolders(day).name);

        % Variables
        date = subFolders(day).name;
        path_files = fullfile(folder, subFolders(day).name);
        filepath = fullfile(folder,char(subFolders(day).name));
        [file_path_function, name, ext] = fileparts(filepath);
```

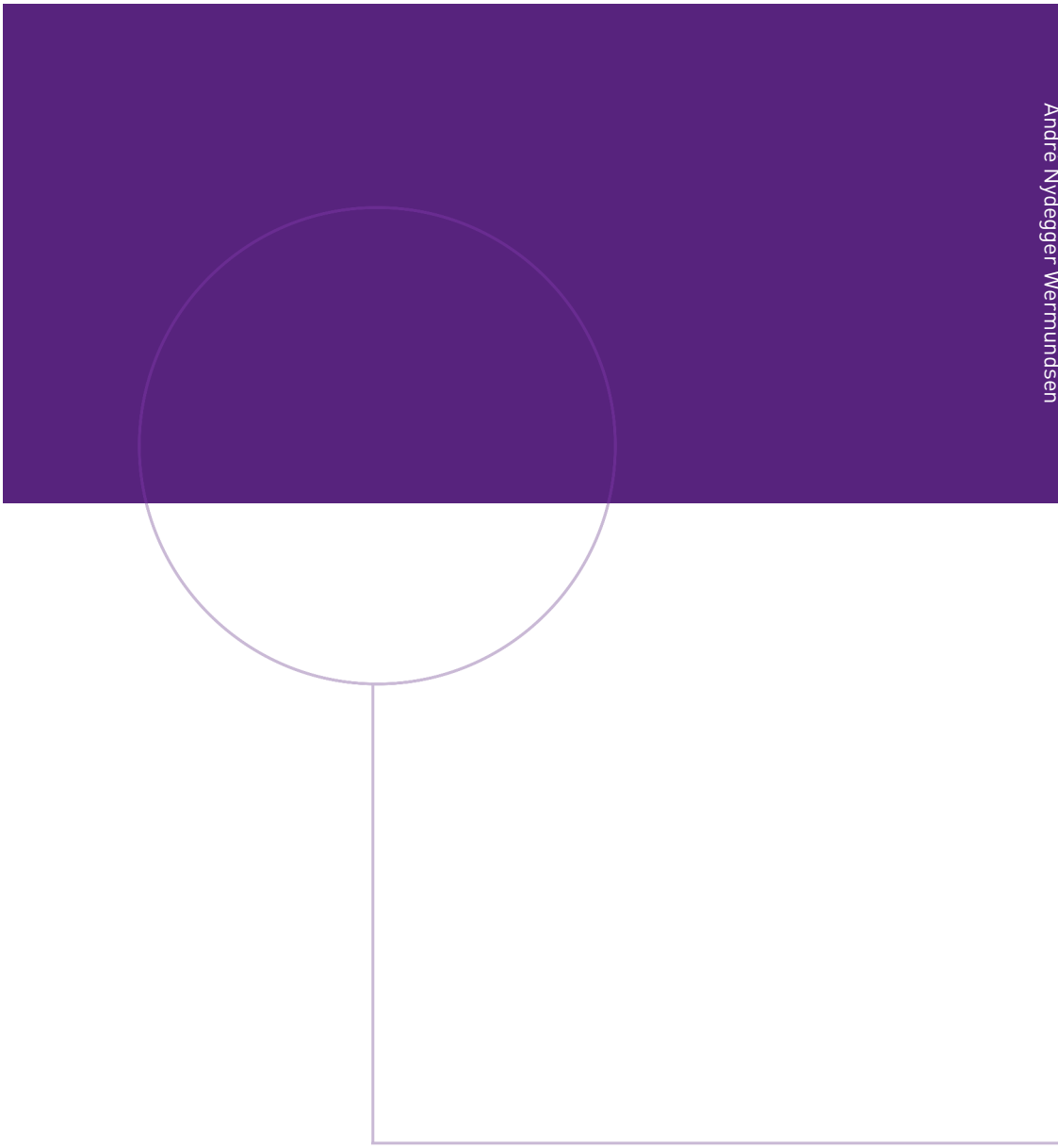
```
        % Creating main struct
        s = struct();

        % Saving files to main struct
        s.powerallphases = csvread(fullfile(path_files,'/powerallphases.csv'),0,1);
        s.powerl1 = csvread(fullfile(path_files,'/powerl1.csv'),0,1);
        s.powerl2 = csvread(fullfile(path_files,'/powerl2.csv'),0,1);
        s.powerl3 = csvread(fullfile(path_files,'/powerl3.csv'),0,1);
        s.currentneutral = csvread(fullfile(path_files,'/currentneutral.csv'),0,1);
        s.currentl1 = csvread(fullfile(path_files,'/currentl1.csv'),0,1);
        s.currentl2 = csvread(fullfile(path_files,'/currentl2.csv'),0,1);
        s.currentl3 = csvread(fullfile(path_files,'/currentl3.csv'),0,1);
        s.voltagel1 = csvread(fullfile(path_files,'/voltagel1.csv'),0,1);
        s.voltagel2 = csvread(fullfile(path_files,'/voltagel2.csv'),0,1);
        s.voltagel3 = csvread(fullfile(path_files,'/voltagel3.csv'),0,1);
        s.phaseanglevoltagel2l1 = csvread(fullfile(path_files,'/phaseanglevoltagel2l1.csv'),0,1);
        s.phaseanglevoltagel3l1 = csvread(fullfile(path_files,'/phaseanglevoltagel3l1.csv'),0,1);
        s.phaseanglecurrentvoltagel1 = csvread(fullfile(path_files,'/phaseanglecurrentvoltagel1.csv'),0
        s.phaseanglecurrentvoltagel2 = csvread(fullfile(path_files,'/phaseanglecurrentvoltagel2.csv'),0
        s.phaseanglecurrentvoltagel3 = csvread(fullfile(path_files,'/phaseanglecurrentvoltagel3.csv'),0
        s.household = household;

        % Creating second struct
        j = struct();

        % Solving the naming problem by nesting
        struct_filename = char(strcat('Appliance', household, '00', regexprep(date, '-', '')));
        j = struct(struct_filename, s);

        % Exporting to .mat file
        filename = strcat(date, '.mat');
        savepath = strcat(file_path_function, '/', filename);
        save(savepath, '-struct', 'j');
        fprintf('Saved file \n');
    end
end
```



Andre Nydegger Wermundsen

NTNU
Norwegian University of
Science and Technology