UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Building appliances energy performance assessment

Zygimantas Jasiunas

Mestrado em informática

Trabalho de Projeto orientado por:
Prof. Doutor José Manuel daSilva Cecílio
Prof. Doutor Pedro MiguelFrazão Fernandes Ferreira

2021

Acknowledgments

I would like to express my most profound appreciation to my direct dissertation advisor José Manuel da Silva Cecílio, for his dedication and restless evenings to help me in writing this work. This year was quite challenging due to the global pandemic, and my advisor guided me to accomplish the goals that have been set for this project. The success of this dissertation would not have been possible without his support.

Furthermore, I would like to extend my sincere thanks to Pedro M. Ferreira for accepting me to be a part of the SATO project team. I will always be grateful for all the help I have received during this year.

Resumo

O consumo de energia tem vindo a crescer na União Europeia todos os anos, sendo de prever que, a curto prazo, se torne insustentável. No sentido de prevenir este cenário, a Comissão Europeia decidiu definir uma Estratégia Energética para a União Europeia, destacando dois objetivos: aumentar a eficiência energética e promover a descarbonização. Atualmente, cerca de 72% dos edifícios existentes na União Europeia não são energeticamente eficientes. Este problema motivou-nos à pesquisa e criação de soluções que permitam uma melhor avaliação do consumo energético por dispositivos elétricos em edifícios residenciais.

Neste contexto, o trabalho desenvolvido nesta tese consiste no desenho de uma solução de monitorização remota que recolhe informações de consumo energético recorrendo a técnicas de intrusive load monitoring, onde cada dispositivo elétrico individual é continuamente monitorizado quanto ao seu consumo energético. Esta abordagem permite compreender o consumo de energia, em tempo real e no dia-a-dia. Este conhecimento oferece-nos a capacidade de avaliar as diferenças existentes entre as medições laboratoriais (abordagem utilizada no sistema de rotulagem de equipamentos elétricos de acordo com a sua eficiência energética) e os consumos domésticos estimados. Para tal, nesta tese exploram-se abordagens de machine learning que pretendem descrever padrões de consumo, bem como reconhecer marcas, modelos e que funções os dispositivos elétricos estarão a executar. O principal objetivo deste trabalho é desenhar e implementar um protótipo de uma solução de IoT flexível e de baixo custo para avaliar equipamentos elétricos. Será utilizado um conjunto de sensores que recolherá dados relacionados com o consumo de energia e os entrega à plataforma SATO para serem posteriormente processados. O sistema será usado para monitorar aparelhos comumente encontrados em residências. Além disso, o sistema terá a capacidade de monitorizar o consumo de água de aparelhos que necessitem de abastecimento de água, como máquinas de lavar e de lavar louça. Os dados recolhidos serão usados para classificação dos aparelhos e modos de operação dos mesmos, em tempo real, permitindo fornecer relatórios sobre o consumo energético e modo de uso dos aparelhos, com grande grau de detalhe. Os relatórios podem incluir o uso de energia por vários ciclos de operação. Por exemplo, um aparelho pode executar vários ciclos de operação, como uma máquina de lavar que consume diferentes quantidades de energia elétrica e água consoante o modo de operação escolhido pelo utilizador. Toda a informação recolhida pode ser posteriormente utilizada em novos serviços de recomendação que ajudaram os utilizadores a definir melhor as configurações adequadas a um determinado dispositivo, minimizando o consumo energético e melhorando a sua eficiência. Adicionalmente toda esta informação pode ser utilizada para o diagnóstico de avarias e/ou manutenção preventiva. Em termos de proposta, o trabalho desenvolvido nesta tese tem as seguintes contribuições:

Sistema de monitorização remota: o sistema de monitorização desenhado e implementado nesta tese avança o estado da arte dos sistemas de monitorização propostos pela literatura devido ao facto de incluir uma lista aprimorada de sensores que podem fornecer mais informações sobre os aparelhos, como o consumo de água da máquina de lavar. Além disso, é altamente flexível e pode ser implementado sem esforço em dispositivos novos ou antigos para monitorização de consumo de recursos.

Conjunto de dados de consumo de energia de eletrodomésticos: Os dados recolhidos podem ser usados para futura investigação científica sobre o consumo de consumo de energia, padrões de uso de energia pelos eletrodomésticos e classificação dos mesmos.

Abordagem de computação na borda (Edge Computing): O sistema de monitorização proposto explora o paradigma de computação na borda, onde parte da computação de preparação de dados é executada na borda, libertando recursos da nuvem para cálculos essenciais e que necessitem de mais poder computacional.

Classificação precisa de dispositivos em tempo real: Com a proposta desenhada nesta tese, podemos classificar os dispositivos com alta precisão, usando os dados recolhidos pelo sistema de monitorização desenvolvido na tese. A abordagem proposta consegue classificar os dispositivos, que são monitorizados, com baixas taxas de falsos positivos.

Para fácil compreensão do trabalho desenvolvido nesta tese, de seguida descreve-se a organização do documento. O Capítulo 1 apresenta o problema do consumo de energia na União Europeia e discute o aumento do consumo da mesma. O capítulo apresenta também os principais objetivos e contribuições do trabalho.

No Capítulo 2 revê-se o trabalho relacionado em termos de sistema de monitorização remota, que inclui sensores, microcontroladores, processamento e filtragem de sinal. Por fim, este capítulo revê os trabalhos existentes na literatura relacionados com o problema de classificação de dispositivos usando abordagens de machine learning.

No Capítulo 3 discutem-se os requisitos do sistema e o projeto de arquitetura conceitual do sistema. Neste capítulo é proposta uma solução de hardware, bem como, o software e firmware necessários à sua operação. Os algoritmos de machine learning necessários à classificação são também discutidos, em termos de configurações necessárias e adequadas ao problema que queremos resolver nesta tese.

O Capítulo 4 representa a implementação de um protótipo que servirá de prova de conceito dos mecanismos discutidos no Capítulo 3. Neste capítulo discute-se também a forma de integração do protótipo na plataforma SATO. Com base na implementação feita, no Capítulo 5 especificam-se um conjunto de testes funcionais que permitem avaliar o desempenho da solução proposta e discutem-se os resultados obtidos a partir desses testes. Por fim, o Capítulo 6 apresenta as conclusões e o trabalho futuro que poderá ser desenvolvido partindo da solução atual.

Palavras-chave: avaliação de energia, processamento de sinal, coleta de dados de sinal,identificação de aparelhos.

Abstract

Energy consumption is daily growing in European Union (EU). One day it will become hardly sustainable. For this not to happen European Commission decided to implement a European Union Strategy, emphasizing two objectives: increasing energy efficiency and decarbonization. About 72% of all buildings in the EU are not adapted to be energy efficient. This problem encourages us to create solutions that would help assess the energy consumption of appliances at residential houses.

In this thesis, we proposed a system that collects data using an intrusive load monitoring approach, where each appliance will have a dedicated monitoring rig to collect the energy consumption data. The proposed solution will help us understand the real-life consumption of each device being monitored and compare the laboratory measurements observed versus domestic consumption estimated by the energy consumption based on the EU energy efficiency labelling system. The system proposed detects device consumption patterns and recognize its brand, model and what actions that appliance is executing, e.g., program of washing in a washing machine. To achieve our goal, we designed a hardware solution capable of collecting sensor data, filtering and send it to a cloud platform (the SATO platform). Additionally, in the cloud, we have a Machine Learning solution that deals with the data and recognizes the appliance and its operation modes. This recognition allows drawing a device/settings profile, which can detect faults and create a recommendation service that helps users define the better settings for a specific appliance, minimizing energy consumption and improving efficiency.

Finally, we examine our prototype approach of the system implemented for targeted objectives in this project report. The document describes the experiments that we did and the final results. Our results show that we can identify the appliance and some of its operation modes. The proposed approach must be improved to make the identification of all operation modes. However, the current version of the system shows exciting results. It can be used to support the design of a new labelling system where daily operation measures can be used to support the new classification system. This way, we have an approach that allows improving the energy consumption, making builds more efficient.

Keywords: energy assessment, signal processing, data collection, appliance recognition.

Contents

List of Figures

List of Tables

Acronyms

BLE Bluetooth low energy.

DSP Digital Signal Processing.

FFT Fast Fourier Transform.

I^{2}C Inter-Integrated Circuit.

IC Internal Circuitry.

IDE Integrated Development Environment.

ILM Intrusive Load Monitoring.

IoT Internet of Things.

IP Internet Protocol.

KNN K-nearest neighbour.

M2M Machine to machine.

MEMS Micro-Electro-Mechanical Systems.

ML Machine Learning.

MQTT MQ Telemetry Transport.

NILM Non-Intrusive load monitoring.

PC Personal Computer.

RF Random Forest.

SBC Single Board Computer.

SDA Serial Data.

SLC Serial Clock.

SPI serial peripherals interface.

SVM Support-vector machine.

TCP Transmission Control Protocol.

TWI Two Wire Connection.

UART Universal Asynchronous Receiver-Transmitter.

USB Universal Serial Bus.

WDT Watchdog timer.

WPA3 Wi-Fi Protected Access 3.

XGBoost Extreme Gradient Boosting.

Chapter 1

Introduction

Energy consumption is a growing topic in European Union (EU). Large Carbon dioxide (CO_2) emission is being shared across the EU (36%) [1]. These statistics are motivating European Union to create two main objectives: increasing energy efficiency and decarbonization. Compared to 1990 levels, EU has an ambitious objective of reaching a 32% share of renewable energy and increasing energy efficiency by at least 32.5% by 2030 [2].

About 72% of the EU buildings are not adapted for being energy efficient. Part of the energy efficiency problem in buildings is due to building construction issues. Since solutions such as demolishing and rebuilding are not economically neither environmentally viable at such a colossal scale, renovation and retrofit strategies are being proposed. However, the building construction problem is not only the one that influences energy efficiency.

This efficiency gap is also due to a deficit in knowledge about the overall real-life energy consumption and performance of buildings and their energy-consuming equipment. Existing equipment and building energy measurements paint a bleak image: real-life energy consumption often exceeds design predictions by more than 100%. As a result, reducing the gap between designed and measured energy use has become central in current efforts to increase energy efficiency. After successfully introducing building energy performance certificates, the market is now ready to assess real-life energy use that includes all energy-consuming equipment in a given building. With this evolution comes the possibility of optimizing the energy performance of the whole building and its energy-consuming equipment [3].

In terms of renovation, the energy refurbishment in old residential buildings can significantly reduce energy consumption [1, 4, 5]. Additionally, the equipment and its usage also determine the performance of a building. In order to draw a realistic picture of truthful energy consumption, an assessment of real-life energy utilization must be done, including all energy-consuming equipment in a given building.

Internet of things (IoT) can monitor energy consumption of home appliances using sensors, edge devices, cloud to perform continuous real-time monitoring. The technology will offer energy resource management to building owners, managers, and energy operators. It will give full disclosure of how we are consuming energy. It will allow us to make better decisions on consuming electricity to better consumption performance and increase energy savings.

1.1 Motivation

To inform users about electricity consumption EU has introduced energy labels that provide a clear and simple indication of the energy efficiency in products at the point of purchase. These labels come with the comparative scale from A, which means most efficient, to G (least efficient). This approach is a crucial

driver for users to choose the most energy-efficient appliance. On the other hand, we have manufacturers competing among themselves to create more efficient devices that will be labeled as highly efficient [6].

Firstly, it must be assessed in the laboratory to determine the efficiency and appropriate labeling of energy efficiency appliances. Studies such as [7, 8] show that devices used in real-life situations are using more energy than compared with those in a laboratory. The authors [7] did experimental work, comparing the energy consumption of various refrigerators, taking into account several usage conditions. The results show that 61% of appliances were using more electricity at home than in the laboratory. When it comes to real-life usage, many variables must be considered: how many times the fridge was opened per day, the ambient temperature, how much load is in the refrigerator, and so on. Due to these circumstances, it is a real challenge to create a realistic energy consumption prediction for particular devices.

One of the critical actions to understand real-life energy consumption is to monitor the equipment used inside buildings. In particular, appliances need to be assessed to understand the actual energy consumption. The ongoing energy transition brings the possibility of real-time energy resource management to building owners/managers and energy operators, with potential benefits for consumers, producers, and the environment. To better tap into this potential, stakeholders must continuously assess the energy performance of building energy systems and appliances, identifying areas where optimizations can be performed. Implementing this assessment capability requires real-time monitoring and control of the building equipment and major energy-consuming appliances. This functionality can be effectively performed by the Internet of Things (IoT) enabled sensors and devices that communicate with the cloud and execute control actions (thereby becoming intelligent devices).

Although modern intelligent devices and systems allow remote monitoring and control, most methods cannot assess their energy performance. This fault is even more prevalent in legacy equipment and appliances. Therefore, upgrading buildings, appliances, and technical building equipment with energy assessment capabilities is still a daunting challenge that must be met to reduce the energy consumption of buildings and their equipment.

This way, consumers will be able to manage appliances within day-to-day situations better. Wood G. et al. [9] claim that if users knew the actual energy consumption of each device, it would help to change the behavior of how these devices are being used, which would help saving energy. A solution for mentioned problems is implementing continuous energy monitoring of appliances using various sensors and IoT technologies that will support the accurate evaluation and assessment of energy consumption by devices and report detailed energy usage to the end-user.

1.2 Objectives

The main goal of this work is to implement a cost-effective and flexible prototype IoT solution to assess energy-consuming equipment. A set of sensors that will collect valuable energy consumption pattern data. This data will be used in the SATO platform to understand the energy utilization of equipment in real-life situations. The prototype will be used to monitor appliances that are commonly found in the household. Furthermore, the system will monitor the water consumption of devices that require a water supply, such as washing machines and dishwashers.

Monitored data will be used for real-time appliance classification. This information is precious, and we can supply the user with highly detailed appliance usage reports and the energy consumption of these appliances. Reports can include energy usage per various cycles. For instance, an appliance that can perform multiple cycles like a washing machine can provide energy and water consummation per specific cycle. This fine-grained data can be used for recommendation services. And will help users define better

settings for a specific appliance, minimize energy consumption, and improve efficiency.

Appliance monitoring with various sensors will produce patterns. Supported by the collected data, pattern approaches will be explored to draw a device/settings profile that can be used to find and predict appliance malfunctioning and faults.

1.3 Contributions

- Monitoring system

  Our proposed monitoring system is more advanced than in literature proposed monitoring systems due to an enhanced list of sensors that can provide more information about appliances, like washing machine water consumption. Furthermore, it is highly flexible and can be effortlessly implemented to new or legacy appliances for resource consummation monitoring.

- Appliance energy consumption dataset

  We are providing appliance consumption data that can be used for further scientific investigations regarding energy consummation investigations, appliance energy usage patterns, and appliance classification.

- Edge computing approach

  Our proposed monitoring system exploits the edge computing paradigm where some of the data preparation computing is executed at the edge relieving cloud resources for more essential computations.

- Highly accurate real-time appliance classification

  We can highly accurately classify the appliance using the data from our monitoring system. Our approach can classify appliances that are being monitored with low false-positive rates.

1.4 Structure of the document

The remainder of this document is organized as follows:

- Chapter 2 – Presents existing in the literature appliance monitoring approaches, reviews various sensors that can be implemented into the monitoring system, and describe the working principles of sensors. This chapter also discusses wired and wireless data communication and collection solutions. Because we are dealing with raw signals from the data, related work regarding signal cleaning and feature extraction is reviewed. Finally, this chapter reviews existing literature works related to appliance classification using machine learning approaches.

- Chapter 3 – Discuss the system requirements, a conceptual system architecture design, and hardware necessary for the implementation, as well as the software and firmware. We also discuss different machine learning algorithms and configuration which allows getting better results in this work.

- Chapter 4 - A prototype implementation is being presented. Discussion of the implementation includes used hardware, its capabilities, and the purpose. It also describes the work done regarding how the hardware has been implemented and the software and firmware tailored for the prototype. In this chapter, we also discuss the appliance classification approach, preparation methods, and used algorithms. Lastly, this chapter discusses prototype implementation to the SATO platform.

- Chapter 5 - Describes the experiments and evaluations regarding appliance classification and prototype capabilities evaluation.
- Chapter 6 - Presents conclusions and future work.

Chapter 2

Related work

In this chapter, we are reviewing literature regarding appliance monitoring approaches, their advantages, and disadvantages. Furthermore, we review existing and available market sensors' capabilities that might be adapted to our monitoring prototype and data communication protocols related to our work. Since we will be working with some raw signal data, we review signal processing approaches such as signal denoising and feature extraction from the signal. We are looking forward to implementing appliance classification approaches. Due to this reason, we include an appliance classification review regarding machine learning adaptation for appliance classification.

2.1 Appliance monitoring approaches

The condition and operation of appliances cannot be well inspected and assessed without an appropriate monitoring system. The goal of such a monitoring system is to reduce energy waste by taking energy-efficient measures like using appliances with more excellent energy performance, appropriate consuming timing, and eliminating unwanted and unnecessary energy misspend [10]. Collected data can deliver more precise information about energy consumption and engage users to reduce consumption and emissions. This energy consumption awareness will promote users for behavioral changes in the way that they interact with appliances [11]. Such a system can be made by using two approaches: the first approach is called Non-Intrusive load monitoring (NILM), and the second is Intrusive Load Monitoring (ILM).

2.1.1 Non-intrusive load monitoring

Non-intrusive load monitoring means that no additional sensors and devices are installed in the house. This method is relying on a single point of measure. NILM is capable of capturing energy usage of individual devices and can help detect high energy demanding devices [12]. This approach desegregates the power load passing through the electricity meter entrance and breaks this load for recognizing individual electrical appliances. Individual devices are being recognized by using various computational and analytical approaches like graph signal processing (GSP) [13], machine learning algorithms, or ensemble bagging tree (EBT) [14]. Supervised and unsupervised methods like a neural network, support vector machine, decision tree, Hidden Markov model (HMM) [12]. Figure 2.1 illustrates NILM approach combined with EBT processing.

NILM framework consists mainly of four steps [12]:

- Data acquisition and processing where raw data is collected and denoised;

- Event detection where an event happens when the device is changing its process state hence the power and current variation occurs;
- Feature extraction where load signatures are extracted using different algorithms (e.g., Fourier transform). It helps to distinguish different appliances
- Load recognition where load features are compared with the features that are collected in a database and then appliance switching mode is being obtained in the database

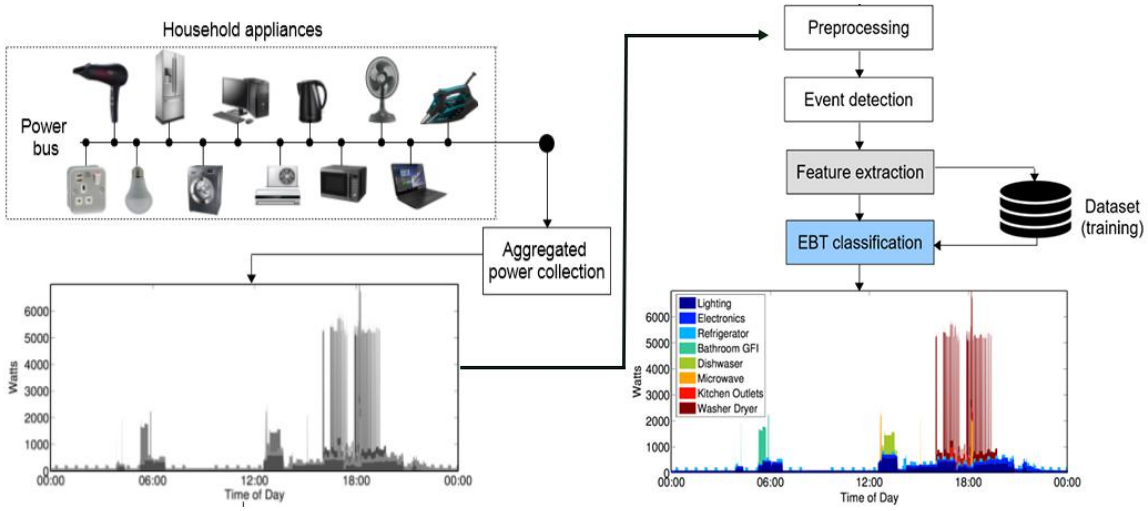


Figure 2.1: One of many NILM based approaches using EBT classification.

2.1.2 Intrusive load monitoring

In the ILM devices must be physically placed inside the environment, somewhere close to the appliance that is being monitored [15].

The level of intrusiveness can be scaled into three main parts: sub-metering where a load meter is monitoring a zone of appliances, smart plug where each device has a dedicated sensor or set of sensors to monitor a single device, and smart appliances that might have embedded meters in the appliance [16, 17]. Typically, an ILM framework is divided into three stages: [15, 16]:

- monitoring single appliance where sensors detects if the appliance is actively used or it is turned off, as well as the appliance location, if sensor (e.g., a smart plug) location is registered;
- Middleware phase where software is interpreting the status depending on the data received from sensors;
- Appliance status phase where the system shows the appliance status control and better energy management.

Commonly, ILM implementation relies on data collection, signal processing, statistics, and Machine Learning (ML) technologies. The key task of an ILM implementation consists of the comparison of unseen signal patterns against previously seen and known signals. ML models are created using collected historical data associated with labels, like a key, value pairs. Those ML models allow identifying, for instance, appliance suppliers, brand identification, and state identification [18].

2.1.3 Common load monitoring system stack

The system architecture of an NILM/ILM solution is illustrated in Figure 2.2. The bottom layer consists of sensors, microprocessors, and the monitored appliances that will fall in one of the categories mentioned before: sub-metering, smart plugs, or smart appliances.

Since sensors may have no computational power and microprocessors are too weak to compute power demanding tasks, data can be passed to edge devices to make necessary calculations [18]. This implementation could also be combined with a relay switch to disconnect the device from power altogether. This is important because in the modern world, many household devices remain on standby mode most of the time. According to International Energy Agency (IEA), typical standby power accounts for 3% to 11% of total annual electricity consumption [19].

Transmitting huge amounts of data to a cloud server becomes expensive in bandwidth and leads to increased latency and cloud resource occupation. These issues can be solved using edge computing. Edge device acts as a bridge between sensor data and the cloud. Many tasks can be performed in the edge device, including basic data processing and filtering. Most of the time, it is not necessary to send every single sensor reading to the cloud. That is where edge computing comes in handy. It filters, prepares the data, and sends only the most necessary one for big data analysis in the cloud. [18, 20, 21].

Depending on the monitoring approach, various devices can be used to monitor the appliances like current sensor, voltage sensor, smart meter, a smart plug, fluke meter, ZigBee energy monitor (ZEM-30), energy detective (TED-pro), and power quality transducers [16]. For data transmission and communication, Internet of Things (IoT) standards are used, such as WiFi, ZigBee, Bluetooth, LoRaWAN, Z-Wave, Cellular 1-5Gs [22].

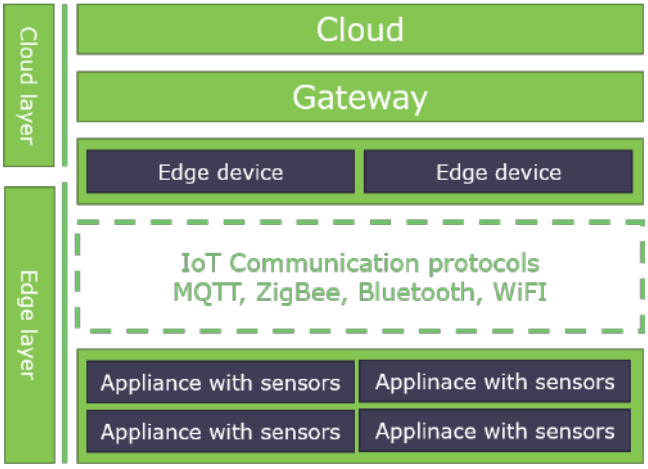


Figure 2.2: Illustration of ILM architecture from IoT perspective.

2.2 Sensors

A sensor could be described as an input device that receives a stimulus and outputs an electrical signal concerning a specific physical quantity (stimulus). In this context, a stimulus can be quantity, property, or a condition received by a sensor and converted into electrical signals. Examples of stimuli can be light intensity and wavelength, sound, force, acceleration, distance, motion, velocity, temperature, chemical compositions. Generally, a sensor is seen as a translator between typically non-electrical into electrical

values. The sensor can provide measurements in the form of voltage, current, or charge. These forms can be branched further into frequency, intensity, polarity, amplitude, phase, or digital code. Hence, the sensor has input properties (stimulus of any kind) and electrical output properties. Sensors can be divided into two groups direct and hybrid. A direct sensor converts stimulus into an electrical signal, while a hybrid additionally needs one or more transducers before the sensor can generate an electrical signal. The transducer is a converter of any one type of energy into property into another type of energy, or property [23]. Figure 2.3 illustrates a sensor that uses multiple transducers to convert stimulus into a signal.

Sensors themselves are usually not intelligent objects. Suppose we want to read the sensor's electrical signal output. In that case, it has to be connected to an intelligent device like a micro-controller unit (MCU) that can convert electrical signals into more meaningful values [24]. Following part of this section reviews available sensor types that will be used in this thesis to collect the data of electrical appliances. Many of those appliances are easily accessible, low-cost yet producing high accuracy measures, meaning that ILM approach is more feasible than ever.

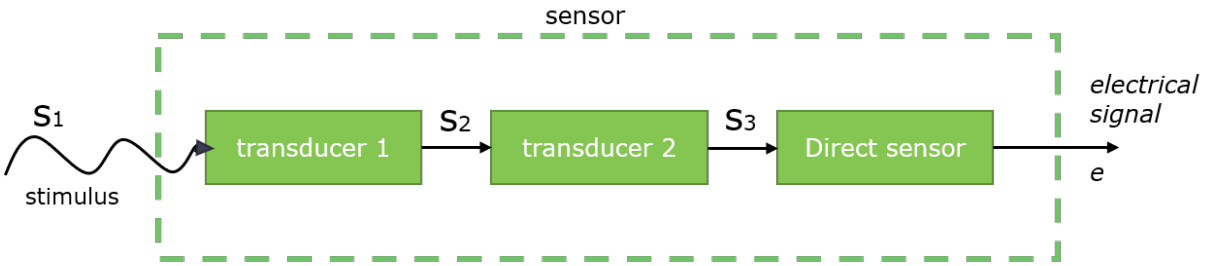


Figure 2.3: Sensor with incorporated multiple transducers. S1,S2,S3 represent various types of energy. And direct sensor produces electrical output (e).

In this work, we used a set of sensors that allows the monitoring of appliances. Since some of these sensors have specific characteristics, in this section, we review them, and mention their main characteristics, as well as how they work for the proposal of this work.

2.2.1 Current sensors

The current sensor is used to measure electricity consumption. There are quite a few low-cost current sensors available in the market. The main differences between them are sensitivity, mounting method, and measurement values that these sensors provide. Some of them can provide more information about the energy consumption of an electrical device, while others can measure only one electricity feature. Next, we will review four energy metering sensors that are shown in Figure 2.4

Figure 2.4: Low-cost current sensors from left to right: ACS712, WCS1800, SCT013, PZEM004T

ACS712 is a low-cost current sensor that comes in three different current rate versions. It can measure 5 amperes (A), 20A, and 30A, and it has a low error rate of 1.5% at 25ºC for optimal operation. ACS712 works at 5 volts (V), and it has an output sensitivity that ranges from 66 to 185 mV/A. Its output voltage is proportional to AC, or DC currents [25].

ACS712 is an analog sensor, meaning we need to convert the sensor-provided voltage into current (A) readings. Its data-sheet offers linear output, which makes conversions from analog to readable value of current simpler. This sensor is based on a linear Hall sensor. The copper conduction paths are located near the surface of the Hall plate. When current is flowing through the copper conductor path, a magnetic field is created, which is sensed by the Hall transducer that produces a voltage which is then converted into a current reading by a microcontroller [26]. This sensor has a disadvantageous mounting. The wire has to be split and connected in series with the sensor to measure the current [25], as illustrated in Figure 2.5.

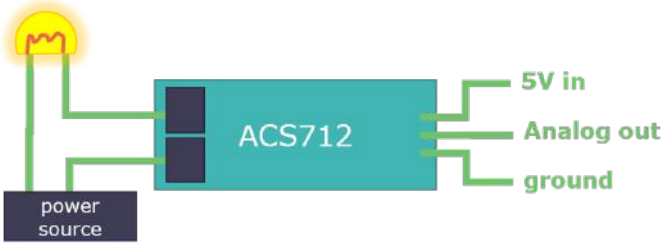


Figure 2.5: ACS712 current sensor mounting diagram.

WCS1800 is another Hall-Effect-based linear current sensor. On the other side of ACS712, WCS1800 does not require cutting the line (wire) for current measurement. This design allows designers to monitor any current path without breaking or changing the original system layout. The "hot" power cable is simply passing through the sensor, as illustrated in Figure 2.6. It has a wide sensing current range of 0 to 35A at 5V. The module can be operated at a voltage range from 3V to 12V. [27].

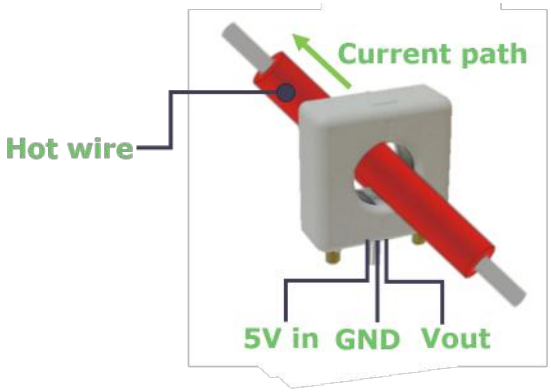


Figure 2.6: WCS1800 current sensor mounting diagram.

In research, [25] WCS1800 measurements were compared against high accuracy dedicated power meter. Results show that when the current is below 1 A, the error can vary from 3% to 5% depending on the calibration approach. For higher values of current (from 1A to 20A), the error is bounded by 1.5%.

This sensor gives analog voltage level output, which ranges between 0.8 to 4.2V. Depending on the MCU, the output range must be scaled down or increased to represent correct current values. Because

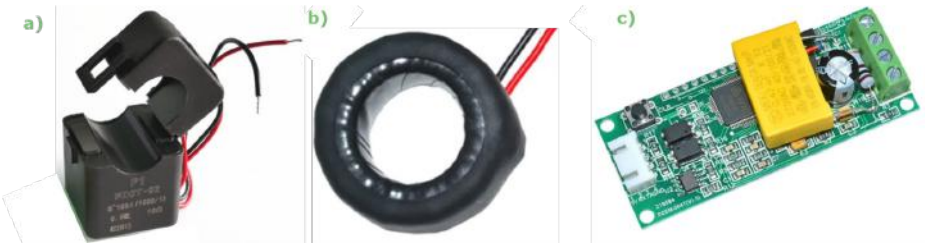


Figure 2.8: a) iron core clamp, b) solid iron core, c) PZEM004T current sensor module.

some MCUs are operating on higher voltage and some lesser like 3V, for example, meaning that the sensed sensor output can not be higher than the operating voltage.

SCT013 is a split core clamp sensor that can mount on a measured wire, as illustrated in Figure 2.7. This sensor uses the induced voltage principle. It was designed only for AC current, but there are many different models to address several current rates: 5A, 30A, 50A, 100A [25]. The SCT013 sensors are current transformers, instrumentation devices that provide a measurement proportional to the intensity that a circuit cross. The measurement is made by electromagnetic induction. The sensor accuracy can be off by only 1-2%. To ensure the highest accuracy, it is critical to confirm that the core has been properly closed. Even a tiny air gap can cause a 10% deviation [28]. SCT013 readings were compared with a more accurate power meter, and the results show that from 0 to 2A, the absolute percentage error is from 2% to 4%, but as the current gets more intense from 4 to 20A, the absolute percentage error is less than 1.5% [29, 25].

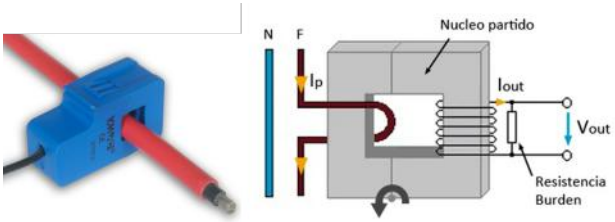


Figure 2.7: SCT013 current sensor mounting.

PZEM004T sensor is more complex and sophisticated compared to other sensors that we have reviewed. It can measure power, voltage, frequency, energy, and current. PZEM004T can measure current up to 100A. Similarly, like SCT013, the wire is passing through its iron core to capture current signals, a sensor can come with two versions of core, in Figure 2.8 (a) clamp iron core, (b) solid iron core [25, 29]. One of the wires that carry the current (usually the AC power phase) goes through the current transformer (CT) (iron core) so that the current can be measured. Power and frequency are measured with higher accuracy because hot and neutral wires are connected to the sensor to measure line voltage, and frequency [30]. It is a digital sensor whose internal circuitry takes care of the calculations and returns already processed data to the microcontroller via serial communication. It can measure voltage in a range between 80V to 260V, power measuring range from 0 to 23kW, frequency measuring ranges from 45Hz to 65Hz. Based on the experiment done in [25] PZEM004T, compared to other reviewed sensors, performs best when it comes to the accuracy of measurements. From 0 to 2A, the error was less than 2.3%. From 2A to 20A, the absolute percentage error was varying from 0% to 0.5% error [25, 29]. The only downside of this sensor is installation complexity comparing to the previously mentioned sensors.

Figure 2.9: MEMS accelerometer sensors. From left to right: ADXL-335, ADXL-345, MPU-6050.

2.2.2 Vibration sensors

We are paying attention to vibrations sensors because some home electrical appliances are resonating due to machinery inside. Objects like motors can generate oscillation of an appliance. As an example, we could take a washing machine. The electric motor is spinning a tub part and causes the machine to oscillate. To measure the vibration patterns, excitation intensity, we can use vibration sensors. In recent related work regarding vibrations, we can meet Micro-Electro-Mechanical Systems (MEMS) accelerometer sensors. They are popular due to their low cost, high availability, and reasonable accuracy. MEMS accelerometers which use are based on single proof of mass for sensing 3D accelerations, have a tiny footprint and are low cost. This type of interconnected mechanism is structurally simpler in terms of suspension design and reduces overall complexity but is more prone to have undesirable noise in the signal. The second design approach using multiple proof-mass sensors prone to have a larger footprint. However, they are superior regarding sensitivity and noise floor. Large size is required to obtain good performance [31]. Further reading on MEMS accelerometer working principles can be found in Appendix A.1.

MEMS sensors:

In the literature review most common low-cost, low-power MEMS accelerometers that were used are ADXL-335, ADXL-345, MPU-6050 [32, 33, 34, 35, 36, 37], Figure 2.9 illustrates MEMS accelerometer sensors.

ADXL-335 is a small, low-power module that supports a 3axis MEMS accelerometer. It is designed to measure with a minimum full-range acceleration of +- 3 g. ADXL-335 can measure shock, motion, and vibrations. Sensor bandwidth can be selected depending on application needs, with arrange of 0.5Hz to 1600Hz for X and Y axes. It has an operating range from 0.5Hz up to 550Hz [38], combined with an analog sensor that uses comb finger type differential and has excellent sensitivity, low drift, and good noise performance. The board comes with a voltage regulator model 662K, which regulates the voltage input to the board and makes sure it will always be powered with the constant 3.3V, which ensures a better quality of measured signals. This module can be implemented in many applications like controlling unnamed aerial vehicles or engineering applications like device health monitoring [39].

ADXL-345 is yet another ADXL family module with a triple-axis, small footprint, low-cost and low-power, a digital accelerometer that can track vibrations of the X, Y, and Z-axis. This module can measure accelerations with a minimum full-scale range of +-2 g/ +-4 g/ +-8 g/ +-16 g/. It includes features user salable resolution, I/O voltage range 1.7V to Vs. ADXL-345 is connected using SPI (3 and 4 wire) and I2C digital interfaces and can operate in a wide temperature range (-40°C to +85°C). An application of ADXL-345 can be on detecting activity/inactivity of a monitored device depending on applications [40]. However, it can be applied for a wide range of applications, e.g., characterization of kinetic energy

harvested by wireless IoT nodes, implementation of vibration-powered sensing for machine condition monitoring, detecting induction motor faults caused by mechanical or electrical faults based on IoT using the wireless vibration sensing nodes. ADXL-345 accuracy is +-4.6%, measuring a range frequency of 3.2kHz [41]. ADXL-345 readings were compared against standard accelerometer, and results show that low-frequencies (<60Hz) have a reasonable error and can be used in low-frequency applications [39].

MPU-6050 module sensor that has X Y Z axes gyroscope and X Y Z accelerometer. The key specifications of the MPU-6050 output frequency are fixed at 1kHz, and the bandwidth is 500Hz. However, MCU can pragmatically set the sampling rate between 4Hz and 1kHz by changing the delay time. This module has an integrated 16-bit analog-digital converter (ADC) that encodes analog signals into 216 output levers ranging from -32768 to +32768. It has four measurement ranges +-2 g/ +-4 g/ +-8 g/ +-16 g/ that are available. Measurement range is increased when measurement resolution is decreased [39, 42, 43]. The module operates at a voltage between 2.375 and 3.46V. Likewise, in other modules, MPU-6050 has integrated power regulator KB33 that allows to power the board with 5V. It can be used in applications such as damage detection based on auto-regressive models, and damage-sensitive features [39].

Understanding technical specifications:

ADXL-345 and MPU-6050 has a selectable range of +-2 g/ +-4 g/ +-8 g/ +-16 g/, which means the higher the range, the higher the acceleration that the sensor can measure. With a measurement range of +-2 g/, a sensor can measure acceleration up to 19.6m/s^2 (2*9.8 m/s), meaning that the maximum range that can be measured is 157 m/s^2 in either orientation along the X, Y, Z axis [44].

As mentioned before, these sensors are capable of measuring static acceleration (gravitational) and dynamic (oscillations). Sensor output response is dependent on its position, as shown in Figure 2.10.

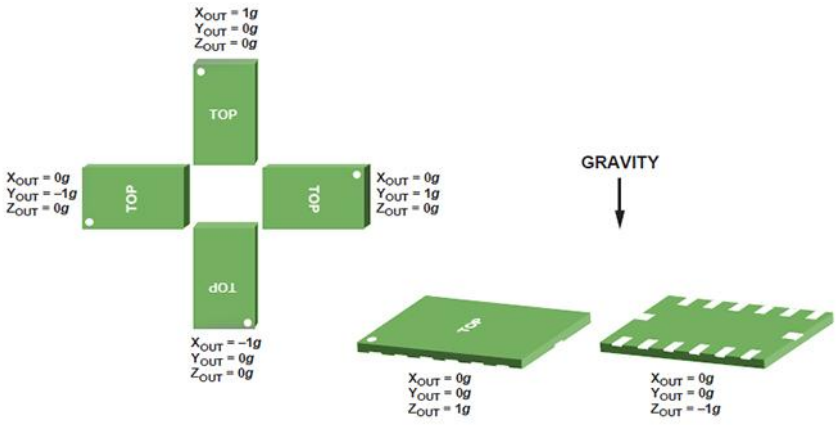


Figure 2.10: Sensor output based on board positioning.

2.2.3 Water-flow sensor

To monitor devices that consume and expel water, e.g., washing machines, dishwashers. We can use water-flow sensors. Various types of flow meters are available in the market that is mentioned in Table 2.1 [45].

Table 2.1: Categories of various flow meters based on assorted principles.

Principle	**Types**
Differential pressure	Orifice Plate type of meter, Rota Meter, Flow Nozzle, Pitot type Tube, Elbow Tap, Venturi Tube
Positive Displacement	Oval Gear type, Nutating Disc type, Rotary Vane type, Reciprocating Piston
Velocity	Turbine type, Vortex Shedding Electro-magnetic, Ultrasonic Doppler type, Ultrasonic Transit Time type.
Mass	Coriolis , Thermal
Open channel	Weir, Flume

Some meters that are mentioned in the Table 2.1, e.g., velocity meters use a sensor that is measuring liquid flow rate based on the passing through speed, ultrasonic sensors that brings two principles together: transit time measurement and Doppler effect, but these sensors come with a high price, and expensive maintenance [45]. Our focus went on low-cost sensors that are easily available yet having considerably good accuracy. Researches [46, 47, 48, 49, 50] used a YF-S201 low-cost, low-power, water-flow sensor that is based on Hall-Effect. As illustrated in Figure 2.11, sensors have the little pinwheel with an attached magnet to it, and Hall Effect transducer, when water is passing through the sensor, the pinwheel starts to spin, triggering the Hall transducer, which examines the rotations of the pinwheel. Further reading about hall-effect regarding water flow can be found in Appendix A.2. Hall sensor starts generating square wave pulses as the water keeps flowing. The amount of water can be measured by counting the pulses per window of time. Sensors generated output is being sent to a micro-controller that is programmatically converting pulses to flow rate and water quantity. The frequency of pulse has a linear relationship with the flow rate. Each pulse is approximately 2.25 milliliters. [46, 47]. YF-S201 sensor that was used in mentioned references can measure 1 to 30L per minute with maximum water pressure of 2.0MPa [47]. Hall-effect-based sensors have several benefits. They are highly durable, works with high-speed operation over $<$100kHz, operate with stationary input, resistant to contamination, like dust, dirt [45].

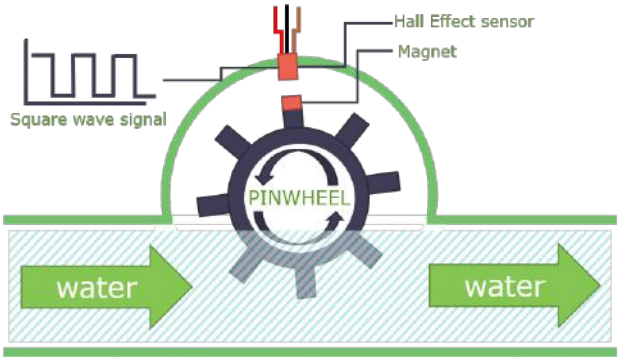


Figure 2.11: Water-flow sensor based on Hall effect working principle.

2.2.4 Temperature and humidity sensors

We are reviewing some low-cost temperature sensors to measure water temperature, ambient temperature, and ambient humidity. One of the available sensors to measure humidity and temperature is **DHT11** sensor. This sensor is combined with humidity and temperature digital output. It is based on one wire serial communication [51]. It can measure humidity between 20% and 90% Relative Humidity

(RH) within operating temperature range of 0ºC to 50ºC with an accuracy of +/-5%. The temperature measurement range is from 0ºC to 50ºC with an accuracy of +/-2ºC. Either one returns an 8-bit resolution to the connected microcontroller. A module can be powered with a voltage range of 3.3 to 5.5 V [52]. Research [53] used this sensor to create a temperature and humidity reporting node based on the IoT approach. DHT11 sensor was communicating with Raspberry Pi computer that was later sending sensor data to the Cloud of the IBM Bluemix through Node-RED platform.

From the same family, we have **DHT22** sensor. Similarly to DHT11, consists of thermistor (temperature sensor) and capacitive transducer for measuring humidity as illustrated in Figure 2.12 [54], further reading about the working principles of these sensors are described in Appendix A.3. Both sensors (DHT11,DHT22) work in the same principle, but DHT22 has improved the accuracy and bandwidth of measuring temperature and humidity. It can measure humidity in the range of 0 to 100% RH, with an accuracy of +/-2%. Temperature ranges from -40ºC up to 80ºC, with an accuracy of +/-0.5ºC [55], which is a significant improvement comparing with its predecessor. It has an implemented microcontroller of 8-bit, which converts analog sensor values to digital, meaning that MCU already receives a digitized signal of humidity and temperature as serialized data [56]. This sensor was used in research [57] to create a local weather station. DHT22 and anemometer were connected to the Arduino Uno MCU board, which was sending the data via Bluetooth to an Android smartphone. Another approach by [56] was to create a smart home application where ESP8266 MCU would send the humidity and temperature readings to Raspberry Pi via MQ Telemetry Transport (MQTT) protocol which then would send it to Node-RED application for data representation based on a web dashboard.

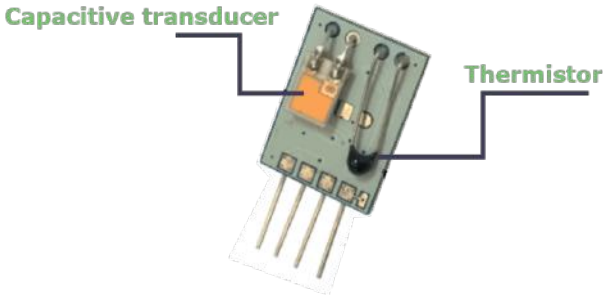


Figure 2.12: The inner part of DHT22 sensor.

LM35DZ is yet another temperature sensor based on analog data output. The analog output is directly proportional to the temperature in Celsius 10mV per degree Celsius rise in temperature, meaning that the MCU board is able to convert the voltage supplied by the sensor into a meaningful temperature value. LM35DZ temperature sensing range is from -55ºC to 150ºC scale, with an accuracy of +/-0.5ºC [58, 59]. A research [59] used this sensor to create an application of rooms/flats remote monitoring. Arduino MCU board using a General Packet Radio Service (GPRS) module was sending the sensor temperature data to an Android application. A research [60] used LM35DZ for the health monitoring application, where a person's body was being monitored by a heartbeat sensor and a temperature sensor. Sensors were connected to an Arduino Mega Board that had connected a Bluetooth module which then kept sending sensors data via Bluetooth to an Android application.

DS18B20 is a digital thermometer that provides 9-bit to 12-bit temperature measurements in Celcius. The sensor communicates with an MCU via a 1-Wire® bus, meaning it needs only one data line and ground for communication. It has the capability of drawing the power directly from the data line, eliminating the need to be externally powered by a supply. Each sensor has a unique 64-bit serial id,

meaning that multiple sensors can be connected to the same communication line. Sensor features include temperature measurement range from -55ºC to +125ºC, with an accuracy of +/-0.5ºC in a range from -10ºC to +85ºC. Sensor resolution can be programmable [61, 62]. The sensor has two available versions, one is simple, and another is designed to be waterproof, as illustrated in 2.13 [62]. A research [63] used this sensor in a design of an automatic water dispenser for blind people. DS18B20 was used to measure and control the water temperature of a dispenser by using an Arduino MCU board.

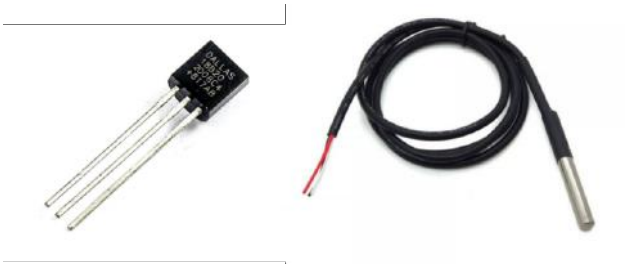


Figure 2.13: Left side, regular DS18B20 and in the right side waterproof version of DS18B20.

2.3 Data communication

To read and collect the data from sensors, it is necessary to use some MCU, a Single Board Computer (SBC) or a combination of both. A typical approach to do this is to connect sensors to MCU and send sensor readings from MCU to an SBC, Personal Computer (PC). They can be sent wirelessly to the cloud or stored inside a unit like an SD memory card to be processed later (non-real-time data processing). During the literature review, it is being noticed that Arduino boards are the most popular among various researches. It provides rapid development due to the high number of different peripherals, many types of boards with different features, wired and wireless protocol support, and a vast community. That is why this work is mainly mentioning cases related to Arduino boards as MCU.

Sensor readings and data from the MCU can be shared using several approaches. Using IoT solutions that vary from the network technology to the communication protocol, like MQTT, ZigBee, Bluetooth, Wi-Fi, LoRA [22]. The other approach is to send the data using wired connections, e.g., Universal Serial Bus (USB), Inter-Integrated Circuit (I^2C), Universal Asynchronous Receiver-Transmitter (UART). The following part of this section will be reviewing related work regarding applications and capabilities of mentioned protocols and technologies.

2.3.1 Transmission solutions for wireless data collection

Solution based on WiFi:

The work done in [41] regarding accelerometer data collection used WiFi for data transmission. The hardware used in work has a wireless internet connection. Hence collected data can be sent to a computer, edge device, or cloud. The authors were sending x, y, z-axis data from Arduino mega connected with a WiFi module to a mobile application called “Blynk App.” This data was also sent and stored in the cloud application called “ThingSpeak,” which was plotting curves of received data from the Arduino. Mentioned approach is illustrated in Figure 2.14, where Arduino mega (c) is connected with NodeMCU ESP8266 WiFi module (a) and ADXL-345 MEMS accelerometer (b).

Another research reported in [64] proposed the design of a WiFi-based single-phase smart energy meter. System architecture is illustrated in Figure 2.15. The system used a current sensor and voltage

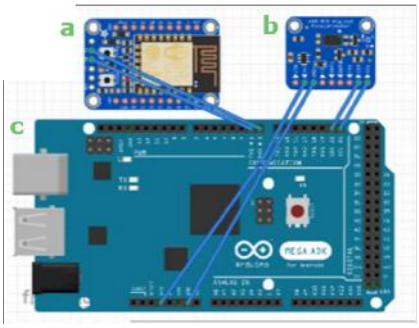


Figure 2.14: Wireless vibration meter architecture approach.

sensor connected to the Arduino MCU, which had an ESP8266 WiFi module. The system was sending energy consumption data to the server. Data could be viewed over a web-based panel.

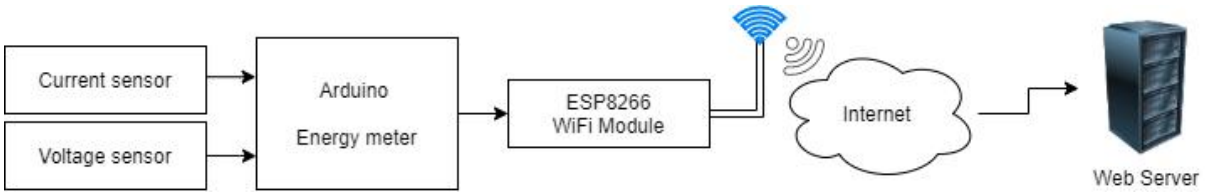


Figure 2.15: Smart meter architecture of the WiFi based single phase energy meter for IoT.

Many projects that are based on wireless communications use the NodeMCU ESP8266 board for wireless communication. The module has a built-in WiFi that supports 802.11b/g/n. WiFi 802.11n, or "WiFi 4", was released in 2009. The main problem with older WiFi versions is the high energy demand. Although nodes used in projects are wireless, in reality, they are pseudo-wireless nodes. They need a constant power source, battery-powered nodes draw the energy rapidly, and these approaches require frequent battery replacement or constant power supply to function properly. Most recent Wifi protocol is 802.11ax also known as "WiFi 6" released in 2019 [65].

Due to the energy consumption problem in IoT-enabled WiFi-based appliances, a Low Power WiFi (WiFI HaLow) protocol was developed concerning IoT requirements. It is a wireless communication MAC and physical layers protocol enabling a long-distance communication of wireless nodes with reduced power consumption compared to regular WiFi [22]. WiFi HaLow was released in 2017. It uses a sub-band of 1GHz, has a wide range and strong signal penetration. It is based on a reduced clock rate of the WiFi 802.11ac, also known as "WiFi 5". Its features include long outdoor WiFi up to 1km, supports data rates up to 15Mbps, with enough throughput to support Transmission Control Protocol (TCP)/Internet Protocol (IP). It is scalable and supports latest WiFi security Wi-Fi Protected Access 3 (WPA3). Unfortunately, there are still no commercially available boards for developers to implement into IoT projects [66]. Arduino released a Nano 33 IoT board in 2019, and it is using the same WiFi version of 802.11b/g/n 2.4HGz as NodeMCU ESP8266 that was released in 2013.

WiFi HaLow includes use cases such as smart meters and sensors, smart grids, environmental and agricultural monitoring, automation of industrial processes, indoor wellness tracking systems, elderly care systems [66].

When using wireless connection solutions like WiFi, it is worth considering using Message Queuing Telemetry Transport (MQTT). It is a lightweight protocol that enables communication process between

IoT devices and a network. With this protocol, communications such as Machine to machine (M2M), server to server, and machine to a server might be established. It works over Transmission Control Protocol / Internet Protocol (TCP/IP). MQTT is advantageous due to its lightweight architecture and can be applied to many MCUs with access to the internet using various modules like WiFi, Ethernet, or Global System for Mobile Communications (GSM). It supports communications over limited bandwidth and unreliable networks. It can exchange packets that do not exceed 256MB. Furthermore, it is suitable for operations with limited power and processing capabilities [22].

The architecture of MQTT can be split into two main components. The first one is a client. Clients can be publishers or subscribers. It always establishes the connection to the MQTT server, also known as a broker. Clients can publish messages via specific topics, subscribe to topics of interest, unsubscribe topics, and detach from the MQTT server. The second component is a broker. Its task is to control the distribution of messages, like a mail carrier, where it receives all the messages and distributes them to recipients (subscribers). Broker tasks are to accept client requests, receive messages published by users, process subscribe and unsubscribe requests, distribute messages. The most used brokers are: Mosquitto, Minimal message broker (RSMB), MQTT.js, HiveMQ, and VerneMQ [67]. Simplified communication between clients and brokers is illustrated in Figure 2.16.

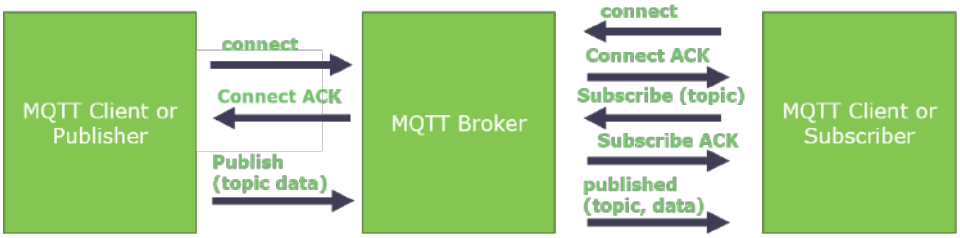


Figure 2.16: MQTT based communication between publisher and subscriber.

A research [68] implemented solution for Building energy monitoring and controlling system using Lora Modulation and MQTT protocol. The experimental system has two end nodes and one gateway. Each end node has implemented Arduino Uno MCU, LoRa Shield and a link to the gateway. To monitor energy, nodes has ACS712 current sensor, ZMPT101B to read voltage and a relay module for load control. To transfer the data to the ThingSpeak dashboard, researchers used MQTT protocol. They were able to see the data on the MQTT dashboard application on Android. The end system measured the voltage, current, power, energy of connected appliances and sent the data over MQTT to the users' dashboard.

In the work of [69] IoT energy monitoring system was designed based on a low-cost PZEM-004T sensor and ESP8266 Wemos D1 MCU. The system tracked voltage, current, power and used the energy of an electrical appliance from multiple nodes. Raspberry Pi was used as a local server to store the data in time series inside InfluxDB, and it also acts as a Mosquitto MQTT broker. Data collection inside the server was based on MQTT protocol communication. Grafana was used to create a dashboard for the energy metering system. Mentioned system architecture architecture illustration can be found in Appendix B.1.

Solution based on Bluetooth low power protocol:

Bluetooth can be classified into standard Bluetooth or Bluetooth low energy (BLE). Standard Bluetooth can provide data throughput up to 3Mbps. BLE was created for applications compatible with low-range wireless communication, low-data rate, and energy-constrained applications. It is designed to

be low cost and with simplified complexity of radio transceivers [70].

In comparison, it uses less energy than WiFi, but the BLE drawback is lower bandwidth compared to WiFi. This trade-off has to be considered when developing a wireless system application. Maximum throughput varies from 58Mbps to 236Mbps for BLE 4.2. BLE protocol stack has three main layers: **controller, host**, and **application** illustrated in Figure 2.17 [71].

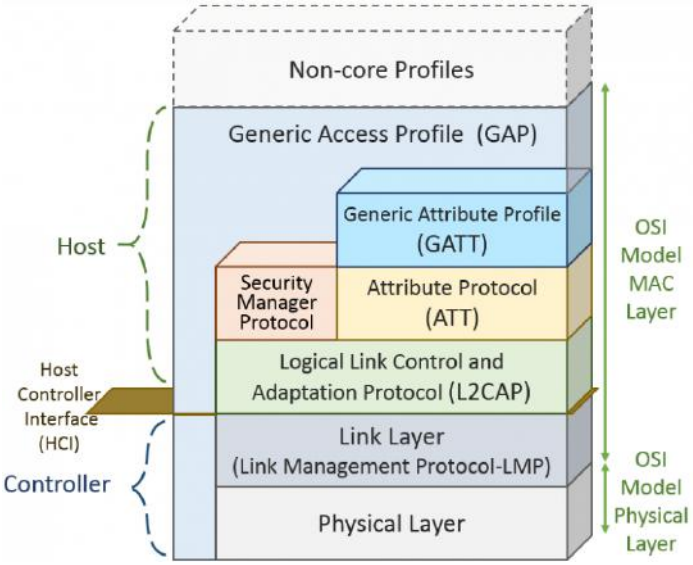


Figure 2.17: Bluetooth low-energy stack.

The physical layer controls radio transmission/receiving, where at the Physical layer, BLE defines 40 Radio Frequency (RF) channels in the 2.4 GHz Industrial Scientific Medical (ISM) band. Three of them, the advertising channels, are explicitly assigned for advertising and discovery services. The remaining 37 data channels are intended for bidirectional exchange of data between devices. The adaptive frequency hopping method is also used over data channels to minimize interference from other technologies (Bluetooth® and Wireless LAN) in the 2.4 GHz ISM Band. The data transmission rate is 1 Mbit/s. The Link Layer is a hardware-firmware co-implementation. It is responsible for various tasks such as managing the physical BLE connections between devices, defining packet structure, supporting all Link Layer states such as Advertising, Scanning, Initiating, and Connecting Master and Slave in addition, providing link layer-level encryption. L2CAP layer "Logical Link Control and Adaption Layer Protocol" works as an interface between the Host and Controller. It provides connection-oriented and connectionless data services to the upper layer protocols using segmentation, multiplexing, and reassembly capabilities [70]. Latest BLE can support star, tree, and mesh topologies. Some researchers exploited mesh topology to integrate applications such as: intelligent lighting, beacons for motion detection, smart office and home automation, remote controls, health monitoring, intelligent door locks, smart home appliances, intelligent wearable devices [70].

Research presented in [72] focused on developing a load monitoring system based on BLE communication. The system consists of current sensor SCT-013, voltage analog sensor AA-091ABN, an MCU board, and BLE modules. MCU was used to compute the analog values and convert them to electrical variables. The approach provided voltage, current, power factor, real power, apparent power electricity readings. This data is sent to the server through the BLE network. The project used Bluno MCU board has integrated TI CC22540 BLE module. The server will be storing data received from the MCU for later data processing. The proposed architecture illustration can be found in Appendix B.2.

Solution based on ZigBee protocol: ZigBee by design is a short wireless communication protocol. It can be used as a Wireless Personal Area Network (WPAN). This protocol aims to provide a simplified and cheaper WPAN network than standard technologies such as Bluetooth and WiFi. ZigBee is combining features such as high scalability, low power, high security, and robustness. It is ideal for wireless sensors networks in the IoT and M2M applications. There are three types of devices in the ZigBee network system: coordinator, router, and end device.

- **Coordinator** node is processing and storing information when data is being received and transmitted. When the network is initialized in behaves as a router that contains the entire stack.
- **Router** node is an intermediary device that allows data to pass from one device to another it is participating in message routing.
- **End device** is restricted to communicate only it parent nodes. It does not route messages inside the network, due this reason wireless end nodes have a longer battery life.

This protocol supports star, tree, and mesh topologies (Figure 2.18). In **star topology** all nodes communicate with only one central node, which is the coordinator. If a packet travels from one node to another, it must it is routed by the coordinator. **Tree topology** means that the coordinator can start the network. It can be linked to several routers and end devices. These routers might also be connected to other routers and end devices concerning the tree hierarchy. **Mesh topology** is when the coordinator is linked to other routers and end devices, many routers can be connected together, including end devices in that network. The advantage of such an approach is flexibility. If the packet is blocked, it might find another path to reach the desired destination.

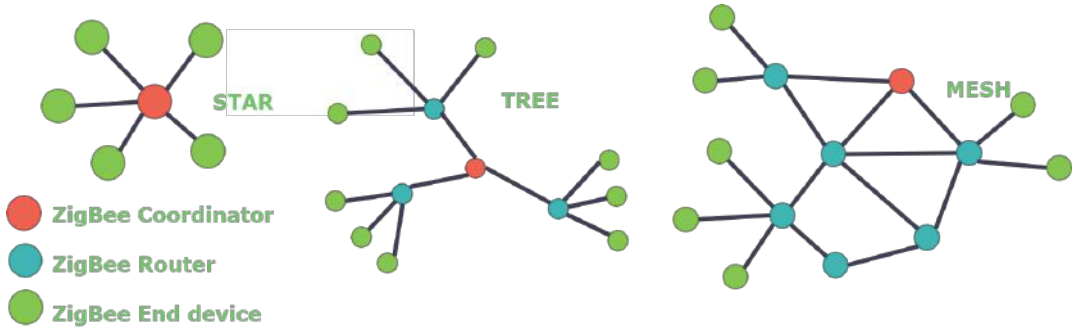


Figure 2.18: ZigBee supported topologies.

A research [73] of continuous monitoring of indoor environment quality had yet another solution for data collection and transmission. The architecture was based on multiple wireless nodes. Each Arduino MCU-based node contains multiple sensors and transmits the data to the computer using ZigBee protocol supporting Xbee modules. Data transmission is based on the ZigBee communication protocol. "Volttron" software is taking care of data visualization and processing. The scheme of this data collection approach can be found in Appendix B.3. XBee module features include: indoor range up to 30m, outdoor (clear line of sight) up to 100m, throughput (theoretical) 250kBps [74].

In work, [75] a ZigBee network-based clamp-on IoT energy meter prototype was developed. For the prototype, the XBee S2C module was used to communicate and transmit the data inside the network. The energy metering prototype used 100A SCT-013 current clamp sensor. An approach used star topology for communication. End nodes were sending sensor data to the central coordinator node. Each end node

consists of Arduino Uno MCU, current sensor, and XBee module. The described system schematics illustration can be found in Appendix B.4. Raspberry Pi SBC was communicating with the Coordinator node using serial communication. A python script was developed to save serially incoming data from a coordinator. A web interface was developed using Flash python library to represent collected energy consumption data.

Solutions based on LoRa/LoRaWAN: LoRa was created for long-range and low-power wireless communications. It is based on an unlicensed radio spectrum in the industrial, scientific, and medical radio bands (ISM). It aims to reduce network repeaters, reduce device costs and increase battery life on wireless nodes. To achieve low power, LoRa uses Chirp-spread-spectrum (CSS) modulation. Long Range Wide-Area-Networks (LoRaWAN) protocol based on physical layer communication protocol. It was developed to face the IoT long-range communication challenges [76]. It enables communications for services and applications like M2M, smart cities, industrial applications. Smart cities require long communication distances ranging from 2-5 Km in the urban area and up to 15km in suburban areas. LoRa can handle processes over large networks that contain enormous amounts of intelligent devices. Its throughput varies from 0.3kbps to 50kbps in the full-duplex mode [22]. However, it is essential to note that to achieve a fair data transmission rate, it has to respect the duty-cycle of 1%, meaning that if data transmission is lasting 50ms, the node has to wait for 99x50 ms (around 5 sec) before sending another message. This means that LoRa is not suitable for applications that require constant and rapid data transmission [77].

Due to reaching comprehensive range coverage, it is used for smart cities, remote location monitoring, it LoRa is a low-power consumption protocol providing longer life for wireless battery-based sensor nodes. A research [78] implemented LoRa Node to monitor remote wind and photo-voltaic power generator performance. LoRa node was connected to the energy device and reported this data to LoRa Gateway based no Raspberry Pi SBC, which stores the data into MongoDB database. Researchers developed a web-based monitoring dashboard where the performance assessment can be tracked remotely.

2.3.2 Wired data collection and protocols

A work regarding an Arduino-based long-term indoor environment data collection project that used wired data storing approach is proposed in [79]. It proposes creating independent nodes consisting of Arduino Pro Mini MCU board, memory card module, real-time clock module DS3234, and various sensors attached to nodes (Temperature and relative humidity, Light intensity, CO^2 concentration). Logging data in the local storage approach is advantageous where other communication protocols are not possible to use. Although such a solution is least desirable due to low data accessibility, data must be manually extracted from memory cards used in the project. The architecture is illustrated in Figure 2.19.

Another handy yet straightforward solution to acquire data from the Arduino MCU board to SBC or PC is using Universal Serial Bus. USB connection with Arduino MCU board has multiple purposes. It can be used for serial communication for acquiring the data from Arduino, program Arduino board using dedicated Integrated Development Environment (IDE) that is used to program the board using Arduino language that is resembling C programming language. It also allows communication with the board by establishing serial terminal communication that can report various sensor readings and store the data in PC or other SBCs [80]. To communicate with the computer, Arduino boards use integrated "USB to UART bridge" based on ATmega16U2 chip to be able to communicate with PC COM ports [81], illustrated in 2.20. Some Arduino boards do not have an integrated USB-UART bridge. In that case, an external bridge is used to interface the Arduino board with the computer or SBC. For example,

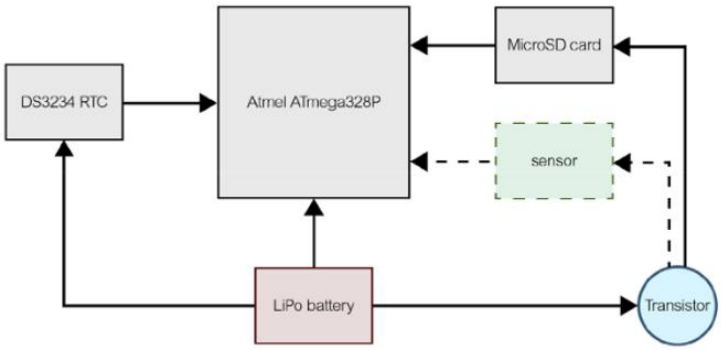


Figure 2.19: Indoor long-tern environment data collection node design.

Arduino Pro Mini ATmega328 needs to have an external USB-UART interface (based on FT232 chip) as illustrated in Figure 2.21.

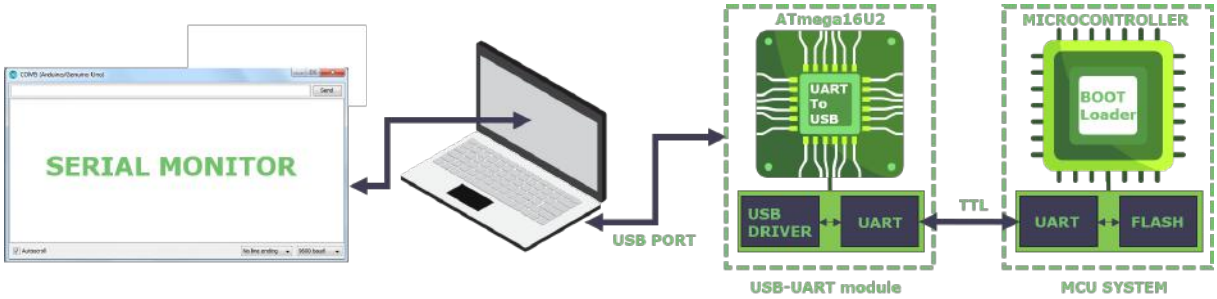


Figure 2.20: PC and MCU board communication via USB-UART module.

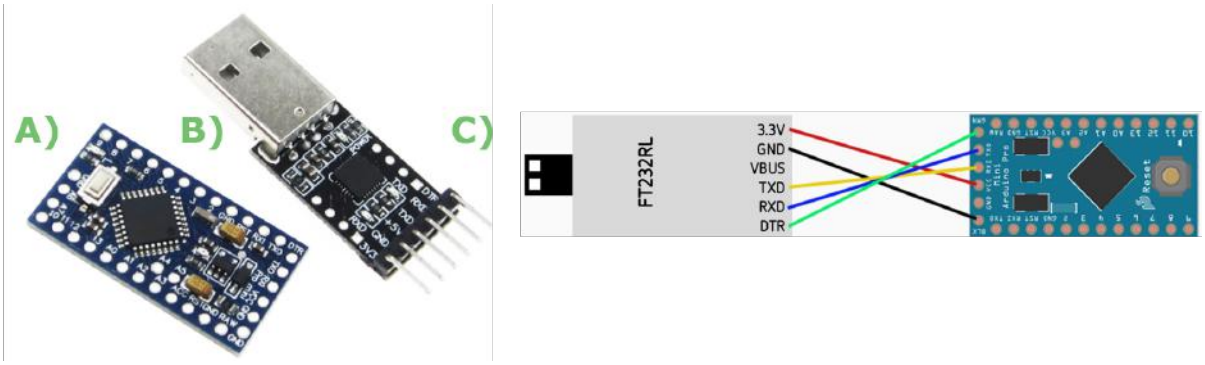


Figure 2.21: a) ATmega328 Pro Mini Borad, b) external USB-UART bridge, c) FT232 bridge interfacing with Arduino Pro Mini ATmega328 board.

Other communication types that Arduino supports are UART, I^2C / Two Wire Connection (TWI) and serial peripherals interface (SPI) [45]. In Table 2 comparison of previously mentioned communication protocols are shown.

UART is a hardware device or a circuit that is used for serial communication between two devices. E.g., two Arduino boards can communicate using UART, or Arduino and Raspberry Pi, as illustrated in Figure 2.22, because it also supports UART. It is worth mentioning that exists different standards of

UART: RS232, RS423, RS422, and RS485. It works by converting data into data packets that are sent sequentially one bit at a time. The sender converts data bytes into bits and converts them into packets that contain a start bit, data frame, parity bit, and stop bit. UART connection has two wires RX and TX. TX pin is for transmitting the data, and RX pin is for receiving it. The receiving device checks for the packet errors by counting the number of 1's and comparing the value with the parity bit that was included in the packet. A parity bit can be 0 or 1 depending on 1's in the data packet. If the number of 1's is odd in the data packet, then the parity bit is 1. If the number is even, then parity bit is 0 [82, 83].

If a packet was received without any errors, the receiver will strip start bits, parity bits, and stop bits, leaving only the data frame. It might receive several previously mentioned packets until it is able to reconstruct the whole data byte. After rebuilding the byte, it is stored in the UART buffer. Unlike SPI and I2C communication protocols that are based on a master-slave approach, UART is multi-master, meaning that any party can initiate packet transition without being explicitly asked [82, 83].

The speed of transmission is dependent on the baud rate that is bits per second (bps). For proper communication, both devices must be with the same set speed. Common values for such rate are 9600, 1200, 2400, 4800, 19200, 38400, 57600, and 115200 bps [82, 83]. The advantages of such a communication protocol are good support, and clock speed is not deeded, vigorous to errors. Disadvantages: limited size of bits [84].

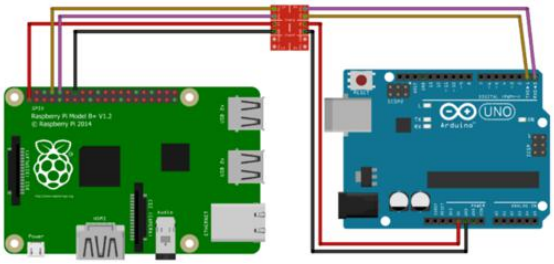


Figure 2.22: Interfacing Arduino and Raspberry Pi wired communication via UART.

I2C also referred to as TWI. This interface bus is a standard for interfacing MCUs and various peripherals together. It is is mostly used for master-slave communication via Serial Data (SDA) and Serial Clock (SLC) wires. SDA provides messages with contained structured data, and SLC guarantees clock synchronization. Moreover, there are several types of bus speed standard mode at 100kb/s, fast mode at 400kb/s, high-speed mode 3.4Mb/s. Many sensors, including MEMS accelerometer, are communicating via this bus [85]. I2C is capable of connecting multiple slave devices to a master device. The protocol defines a unique 7-bit slave address. By the slave address, these devices are identified in the bus. Figure 2.23 illustrates the diagram of two connected devices via an I2C bus. Some of the advantages include low power, ease of use, being widely supported between different boards, automatically configured. Disadvantages are: unsupported long-distance (hard wired cable cannot be too long, max 10 meters) communications due to the origin of the protocol, unsupported high-speed connections, a limited number of nodes [84]. In I2C connection, a master (microcontroller) or a slave (peripheral) can act as digital input or digital output it is called *tri-state*. In this state, lines are neither LOW of HIGH, instead, it is a floating value. The output is also called open-collector, meaning that lines require a pull-up resistor. Usually, it is a 4.7 kΩ resistor pulling up 5V or 3.3V, sometimes a line is up with 5V, and the peripheral is operating on 3.3V. In this case, a voltage level converter has to be used [86].

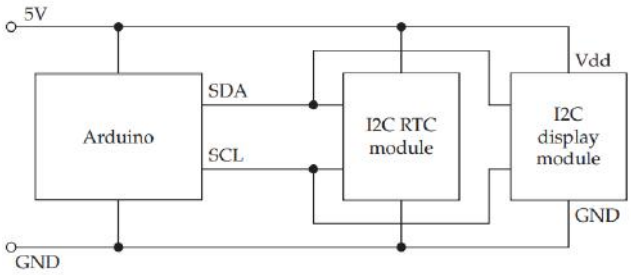


Figure 2.23: Interfacing multiple devices under the same I2C bus.

Master defines clock frequency on SCL line when the data has to be sent, master or slave uses an SDA line out of tri-state and sneds the data as logic highs or lows in other words 1 or 0 in respect of clock signal. After the transmission is completed, the SDA pin is returned to its floating tri-state. Using this principle bus can communicate both ways [86].

SPI it is yet another serial bus standard that enables to interface peripherals to Arduino or any other microcontroller that supports SPI communications. It is fast but requires more connection pins compared with only two that I2C is requiring. SPI communication lines consist of Pin Chip Select (CS)/Slave Select (SS). Serial Clock (SCK), Master Out Slave In (MOSI), and Master In Slave Out (MISO). SPI is capable of communicating in full-duplex mode. SPI principle is based on Master and Slave communication. Master peripheral is the one to dictate the clock frequency rate in SCK line for synchronized communication. CS and SS lines are for slave state determination which can be active or inactive. If the slave peripheral is active, the master node is able to start data exchange. CS line can also be used to select the address of a slave, which will exchange the data with the master, meaning multiple slaves can be connected. [87]. Each peripheral connected to the microcontroller must have a unique SS connection. This approach addresses the desired peripheral by turning all other peripherals off. A broad spectrum of devices can be connected to a microcontroller using SPI. Some peripherals can have both SPI and I2C communication interfaces. A diagram is presented in Figure 2.24 which has two slave devices connected to the Arduino master node, D2 and D3 are used for SS ports, SCLK, MOSI,MISO ports depend on the microcontroller model, and they have to be checked in appropriate board documentation. It is worth mentioning that SPI setup does not require pulling up resistors [88].

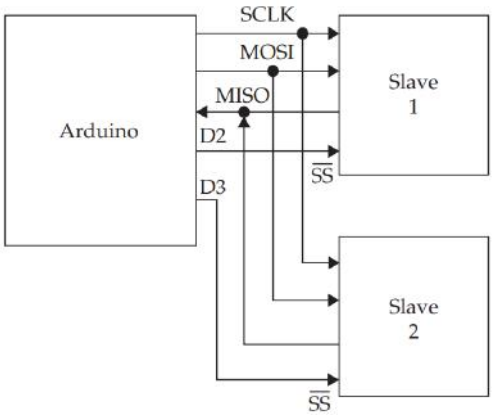


Figure 2.24: Arduino interfaced with two slaves via SPI connection.

1-Wire it is another bus standard. It was designed to complete a similar purpose as the I^2C bus,

meaning communication between peripheral IC with the least number of data transfer lines. 1-Wire communication is slower in comparison with I^2C bus. 1-Wire protocol has a feature called *parasitic power*, it grants peripheral to be connected to a microcontroller with just two wires: ground (GND), and combined power and data wire. In comparison with SPI and I^2C communication buses, 1-Wire does not have that many potential devices to exploit this feature. The most popular sensor available is Dallas semiconductor DS18B20. 1-Wire bus supports multiple devices. Each device has a unique ID. It is possible to chain a maximum of 255 devices using a single bus. 1-Wire device connection to Arduino board is shown in Figure 2.25 [89]. When a system has only one slave device on the bus, it is called *single drop*, when multiple devices are connected, it is referred to *multi-drop* system. 1-Wire network uses tight control, meaning no slave allow to speak without getting a request from the master node. Communication starts when the master sends a reset pulse, slave replies with a presence pulse. If the presents pulse replies to reset pulse, it means that the bus is ready to operate on 1-wire. To send a reset pulse, the master puts the bus line low for 480us ($480x10^{-6}$ seconds). After that, the master switch into receive mode. Slave then pulls a presents pulse for 60us to 240us. After this master sends a search ROM command (F0h) on 1-wire bus. Master identifies slave addresses by process of elimination. To find one slave master needs around 13.16ms, meaning it can identify around 75 different slaves per second. If it is a temperature sensor, the master sends a temperature conversion command to slaves. After receiving the command, slaves start a temperature conversion and saving this data in a specific register. In *multi-drop* mode, the master sends the command to read the temperature followed by the sensor address. If the slave's address matched, the request slave is responding with the data while other sensors are yielding. This process repeats till data is received from all slaves connected to the bus [90].

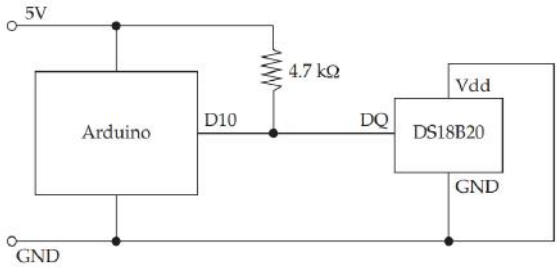


Figure 2.25: 1-Wire device interface with arduino using *parasitic power*.

2.4 Digital signal processing

The reason we are reviewing Digital Signal Processing (DSP) techniques is that this work includes an accelerometer that will produce raw time-series data, which is also referred to as a *signal*, is innately expressed in the time-domain. An intuitive representation of temporal relations between data points can be provided by exploiting time-domain signal processing.

When working with sensors, we must take into consideration noise. Noise is present in almost all readings provided by the sensors. There are different types of noises, e.g., environmental noise like temperature variations, undesirable movements, or noise from hardware like electrical noise or mechanical, thermal noise. Due to these signal imperfections, we need to use signal processing to clean the signal and improve its quality.

An intuitive representation of temporal relations between data points can be provided by exploiting time-domain signal processing. When the data points repeat over regular or semi-regular intervals, a

transformation can be used to convert time-domain information into frequency-domain to extract more meaningful information about the oscillatory frequencies hidden in time series.

This section will be divided into two parts, signal denoising, where we review some available methods to reduce noise to signal ratio. Next, we reviewed fundamental time-domain procedures and show how oscillation data in the time-domain can get a different form of representation, mainly frequency-domain.

2.4.1 Signal denoising approaches

In this sub-section, we will review some denoising methods for one dimension time-domain signals. Moving average filter, weighted moving average filter, median filter, Simple exponential smoothing filter, Fourier Coefficient Suppression filtering, Wavelet Shrinkage filtering were reviewed.

Moving average filter

Moving average (MA) filter is a simple low pass Finite Impulse Response (FIR) filter commonly used for smoothing the series of the sampled signal data. Moving average is a ubiquitous technique in DSP because it is fast and can remove random noise. MA is used for tasks such as random noise reduction while preserving sharp step response. In literature, it is common to see a low-pass FIR filter refer to MA. As the filter's name suggests, it filters by averaging several points from the input signal to produce each point in the output signal. Temporal statistics are computed using an N sample of the data points along the time axis, and the samples in a temporal moving window are averaged to produce the output at various points of time [91]. Equation 2.1 shows the MA approach, where each acceleration point $a(t)$ at instant t with an average of the point in the vicinity of t where N is a span and Δt is the time step. As an example, when N = 125, the 100th sample will be equal to averaged samples from 38th point to the 162nd point [92]. N is the number of points that will be averaged. User can define the N depending on the problem. Example of MA is illustrated in Figure 2.26, a) an original signal a, b) $\overline{a}$, when N = 4, c) $\overline{a}$, when N = 8, d) $\overline{a}$, when N = 16 [93].

$$\overline{a}(t) = \frac{1}{N} \sum_{j=(N-1)/2}^{j=-(N-1)/2} a[t + j\Delta t] \tag{2.1}$$

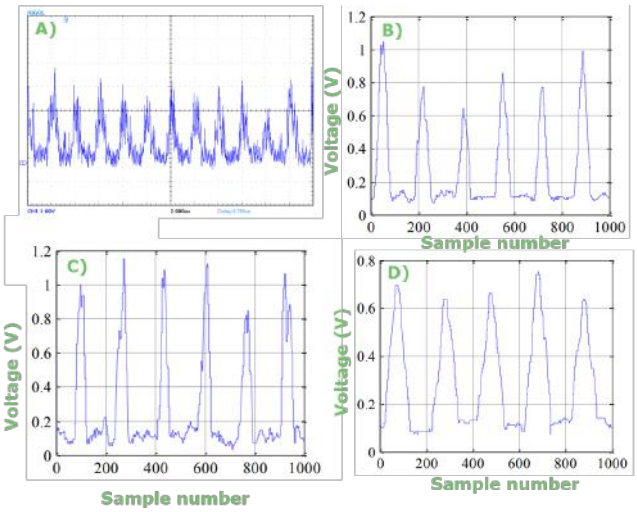


Figure 2.26: a) original signal, b) MA filtered signal, when N = 4, c) when N = 8, d) when N = 16.

Weighted moving average filter Weighted moving average or WMA is a slightly upgraded version

of a MA. In MA, every single data point is equally important. However, more current values tend to reflect better the state of the process at time t. When filtering, it is more useful to emphasize more recent data in signal segmentation. The weighted smoothed average assigns a greater weight to the more recent data. WMA works by assigning the weight to more current data than the older values. WMA assigns a weighting factor to each value in the data series according to its age. The most recent data gets the greatest weight, and each older value gets a smaller weight. It is useful to acquire a smooth curve for better trend classification. It reveals long-term trends. WMA is expressed in Equation 2.2, where x(n) is the original signal data point, and $\overline{x}(n)$ is an output [94]. An example of filtering with WMA is demonstrated in Figure 2.27.

$$\overline{x}(n) = \frac{x(n)\sum_{n=1}^{N}((1-0.05*i)*[x(n-i)+x(n+i)])}{1+\sum_{n=1}^{N}((1-0.05*i))} \tag{2.2}$$

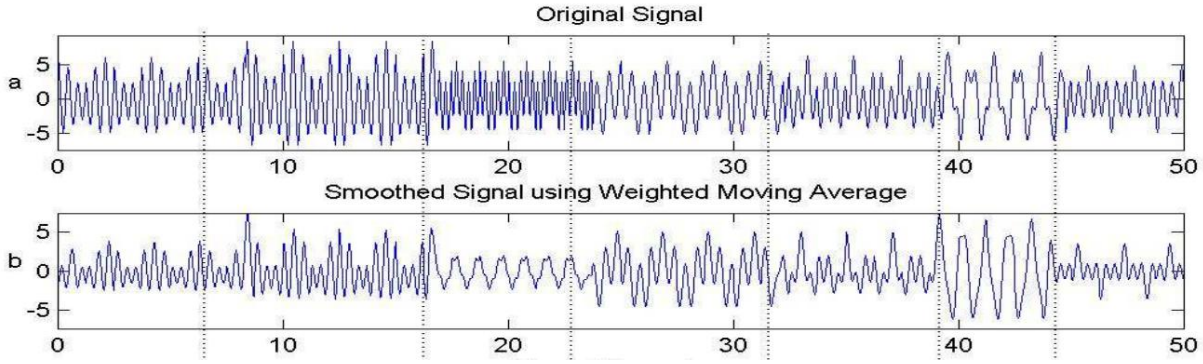


Figure 2.27: a) original signal, b) signal after WMA filtering.

Median filter Median filter, or MF, is considered to be a nonlinear digital filtering approach, often used to remove random noise from signals. It is a nonlinear local filter, and its product is a value in the central element of a sorted array of amplitude values from the filter window. (It replaces the value with the median of those values). Similar to the moving average filter, instead of averaging the neighboring values, it takes a median of those neighbors. The principle is the same as in MA, where filter window size (N) is defined by the user, sliding through the signal. It is worth mentioning that MF truncates the high peaks of an input signal. Example of MF is represented in 2.3 when window size is 11 [95].

$$\overline{x}(n) = median[x(n-5), x(n-4), ...x(n), x(n+4), x(n+5),] \tag{2.3}$$

An example is represented in Figure 2.28, where a) is an original sinusoidal signal of 1kHz frequency contaminated with 15kHz frequency signal, b) is filtered signal using MF.

Simple exponential smoothing

Simple exponential smoothing (SES) is a popular form of signal smoothing. It is known to be a recursive system with infinite impulse response (IIR). It is an effective time series technique used for data prepossessing to eliminate various abnormal trends like noise in the raw data signal. Exponential smoothing is an exponential weighted moving average (EWMA) approach where different weights are given to the recent and previous measurements. The weight depends on the magnitude of the control parameter (smoothing constant) α, causing the forecast values or the filter output to change accordingly. There are two commonly used forms of SES. They are expressed in Equation 2.4 and Equation 2.5 [96]:

$$\overline{x}(n) = \alpha x(n) + (1-\alpha)\overline{x}(n-1) \quad (2.4) \qquad \overline{x}(n) = \alpha x(n-1) + (1-\alpha)\overline{x}(n-1) \quad (2.5)$$

where $\overline{x}(n)$ output and $x(n)$ is an input at the *n*th time instant. Equation 2.4 represents a filter, whilst the 2.5 can be used a filter and a forecast [96]. An example is represented in Figure 2.29, where a) an original signal of 1kHz is contaminated with 15kHz noise, b) filtered signal using SEM, when α = 0.15.

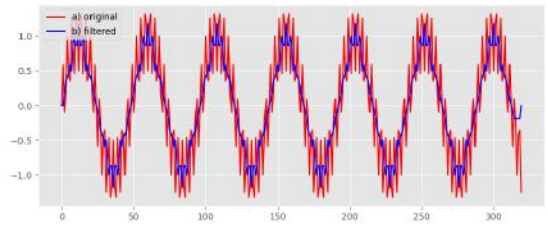


Figure 2.28: a) red line original signal, b) blue line signal after FA filtering.

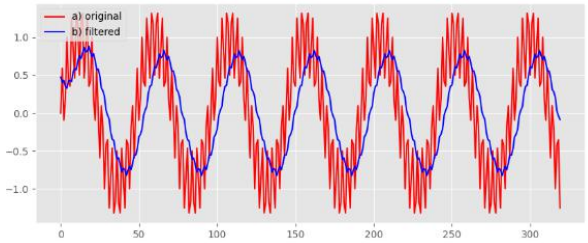


Figure 2.29: a) red line original signal, b) blue line signal after SEM filtering.

Fourier Coefficient Suppression

Fourier Coefficient Suppression (FCS). The Fourier filtering method is based on the Fourier transform, which decomposes a signal into its frequency components. It disassembles the periodic signal into its sinusoidal components, suppresses the high-frequency components and achieves the denoising of a signal. It is called low-pass filtering. The name of the calculation of the Fourier series coefficients is the Discrete Fourier Series (DFS) algorithm. In the section on signal feature extraction, we will be talking about it in more detail. DFS transforms a finite number of coefficients. The transform evaluates only a finite number of coefficients, the total being equal to the original number of data points in one period of the original signal. Because of this, each spectral line is regarded as the k^th harmonic of the basic period in the data. DFS calculated the coefficients of given data points using Equation 2.6 and Equation 2.7 [97]:

$$A_n = \frac{2}{N}\sum_{k=1}^{N} e(k)[sin(\frac{2\pi nk}{N})] \quad (2.6) \qquad B_n = \frac{2}{N}\sum_{k=1}^{N} e(k)[cos(\frac{2\pi nk}{N})] \quad (2.7)$$

Where e(k) is a discreet time signal and k is the discreet time index and N is the N^O of samples to be processes, n is a coefficient index such that 1≤n≤N. To recover original signal from its coefficients, we can see in Equation 2.8 [97].

$$e(k) = \sum_{k=1}^{N} A_n[cos(\frac{2\pi nk}{N})] + B_n[sin(\frac{2\pi nk}{N})] \quad (2.8)$$

For frequency component identification, the scale of the coefficient is resolved by knowing the sampling frequency, meaning that coefficient index n=1 will correspond to a frequency of f_s/N where f_s is the sampling frequency of the data and N is the number of points. To remove undesired frequencies, a corresponding frequency must be eliminated on both sides of the Nyquist frequencies. Moreover, using Equation 2.8, the signal will be reconstructed back to its original shape but with removed noise or undesired frequencies [97]. DSF denoising pipeline illustrated in Figure 2.30

Wavelet Shrinkage Demonising The wavelet Shrinkage approach is a more complex algorithm compared with the previously mentioned approaches. Wavelet Shrinkage (WS) is based on the Discrete Wavelet Transform (DWT) algorithm. Wavelet transform is a variation of an approach called *template matching*, where small portions of a predefined signal template are sliding through the raw signal

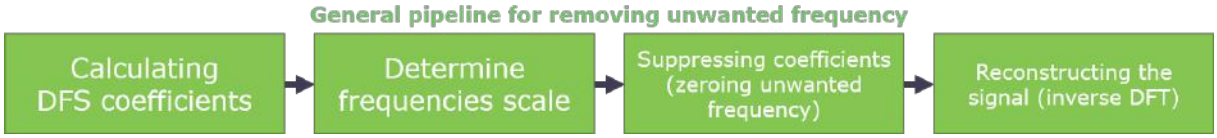


Figure 2.30: Pipeline of denoising signal using DFS.

performing correlation of the matched template with new signal in the time series. Filter output will have high output if the template is correlated and low output if the segments differ from the predefined segment template. As illustrated in Figure 2.31, top signal clean electrocardiography (ECG), middle ECG signal is contaminated with noise, and the little signal in red is the previously mentioned predefined pattern sliding through the raw signal data, the last signal is an output of a "high" signal when the pattern matched the time series raw signal [98].

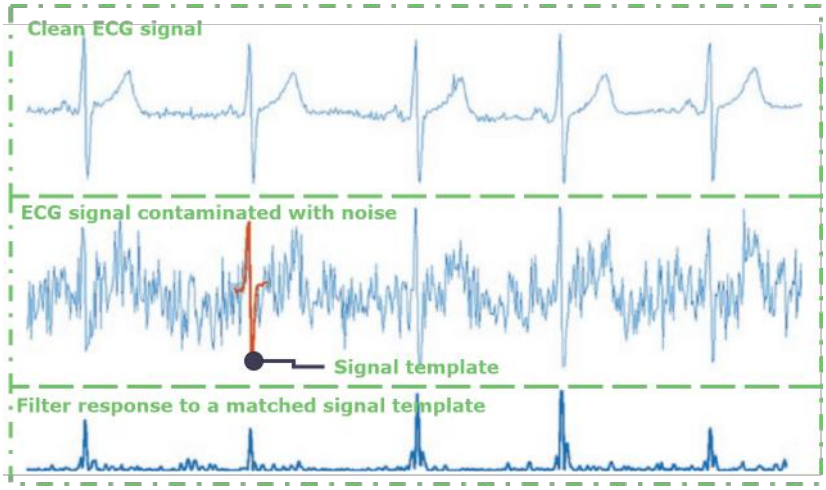


Figure 2.31: Pattern matching example

A mathematical function that is disintegrating an original signal into different signal components is called wavelets. Wavelet transform (WT) basis functions are scaled depending on the frequency. There are different types of small waves that are commonly known as Mother (pattern signal) wavelet, and it is used for the implementation of WT. Different types of functions are using these mother wavelets to perform the decomposition of the signal x(t), and from there, a weighted set of scaled wavelet functions are generated y(t) [99].

Different types of mother wavelets can be used depending on signal types and processing requirements, e.g., Daubechies, Haar, Symlet, Coiflet, Mexican Hat, Morlet [99]. The selection of a mother wavelet is important because obtained results will be affected by the selected type of wavelet base. Furthermore, it is necessary to find out the best correlation between received signals and mother wavelet [100]. Recent research on wavelet application analysis has concluded that denoising, feature extraction, time-frequency analysis, parameter identification, and prediction claims that Daubechies, Symlet, Coiflet, Morlet wavelets are appropriate choices [101]. The main idea is that WT is breaking the signal into different frequency levels of coefficients. Analyzing signals in different frequency bands and scales can help determine the signal behavior. Furthermore, it helps detecting corrupted signals, and eliminating them becomes the least complicated process [99, 102]. There are two defined WTs: *Continuous Wavelet Transform* (CWT) and *Discrete Wavelet Transform* (DWT). CWT is a computationally heavy task. Due to this reason, a

DWT was introduced. It uses scale and position values based on powers dyadic dilation and translation, and it is mathematically expressed in Equation 2.9 [102].

$$DWT(m,n) = \int_{+\infty}^{-\infty} x(t)_{\varphi^*{}_{m,n}}(t)dt \tag{2.9}$$

where

$$\varphi^*{}_{m,n}(t) = a_0^{-m/2}(\frac{(t - na_0^m b_0)}{a_0^m}) \tag{2.10}$$

Where m is frequency localization, and n is time localization. Generally, a0 can be set to 2 and b0 to 1, and this gives us dyadic orthogonal wavelet transform and sets the basis of Multi-Resolution analysis (MRA) [102]. To eliminated unwanted signals, the technique called thresholding (TH) is being used. There are two types of threshold: Soft TH and Hard TH. Hard TH is when values are set to zero when the absolute value of the signal bit is below the threshold. Hard TH function W_{ht} (Equation 2.11) is more sensitive to small changes in the signal and has a higher variance. Soft TH function W_{st} (Equation 2.12) is more stable than hard TH. Nevertheless, computationally, it is more costly in comparison with the Hard TH [102, 103].

$$W_{ht} = \begin{cases} w, & |w| \geq t \\ 0, & |w| < t \end{cases} \tag{2.11}$$

$$W_{st} = \begin{cases} [sign(w)](|w| - t), & |w| \geq t \\ 0, & |w| < t \end{cases} \tag{2.12}$$

The pipeline to eliminate the noise can the used as illustrated in Figure 2.32 using DWT and TH.

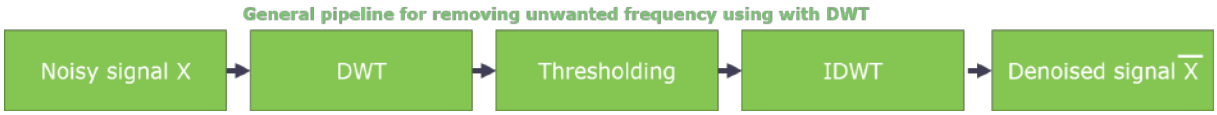


Figure 2.32: Signal denoising pipeline using DWT algorithm and TH.

2.4.2 Feature extraction

Signal magnitude features and time window averaging We can perform essential integration of the feature extraction called peak-picking. It simply determines the maximum and the minimum values across data points in the specified time interval. Usually it is captured in the time of occurrence as the feature of that time segment. We also can extract yet another feature using window averaging. Time series can be averaged over all parts of the time interval to create a feature for that time segment. The averaging approach is sometimes preferable compared to peak-picking because the signal might have some noise that can have transient magnitude peaks, which would corrupt results by using peak-picking. If multiple observations of a noisy time series signal are available, they can be time-aligned and averaged across observations, meaning if we have multiple sensors recording the same event, the synchronized data points can be averaged to create a new signal as illustrated in Figure 2.33 [98].

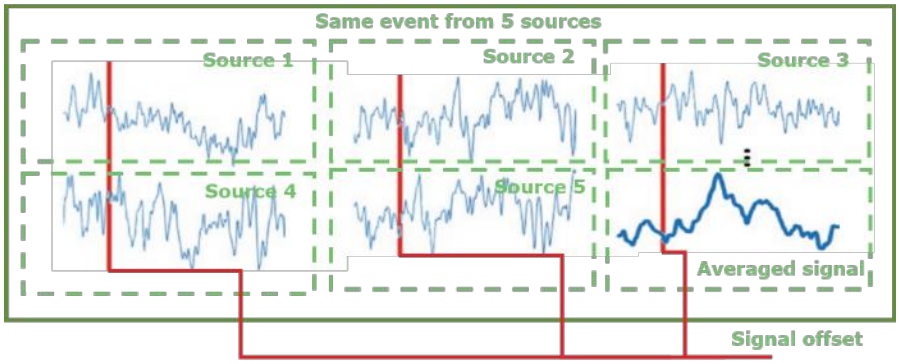


Figure 2.33: Averaging synchronised signal from multiple sources.

Fourier transform Fourier transform (FT) is a widely used technique for frequency analysis that converts a time-domain signal into the frequency domain. The selected input signal for the analysis can be described as a sum of essential sinusoids of various frequencies. There are few variations of FT such as *discrete Fourier transform* (DFT), **Fast Fourier Transform (FFT)** and *Short Time Fourier Transform* (STFT). They are generally used for power quality disturbance recognition. Based on the output of the transform, FFT is similar to DFT but requires less time due to its optimization and reduction of computational complexity. STFT is another FT variant that transforms the waveform into small stationary slices. It is used to get the frequency/phase signal information, including the time domain. We are interested in FFT because it is an efficient and effective algorithm that can be used to extract features from the signal, such as dominating frequencies which are not possible to know just by looking at time-domain-based signals. N frequency representatives uniformly spaced over a frequency range of sampling rate/2 is a product of the N-sample time series after the FFT is applied to a signal. Due to one-to-one transformation, there is no loss of information, and the signal can be reconstructed from the frequency to the time domain. The maximum frequency of sampling rate divided by two in this transformation is called Nyquist frequency and refers to the greatest frequency that can be reconstructed using the FFT, meaning that if we sample with, for example, 50Hz sampling rate, the maximum highest frequency can be 25Hz after the transformation to the frequency domain. Another way to call frequency domain samples is frequency bins which are a pretty common expression in DSP literature regarding FFT. Each sinusoidal amplitude of the signal is being tracked by each of the frequency bins of the FFT magnitude spectrum. Complex values are being produced by the FFT that is later converted to the phase and magnitude. The FFT spectrum of an actual signal has symmetry such that only half of the bins are unique, from zero to + sampling rate divided by two. The bins from zero to sampling rate divided in half is a mirrored image of the positive bins (i.e., zero frequency). Therefore, for an N-sample actual signal, there are N/2 unique frequency bins from zero to sampling rate divided by two. This knowledge allows us to use FFT and know how to interpret the result provided by the algorithm without deep-diving into the algorithm's mathematics related to the association of negative frequencies. Finer FFT results can be achieved by appending M zeros to the N-sample signal, producing (M + N)/ 2 bins from zero to the sampling rate/2. This is known as zero paddings. Zero padding does not increase spectral resolution since no additional signal information is included in the computation, but it provides an interpolated spectrum with different bin frequencies. [98].

The example is illustrated in Figure 2.34 where we can see the sample of time-domain vibrations wherein (A) we can see a signal of 1000Hz and (B) we can clearly see a spike at 1000Hz in frequency domain analysis, in (C) signal is being received containing 1000Hz and 3000Hz frequency components,

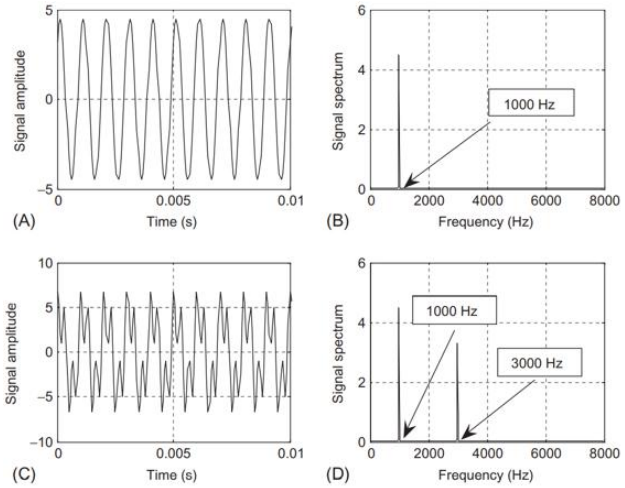


Figure 2.34: Extracting frequencies from a time series signal with FFT

and (D) indicates this information in signal spectrum. To perform FFT analysis, the sampling rate and the number of samples have to be decided. In vibration analysis, short time intervals (< 3 seconds) are preferred for accuracy and consistency. FTT size sample number is evaluated in powers of two (e.g., 1,2,4,8,16 . . . n). The number of bins (number of samples in the frequency domain) is defined by FFT size [104].

2.5 Appliance recognition using machine learning

One crucial part of the work done in this thesis is related to appliance classification. It will be used to assess energy consumption, pattern recognition, and user profiling. This will be used to detect faults and create a recommendation service that will help users define better settings for a specific appliance, minimize energy consumption, and improve efficiency.

In work done by Zufferey, D. et al. [105] an ILM approach was applied using the plogg - smart energy meter plug, acquisition device that was capable of capturing a four-dimensional vector of electrical parameters including power (W), reactive power (var), current (A) and phase of voltage relative to current (ϕ). Five classes of appliances were investigated, including laptop, fridge, computer and monitor, phone charger, and coffee machine. For each class, six different models of each class were recorded for a total of 30 acquisitions. The sensor was programmed to send an acquisition snapshot with a sampling frequency of 10^{-1} Hz (one data acquisition taken each 10 seconds). For data classification, two algorithms were used, namely K-nearest neighbour (KNN) and Gaussian Mixture Model (GMM). In the experiment 85% of accuracy was reached for both algorithms.

A similar approach was used by Reinhardt, A. et al. in [106], where power measurement and actuation unit (MAUs) were placed in-between power sockets and electrical appliances. MAUs provided power consumption information in the time domain of any electrical device connected to it to extract fine-grained information. Four different feature bases were generated to supply classifier algorithms considering temporal behavior, power consumption, the shape of consumption, and level of noise. Thirty-one different device types were recorded in home and office environments, and nine classifier algorithms Bagging, Bayesian Network, J48, LogitBoost, Naive Bayes, Random Committee, Random Forest, Random Tree, were used. The accuracy results were fluctuating from 84.13% up to 95.5%. The

Table 2.2: Compiled classification algorithm usage based on revision of related work.

Algorithm	Average accuracy	Citation
K-Nearest Neighbour	87.48%	[105] [108] [110] [111] [112]
Random Forest (RF)	94.1%	[106][110][112][113]
Support-vector machine	94.47%	[113] [107][111]
Bagging	94.2%	[106] [110][113]
Gaussian Mixture Models	89.86%	[105] [108] [114]
Naive Bayes	81.1%	[106] [111]
Logit Boost	94.32%	[106] [113]

highest accuracy of the experiment was 95.5% reached by Random Committee.

In [107], a research of real-time device identification of 14 devices data were recorded using a smart plug. Four electrical features were recorded: Active Power, Reactive Power, Vrms, Irms, and phase shift. The authors chose the Support Vector Machines, Artificial Neural Network, K-Means, Silhouette, Mean-Shift classification algorithms for the classification propose. According to this publication, Silhouette and K-Means reached 98% accuracy, and the lowest accuracy has been recorded by the Mean Shift algorithm 94%.

Authors of [108] were experimenting with quite well known appliance energy consumption pattern database called ACS-F1 [109]. This collection contains ten devices and hastens devices per category, capturing 100 different device consumption patterns. The database contains features such as real power (W), reactive power (var), root means square (RMS) current (A), frequency (Hz), RMS voltage (V), phase of voltage relative to current (ϕ). Two classification algorithms together with ACS-F1 database, KNN and Gaussian Mixture Model (GMM). The authors applied the dynamic coefficients algorithm as a preprocessing data approach. The classification was applied by device types. Classification of KNN has reached 88% accuracy, and including dynamic coefficients features, accuracy increased up to 90%. GMM had a slightly better accuracy of 93.6%, including prepossessing of dynamic coefficients. The Paper does not mention the accuracy of GMM without prepossessing. For completeness of this section, we created a Table 2.2 mentioning machine learning algorithms used for appliance classification and references. The table includes the average accuracy generated based on multiple sources.

In our work we experiment with KNN as our base line, Support-vector machine (SVM), Random Forest (RF) and Extreme Gradient Boosting (XGBoost). To be familiar with the algorithms in the following sections, we talk about these algorithms.

2.5.1 K-Nearest Neighbour

KNN is a supervised machine learning algorithm that can be used for classification and regression problems, which is a commonly used distance-based classification algorithm out there. An entity is being classified by counting the nearest neighbors. The majority of neighbors, based on the entity's point, will decide its label. The distance is computed by using different metrics: Euclidean, Chebychev, Minkowski, Cosine, Jaccard, Hamming. When performing this algorithm, an essential hyper-parameter is the value k. It defines how many neighbors are taken into consideration while defining the class label. We will be using this approach, setting a base for more complex classification algorithms. In our experiments, we are using a KNN classifier from the Python Scikit-learn package. For selecting the best number of neighbors, we ran KNN N times, increasing the value k(1..N) and capturing the results.

2.5.2 Support-vector machine

SVM is an ML model that is capable of performing linear, nonlinear classification, regression, and outlier detection [115]. A mathematical model, expressed as: **y = wx‘ +** γ , is being manipulated to allow a division of linear domain. If the data domain of two classes can be divided linearly (e.g., hyperplane or simply straight line) in the original domain, it is a linear SVM. If present data is not linearly separable, it is transformed into a feature space where the data domain can be linearly separated. In nonlinear SVM, kernel functions are being used to separate the classes inside the hyperplane. A non-linear SVM adopts a non-linear equation **y = w**ϕ **(x‘) +** γ,. SVM can be used with linear, polynomial, radial, Sigmoid, Gaussian, Bessel, ANOVA kernels [116].

2.5.3 Random forest

Random forest (RF) based on large number T of classification and regression trees also known as decision trees (DTs). T is one of the most important hyper-parameters that is being set by the user. In a setting of RF, many DTs are generated using randomly selected training subsets of predictor variables for modeling the outcome. When predicting the label, each tree provides its prediction, and the class label is defined by the vote of a majority of DTs. Due to this architecture, RF often provides higher accuracy rates in comparison with a single DT [117].

2.5.4 Extreme Gradient Boosting

XGBoost is a short for Extreme gradient boosting. To understand the idea behind XGBoost first, we need to understand how gradient boosting works. Functions cannot accurately describe the data. It can only be an approximation of the data distribution. These functions might introduce some errors: $y_i = F_1(X_i) + error_{1i}$, where y_i is the outcome variable X_i is a vector of predictors. If we assume that $F_1(X_i)$ function is a weak learner, the X, y relationship is strong enough. The existing error might not be white noise and might correlate with y that is not explicit. A second model can be trained considering the error term: $y_i - F_1(X_i) = error_{1i} = h_1(X_i) + error_{2i}$. Now the updated model would look like $F_2(X_i) = F_1(X_i) + h_1(X_i)$ and these steps might run iteratively N times, where model fitting can be performed at the N^{th} step: $F_N(X_i) = F_{N-1}(X_i) + h_{N-1}(X_i)$. Gradient boosting can be used with a weak learner with a functional form such as Generalized Linear Models (GLM), decision trees, or neural networks. Although, in practice, it is common to focus on tree-based learner boosting [118]. In ML terminology previously mentioned "error" is called a loss function $L[Y, F_N(X)]$. extreme Gradient Boosting or XGBoost is based on a decision tree implementation with gradient boost. To reduce overfitting and increase execution speed, a randomization function is being implemented, which are random subsamples to rain individual trees and column subsampling at tree and tree node levels [119]. We have used the python XGBoost library that implements machine learning algorithms under the gradient boosting machines framework in our work. Moreover, it can support objective functions including classification, ranking, and regression [120]. XGBoost contains multiple hyper-parameters, which need to be tuned for optimal classification. It is nearly impossible to tune it "by hand," for automated calibration, we have used the grid search approach to find the best combination of hyper-parameters.

Chapter 3

Appliance monitoring and classification approach

3.1 Requirements

There is a variety of different appliances that we use in day-to-day situations. Our goal is to create a low-cost prototype that would capture appliance energy consumption and record the utilization trends to create an energy performance assessment. To reach it, we need to create a system that would capture various measurements, like energy consumption, including current, energy, power, voltage. We are planning to recognize the device type and model using machine learning algorithms based on energy consumption patterns. Furthermore, we have appliances that consume other resources like water. Due to this reason, we are focusing on washing machines and dishwashers where our system would measure water consumption. Water can have different temperatures during the intake and exhaust of these appliances. Due to this reason, we measure the temperature of water intake, exhaust, and environmental temperature.

On top of that, washing machines produce oscillations due to electric motors spinning the tub part. Dishwashers, as well, have spinning upper and lower spray arms that might make machines oscillate. We are planning to measure vibrations that would create oscillatory patterns of such appliances. Furthermore, some appliances are multifunctional, like washing machines, meaning that they produce different consumption patterns based on the selected program. Oscillations intertwined with energy and water consumption measures should generate a specific pattern based on the chosen program. We aim to recognize the chosen program of an appliance based on the measured patterns.

We capture measurements with a frequency of 1Hz (1 measurement sample per second). Although the sampling frequency of vibrations has to be higher, we use 200Hz sampling frequency for oscillations. Table 3.1describes planned appliance measurements in this project. System prototypes must be highly scalable and new appliances effortlessly integrated into the monitoring platform - preferably wireless independent from each other node where each appliance has a dedicated monitoring node.

The following section goes through the system architecture and the necessary parts to reach our objectives and fills these requirements.

3.2 System architecture

The proposed system will collect the energy consumption of appliances and multifunctional appliance operational patterns during the program execution cycles, e.g., washing machines, dishwashers. The

Table 3.1: The measurements that are captured from appliances.

Derived measurement quantity or feature	**Name of Unit**	**Symbol of Unit**
Energy (all electrical appliances)		
Power	Watt	W
Energy consumption	Kilowatt-hour	kWh
Current	Ampere	I, i
Frequency	Hertz	Hz
Voltage	Volt	V
Power Factor	Power Factor	PF
Operational (washing machines, dishwashers)		
Water flow	L/s	liters per second
Water volume	Milliliter, liter	mL, L
Acceleration X,Y,Z	Acceleration	m/s^2
Acceleration (min, max, mean)	-	-
Temperature (of appliance water)	Degree Celsius	^oC
Environmental		
Temperature	Degree Celsius	^oC
Relative Humidity	-	expressed in %

system will be collecting energy features (power, energy consumption, current, frequency, voltage, power factor) using a current sensor. For the operational mode of specific appliances, a water flow sensor is used to collect water consumption. This information is relevant to determine the water consumption per operation cycle. Additionally, temperature sensors used for measuring water temperatures of machines that consume water, and accelerometer sensors to measure oscillations of appliances such as washing machines and dishwashers. Also, we measure environmental temperature and humidity. Temperature and humidity measurements are essential when monitoring devices such as refrigerators because they need to use more energy to maintain a constant temperature inside while being affected by higher temperature variations.

Figure 3.1 shows the architecture that we designed during this thesis. It comprises components that represent the sensor hardware, signal acquisition, signal processing, and appliance classification.

To read the sensor values, an MCU board is used to capture and compute all sensor measurements. Then, data is sent to a single-board computer that aggregates it. This single-board computer acts as an edge device from the IoT perspective because it does minimal data prepossessing before sending the data to the cloud. The proposed architecture, from an IoT perspective, is illustrated in Figure 3.2 an IoT perspective, is illustrated in the Figure 3.2.

Figure 3.3 illustrates how the appliances can be monitored. We separate the use of sensors in **(a)** appliance with multiple sensors, and **(b)** appliance with a single current sensor. We call appliances with multiple sensors complex appliances, e.g., washing machines and appliances that require a single sensor, a simple appliance. Some appliances such as washing machines, dishwashers require more sensors to extract operational patterns like in 3.3-a, where a washing machine can have a temperature, acceleration, current, and water sensors. In contrast, other appliances will use only the energy monitoring module as in 3.3-b.

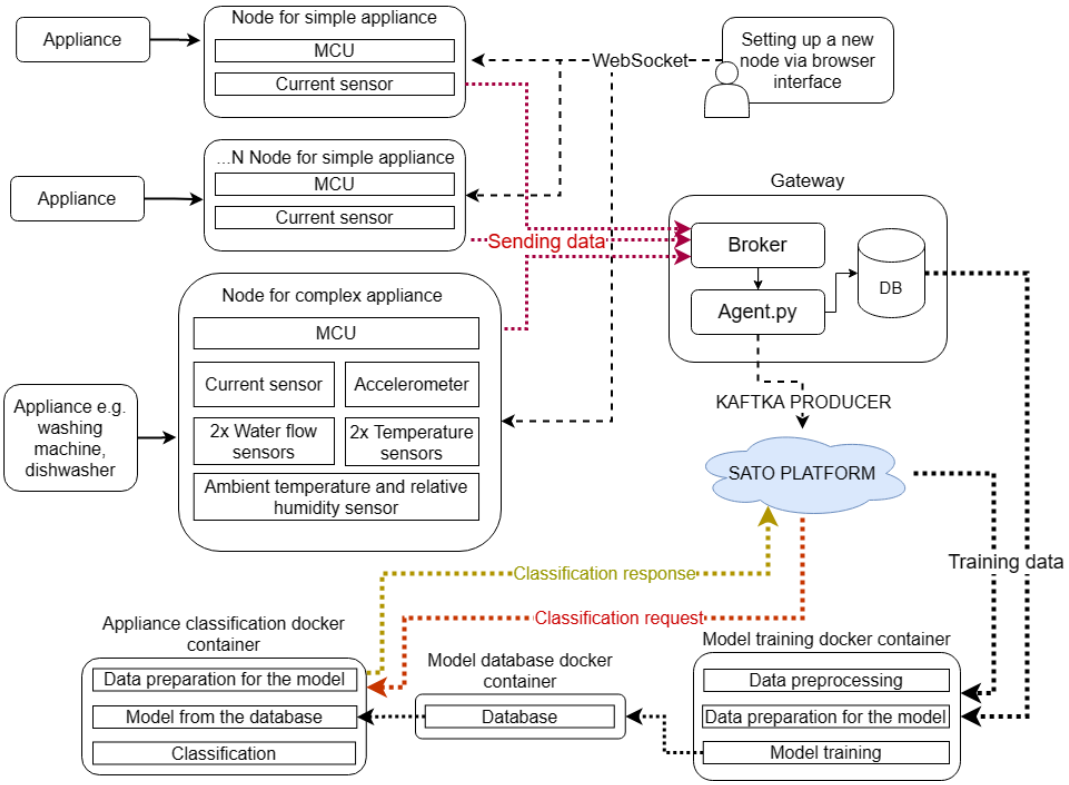


Figure 3.1: Proposed preliminary monitoring and appliance classification prototype architecture.

For appliance energy consumption and pattern collection, we plan to use the following sensors:

- **Current sensor**, which measures the power consumption. This sensor will be applied to all-electric appliances.
- **Water-flow sensor**, which measures the quantity of water consumed by an appliance. This sensor will be applied only for washing machines and dishwashers. Multiple sensors will be used. One for water intake, the other one for water drain.
- **Temperature sensor**, which is going to measure the temperature of drain and intake pipes of appliances that consume and exhaust water.
- **Multi-sensor** of temperature and relative humidity to constantly measure temperature and humidity of the ambient.
- **MEMS accelerometer**, which is going to measure vibrations of devices that might produce oscillations, such as washing machine, dishwasher, dryer.

In our proposal, we used an MCU that reads sensor signals and converts them into readable measurements. This data is transferred into a single-board computer (SBC). For MCU and SBC communication, we decided to use wired serial communication to record appliance consumption patterns. The SBC is acting as a gateway since it has an internet connection. After the data is collected and stored inside SBC, it is processed. In the upcoming section, we discuss the plan of data processing. The architecture also includes the appliance classification part, where we have dedicated containers for model training and storage. A dedicated container is responsible for the appliance classification task. This container will be

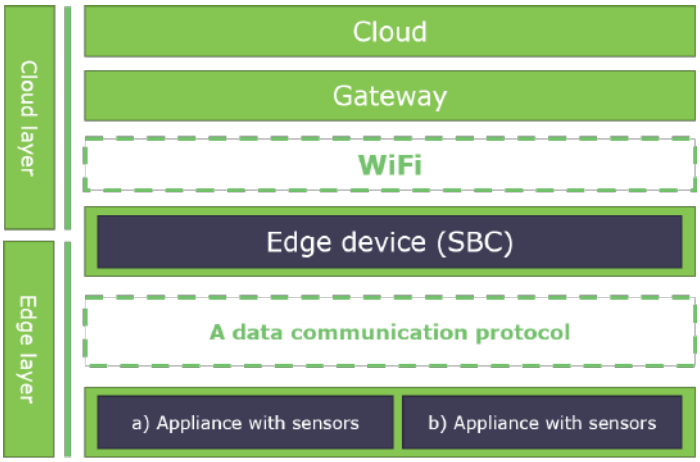


Figure 3.2: Proposed architecture stack from IoT perspective.

getting trained models from the model database container. This flow is advantageous because there will be no downtime or complications when updating models for classification.

3.2.1 Data collection and processing

The data from the MCU to SBC is gathered via a data communication protocol. All sensor readings are transmitted and stored in the SBC database. SBC is used as an edge device that will do some data prepossessing beforehand sending data to the cloud. The signals that we are planning to collect from each sensor are described in Table 3.2. Data is going to be collected at a frequency of 1Hz (1 sample per second), except for accelerometer readings. Due to the nature of acceleration measurement, we need to collect the data in higher frequencies. We plan to measure with a sample rate of $\approx 200Hz$ (200 samples per second). This data is stored in the SBC database. A diagram illustrating the final pipeline starting from the appliance and finishing with data storage, as shown in Figure 3.4. The proposed solution allows using multiple water-flow sensors to measure water intake and exhaust. These sensors will provide us with L/min information. We capture if the water was flowing during the sample snapshot (true if it is running and false if it is not). Furthermore, we use multiple temperature sensors to measure the water temperature of the intake and drain hoses.

Table 3.2: The measurements that are captured from appliances and sampling frequency. Source: author.

Planned to use sensor outputs				
Accelerometer	**Current sensor**	**Water-flow sensor**	**Temperature**	**Humidity**
X,Y,Z axis m/s^2	V, I, W, Hz, Pf, kWh	L/min, L, true, false	water hose intake/exhaust, ambient ^oC,	RH in %
Sampling frequency of sensors				
200Hz	1Hz			

We expect to receive already processed "clean" data from current, water flow, temperature, and humidity sensors. This data does not require signal processing after being acquired from the MCU board. We presume that it is already clean. Although we might experience some variations between points, this data must be observed to apply any smoothing algorithms on these data points.

Accelerometer data is more complicated comparing to other sensor data, and it requires further processing. We might need to use noise-cleaning of the acquired signal, but it all depends on recording the data and observing. Furthermore, we use accelerometers data to extract more important data features,

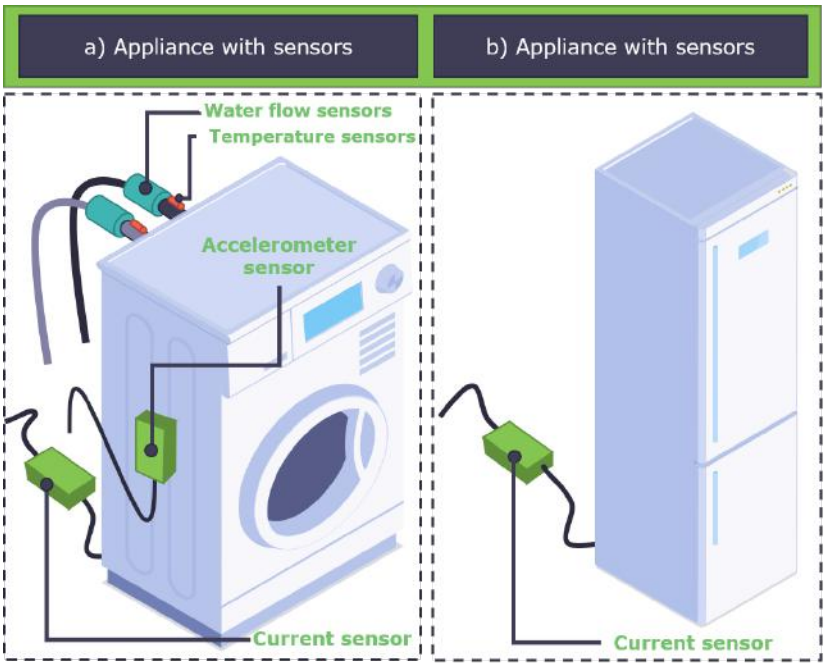


Figure 3.3: a) appliances with multiple sensors, b) appliances with single energy monitoring unit.

Figure 3.4: Data collection pipeline.

such as dominating frequencies. These dominating frequencies will be an additional feature in the feature list that will create patterns that will be used for machine learning algorithms. Each second we are taking a sample from current, water-flow, and temperature sensors. During this period, 200 accelerometer samples are acquired from the accelerometer. Then, those 200 samples are processed in each second by extracting the maximum, minimum, median, mode of the oscillation signal, adding additional value to our pattern acquisition.

3.2.2 Software design

Where we have the hardware, software design is an inevitable part of work. To tie everything up, we need to write the firmware for MCU and an agent algorithm for SBC. MCU needs to have firmware uploaded to its memory to communicate with connected sensors and extract the data from them. Moreover, an algorithm has to be written for SBC to communicate with MCU for data collection.

First, we design a firmware architecture for the MCU board, which is illustrated in Figure 3.5. To make the firmware more flexible and less dependable on the user, we start by setting a watchdog timer. It is useful when the microcontroller freezes while performing a task. "A watchdog timer (WDT) is an MCU timer that automatically generates a system reset if the main program refuses to service it periodically. It is often used to automatically reset an embedded device that hangs because of a software or hardware fault" [121]. After the watchdog is initiated, firmware splits into two main components: set up of a new

monitoring node and appliance monitoring. Set up of a new monitoring node happens if the EEPROM in a long-term information memory is empty. If so, the microcontroller starts a sequence where an access point (AP) is activated and waiting for new clients to connect. Any device with WiFi and an internet browser should be able to set up the MCU. When the client is connected via WiFi over TCP protocol, it needs to access the gateway IP address via its browser, which then receives a hosted by the MCU website. Using the website interface web client is able to send necessary data to the MCU, like information about the device that is being prepared to be monitored, WiFi SSID and password, and MQTT address, when all this necessary data is being sent from the browser to MCU via WebSocket protocol. When MCU receives this data, it is being saved into EEPROM memory, and finally, it restarts.

If MCU detects that EEPROM is not empty, the monitoring path is being executed. First, it tries to connect to Wi-Fi using supplied information from EEPROM. Then it connects to the MQTT server. Suppose the connection is established, MCU checks if it was registered in the edge device. If not, it sends a registration request via MQTT with information containing its unique id and information about the device that is being monitored. If the edge device replies to MCU with an acknowledgment of registration, it saves EPROM memory the registration status. If an accelerometer is needed, a new protothread is created for parallel accelerometer monitoring. This protothread is responsible for sending MQTT messages only with accelerometer data. Furthermore, inside the sensor initiation block, MCU checks the health of other sensors, like temperature, humidity, and current. If needed, an interrupt is created to count pulses from the water flow sensor if the water flow monitoring is required. The main thread is responsible for reporting all sensor data except accelerometers. Each second it detaches the interrupt to stop counting pulses while sensor data is being collected after it is sent. It detaches the interrupt, and the cycle repeats.

On the other side of the MCU, there is an SBC that receives the data. This SBC implements an Agent responsible for the communication between the monitoring MCU node and edge device. When Agent starts running in the edge device, it first connects to the same MQTT broker service as all MCU monitoring nodes. Then it connects to the Kafka service and finally to the database. When everything is connected, Agent listens for incoming messages. Three different messages are accepted by this Agent: register, accelerometer, and sensor. If a register message is received from a new node, Agent registers a new microcontroller inside the database using its unique ID, which device type and model it monitors. If registration is successful, it replies to the microcontroller with acknowledgment and proceeds with the registration at the SATO platform. If an accelerometer message is received, it starts accumulating X, Y, Z-axis accelerometer data into arrays for later processing. Lastly, the third type of message allowed by the Agent is the sensor message. This message contains information about sensor values, and when it arrives, it is triggering an event to calculate the feature extraction transform of received X, Y, Z-axis values.

3.2.3 Appliance classification and pattern recognition

Data that is obtained by our prototype could be used for appliance classification. There are multiple advantages to implement appliance classification, such as detailed energy consumption reports per appliance and its functioning cycle, fault recognition, meaning when we are monitoring one device. For example, if it is continuously miss-classified, which could indicate that the appliance became faulty and its consumption patterns started to drift in any way, which would flag further investigation.

Based on recorded data, we aim to recognize the appliance type, model, and programs of multi-functional appliances, e.g., washing machines have multiple washing cycles to choose from or a fan with multiple speeds. We are experimenting with various classification algorithms and classification

approaches. We compare two model implementation approaches where a single model will be trained to classify appliance device type models and their functioning cycle. The second approach is to create a hybrid model that will split the classification problem into multiple smaller classification problems. For instance, the first layer would be responsible for classifying only appliance types. The second layer would classify its brand/model, and the third one would classify appliance action or a function cycle. Each model would be trained on more specific data, which might give us an advantage over better accuracy rather than using a "one for all" model, which would be overloaded with enormous amounts of data that it would classify, which might burden the model and decrease its accuracy. An illustration of both classification approaches is illustrated in Figure. 3.7.

The classification would follow tree typology. The top highest layer model would classify appliances by type, the second layer would be specified for the device brand among other same type brands, and the third level of the three would classify specific appliance cycle or program if the appliance is multifunctional.

The second part of the illustration shows a possible approach for fault detection using auto-encoders. Auto-encoder is an unsupervised artificial neural network that learns to compress and encode data efficiently. It later learns how to reconstruct the same data back from the reduced encoded representation to a representation that is as close to the original input as possible. Well trained auto-encoder can be implemented as a fault detection tool. For example, if our machine learning model classifies cycle X, this cycle time-series data is fed into the encoder part, creating a reduced representation of received new data and passes. This representation can be passed to the previously trained decoder of a cycle X., Depending on the reconstruction. We can compare the differences between healthy looking signal and a reconstructed one. Based on these signal differences, we can raise red flags if the signal has changed, meaning abnormal behavior of an appliance functioning cycle. This approach is practical when we want to predict early faults of an appliance and suggest maintenance if needed. The same approach might be used for any other monitored appliance.

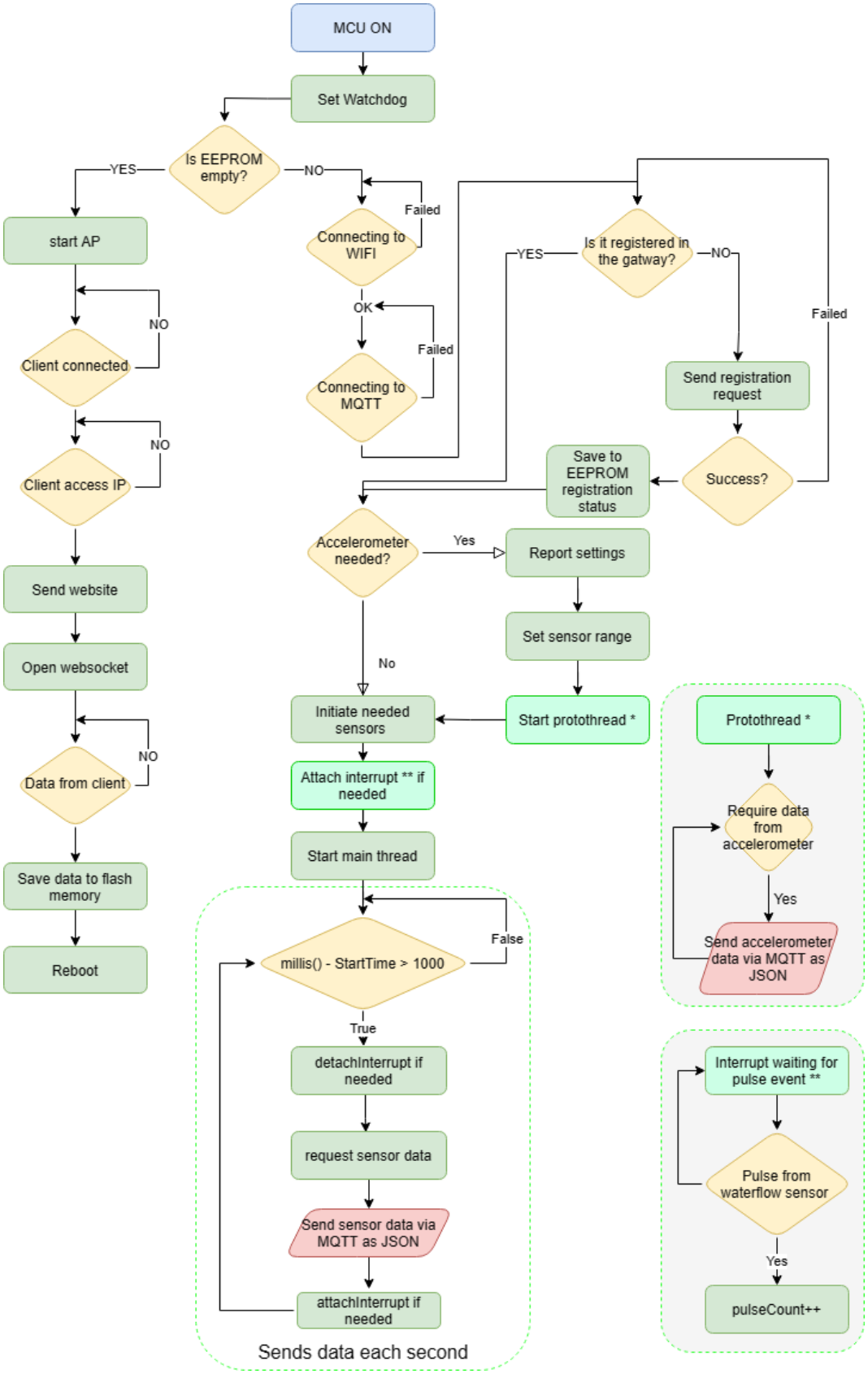


Figure 3.5: MCU firmware prototype design.

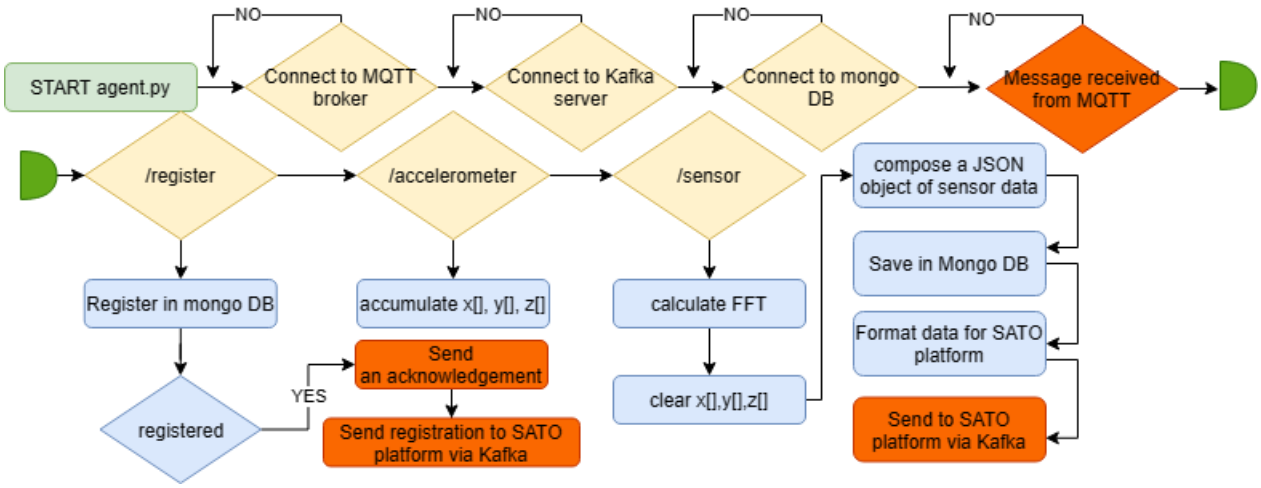


Figure 3.6: Agent flowchart.

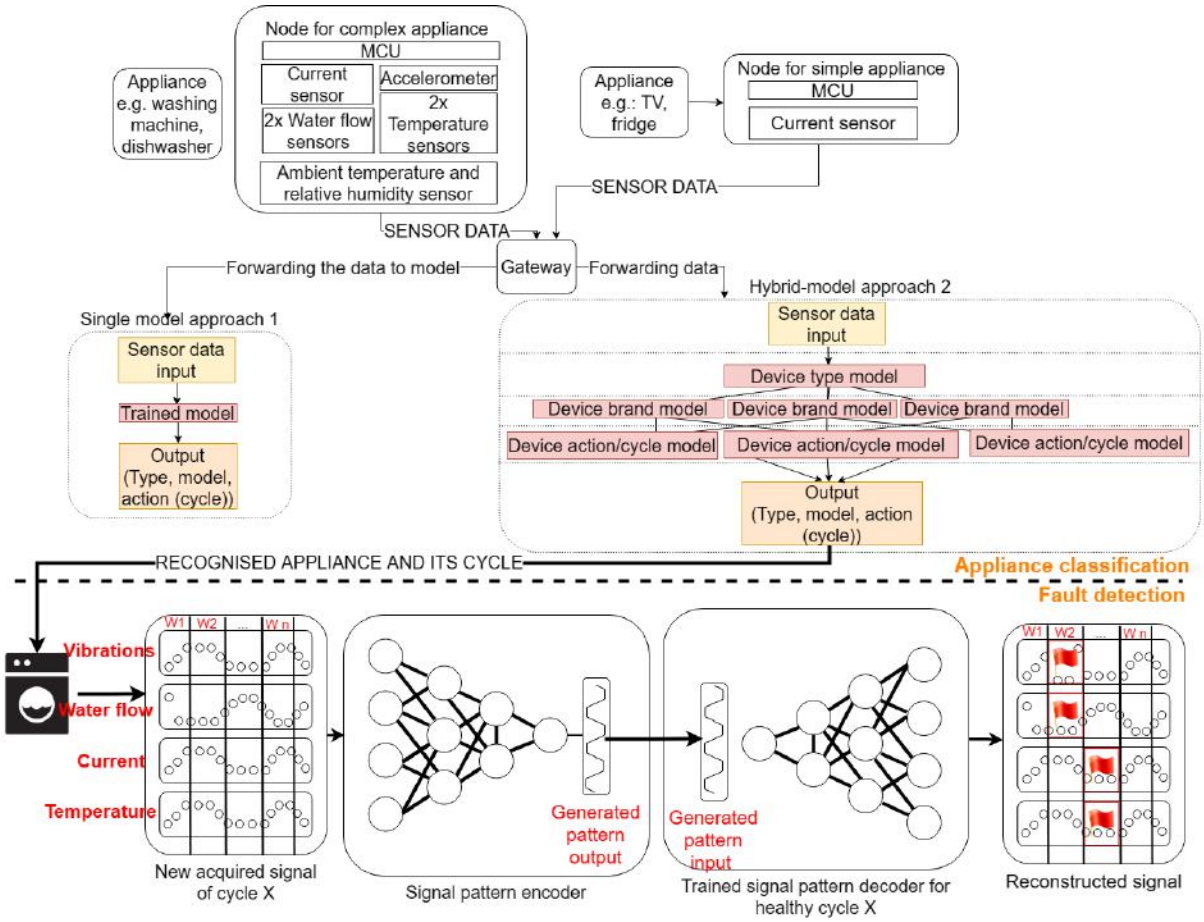


Figure 3.7: Appliance classification and fault detection approach

Chapter 4

Prototype Implementation

As mentioned in the design section, to reach our goal of creating a monitoring system capable of recording appliance energy consumption, water consumption features, the temperature of water intake and drain hoses, and humidity and temperature of the ambient, we will need various sensors. Four main parts are involved in the system. It includes sensors, a microcontroller unit (MCU), an SBC, and a cloud infrastructure that runs machine learning algorithms.

In this section, we discuss, in detail, a prototype implementation. We specify the hardware used in the prototype. Furthermore, to make use of hardware, we need to take care of the software part that will allow us to acquire sensor data and store it in the SBC (Raspberry Pi gateway). The software part is discussed in detail in a software subsection. To implement the prototype, we are following the architecture that is illustrated in Chapter 3, Figure 3.1.

The following sections discuss the appliance hardware in detail, the software and firmware installed inside nodes, including new node registration, data collection, and device classification experiments.

Beforehand going into this section, Table 4.1 describes hardware that is going used in the prototype development, as well as the communication protocols of the sensors and supported communication protocols of SBC and MCU. In upcoming sections, "MCU" means Arduino Nano IoT 33 microcontroller board.

Table 4.1: List of hardware that is used in the prototype. Source: author.

SBC			
Manufacturer	**Model**	**Communication protocol**	**Use case**
Raspberry Pi	4 B 8GB	UART, I^2C, 1-Wire, WiFi, Bluetooth etc.	Data collection and local storage from MCU Edge device that is sending data to cloud
MCU			
Manufacturer	**Model**	**Communication protocol**	**Use case**
Arduino	Nano IoT 33	UART, I^2C, 1-Wire, WiFi, BLE, Bluetooth etc.	Communication with sensors Data transmission to SBC
Sensors			
Type	**Model**	**Communication protocol**	**Provided data by the sensor**
Current sensor	PZEM-004T	UART via RX/TX	V, I, W, Hz, Pf, kWh
Accelerometer	ADXL-345	I^2C bus via SLC, SDA	X,Y,Z axis m/s^2
Accelerometer	MPU-6050	I^2C bus via SLC, SDA	X,Y,Z axis m/s^2
Water Flow	FS300A G3/4	-	L/s, water flowing true/false
Water Flow	YF-DN50 G2	-	L/s, water flowing true/false
Temperature sensor	DS18B20	1-Wire Bus	Temperature in Celsius
Temperature/Humidity	DHT22	-	Temperature in Celsius, RH

4.1 Hardware

We selected sensors that would be low-cost to implement during the selection of hardware, although supplying good accuracy and are thrust-worthy. For the MCU, we have chosen a novel Arduino board called Arduino Nano IoT 33, which supports all low-end communication protocols that provide wireless communication protocols WiFi 4, Bluetooth, and BLE. Finally, for SBC, we have selected a Raspberry Pi 4 due to its support of communication protocols and a complete operating system based on Linux. It can execute tasks like light data prepossessing before sending data to the cloud, which is considered an edge device from an IoT perspective.

Microcontroller Unit

For the Microcontroller Unit (MCU), we have selected a low-cost Arduino Nano IoT 33. It is dedicated to IoT applications, supporting the most popular communication protocols among IoT devices. The board's main processor is a low-power Arm® Cortex®-M0 32-bit SAMD21. The WiFi and Bluetooth® connectivity is performed with a module from u-blox, the NINA-W10 [122], a low power chipset operating in the 2.4GHz range. NINA-W10 supports Wi-Fi 802.11b/g/n and dual-mode Bluetooth v4.2 [123]. Arduino Nano IoT 33 features include:

- **Microcontroller**. SAMD21 Cortex®-M0+ 32bit low power ARM MCU
- **Radio module**. u-blox NINA-W102
- **Operating Voltage**. 5V
- **Input Voltage (limit)**. 21V
- **Clock Speed**. 48MHz
- **CPU Flash Memory**. 256KB
- **SRAM**. 32KB

This MCU was a perfect choice for our project because it supports enough connection ports for sensors that are going to be used. Major communication protocols are also natively supported on this board. Furthermore, it is low-power and low-cost. All sensors that will be used in this project will be connected to this MCU board, which will convert sensor data to measurable values and send it to a single board computer.

Single board computer

For the Single board computer (SBC), we have chosen the latest version (by the time of this thesis) of the Raspberry Pi 4 Model B, 8GB ram model. It has Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz processor, 8GB LPDDR4-3200 SDRAM, supports 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE communication protocols.

From the IoT perspective, SBC will be our edge device. It will serve as a local database and local edge data preprocessor before sending data to the cloud. As mentioned in the design part, it communicates with the MCU via a simple USB connection. We are using Raspberry Pi OS, a Debian-based operating system, and it is the most stable operating system to use with Raspberry Pi computer. Although, it is possible to install third-party software such as Ubuntu Desktop, Ubuntu Server, Ubuntu Core, or even

Windows operating system. It does not have a dedicated HDD or SSD. Instead, it uses a MicroSD memory card for internal memory and operating system.

Sensors

In this section, we mention sensors that are used in the prototype. It briefly reviews the capabilities and communication protocol of the sensor. We also include the wiring diagrams of sensors to connect them to Arduino Nano IoT 33, wiring illustrations were made using diagram development *Fritzing software* [124].

For energy metering, we have selected **PZEM-004T** current sensor. Many of the energy monitoring projects reviewed in related work used multiple sensors to monitor electricity, mainly two sensors, one for current measurement and the second one for voltage in the power line measurement. PZEM-004T provides both measurements in one module. From a wiring perspective, it is slightly more complicated to set it up than other sensors, although the study done by Khwanrit et al. [25] shows that this sensor has a superb metering accuracy. Due to this reason, we have selected it during hardware selection. The sensor is providing considerable energy features RMS Current I, RMS voltage V, power W, energy kWh, power line frequency Hz, power factor pf (it is used to determine the energy efficiency in the circuit, it is a ratio between true power and apparent power). The sensor communicates with MCU using UART communication protocol with a bound rate of 9600 via serial receiver RX, transmitter TX ports. Functional boundaries of a sensor:

- **rmsVoltage**. Measuring range: 80 - 260V, resolution: 0.1V.
- **rmsCurrent**. Measuring range: 0 - 100A, starting measure current: 0.02A, resolution: 0.01A.
- **Active power**. Measuring range 0 - 23kW, starting measure power: 0.4W, resolution: 0.1W
- **Power factor**. Measuring range 0.00 - 1.00, resolution 0.1Hz.
- **Active energy**. Measuring range 0 - 9999.99kWh, resolution 1Wh.

A PZEM-004T connection with Arduino nano IoT 33 MCU board is illustrated in Figure 4.1, where GND to GND, 5V to 5V, and RX goes to TX, and TX goes to RX, which makes sense because the transmitter is communicating with the receiver. The sensor has four input ports that need to be interfaced with electricity wires. L, N means hot and neutral wires, and CT means a current transformer. A hot wire of a load passes through the current transformer, measuring the current by converting the magnetic field into a measurable value. Hot and neutral wires are used to measure the voltage and frequency in the electricity line.

Since we are working with electricity, we used dedicated electrical boxes to isolate sensors to follow security guidelines. After setting up the sensor, it will step in between the appliance and the wall socket to measure energy consumption. Figure 4.2 illustrates this setup. The left side shows a box with the connected sensor inside, and on the right side, we can securely close the sensor inside the electrical box.

For oscillation monitoring we have prepared an **ADXL-345** and **MPU-6050** MEMS Tri-Axis accelerometers. Both sensors are digital and communicate using an I^2C bus that uses two pins SCL for clock pin and SDA for data pin. They supply data of three axes oscillations, in X, Y and Z-axis. These low-cost sensors have a wide application scale. Many researchers used these sensors in different applications related to vibration measurement. They are quite sensitive to even the smallest vibrations, and we believe it is a good fit for our application. Both ADXL-345 and MPU-6050 have a full-scale

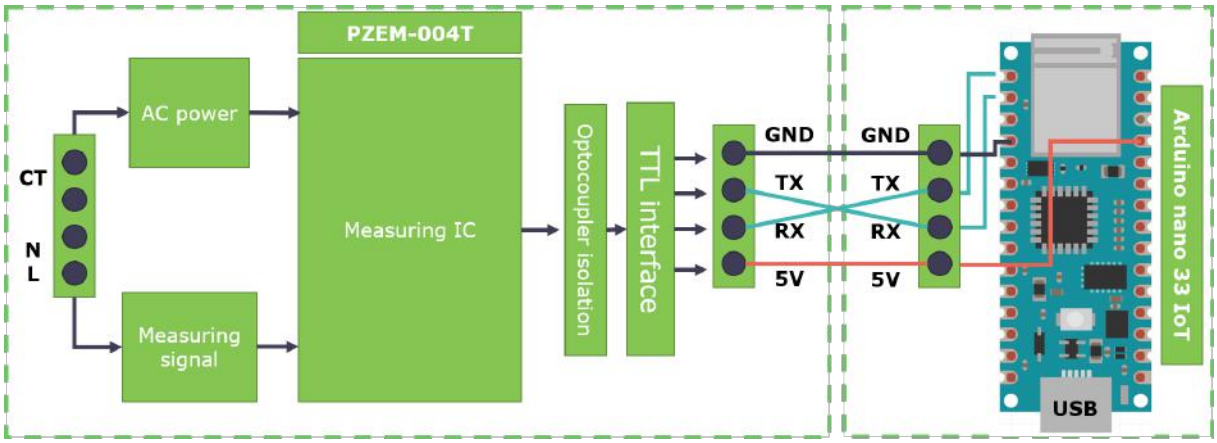


Figure 4.1: PZEM-004T interfacing with Arduino Nano IoT 33.

Figure 4.2: PZEM-004T setup.

programmable range of +-2g,+-4g,+-8g, and +-16g. We are recording vibrations with a sampling rate of 200Hz, which should be sufficient for our application. To protect the boards, we have placed them securely in special plastic cases as illustrated in Figure 4.3, which required some soldering and wiring to adapt sensor pins to the case pins, but these connectors will simplify sensor attachment to the Arduino Nano IoT 33. Although MPU-6050 and ADXL-345 pin-out are different, the casing creates a synchronized pin-out. Sensor wiring with Arduino Nano IoT 33 is illustrated in Figure 4.4. We attach these sensors to appliances using a magnet since the majority of the appliance casing is made out of steel or using M5 double-sided tape.

For water-flow measurement, we are using **FS300A G3/4** flow-meter model for the water intake hose. For the water drain hose, we need a bigger diameter sensor. For that reason, we use **YF-DN50 G2** flow-meter model. They are based on the Hall-Effect sensor discussed in the sensor part of related work, and from a measuring perspective, they are the same. However, due to different diameters, we need to calibrate them programmatically to calculate the water quantity passed through. Technical features of **FS300A G3/4** flow-meter:

- **Flow range**: 1-60L/min
- **Working pressure**: $<$1.2/Mpa
- **Operating temperature**: $\leq$80℃,
- **liquid temperature**: $\leq$120℃.

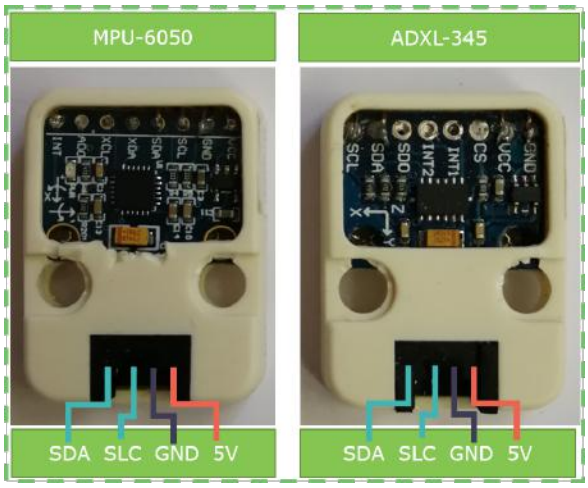


Figure 4.3: MPU-6050 and ADXL-345 inside casing.

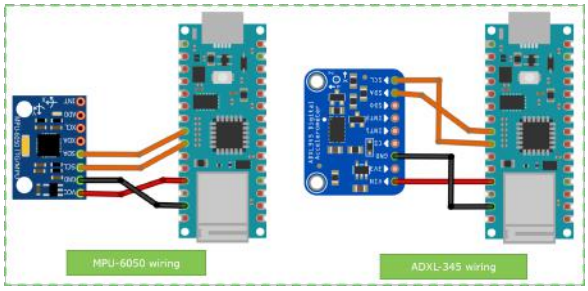


Figure 4.4: MPU-6050 and ADXL-345.

Technical features of **YF-DN50 G2** flow-meter:

- **Flow range**: 10-200L/min
- **Working pressure**: $<$1.75/Mpa
- **Operating temperature**: $\leq$80°C,
- **liquid temperature**: $\leq$120°C.

We had to calibrate the sensor by adjusting the number of ml per electrical pulse or, in other words, ml quantity per the whole spin of a wheel inside the sensor. We adjusted it by pouring water inside of it and checking the sensor output. We tested it multiple times by pouring 1L of water to pass through the sensor. We have observed that the suitable calibration sensor can be quite precise. Only +-4 ml out of 1000ml, making +-0.4% error per liter. The wiring is illustrated in Figure 4.5, where sensors are connected to D2 and D3 digital ports of Arduino MCU, each port is responsible for tracking the water flow separately.

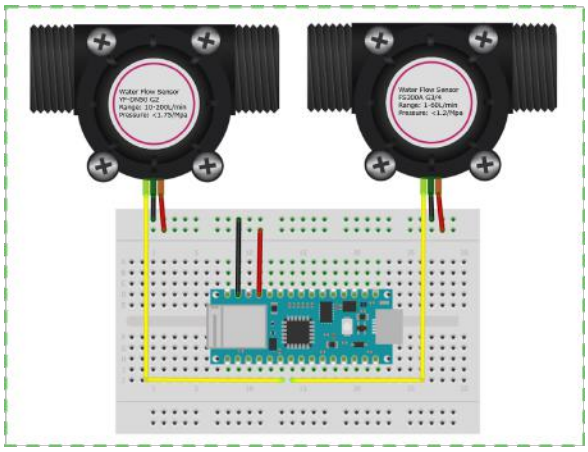


Figure 4.5: Water-flow sensors wiring to MCU.

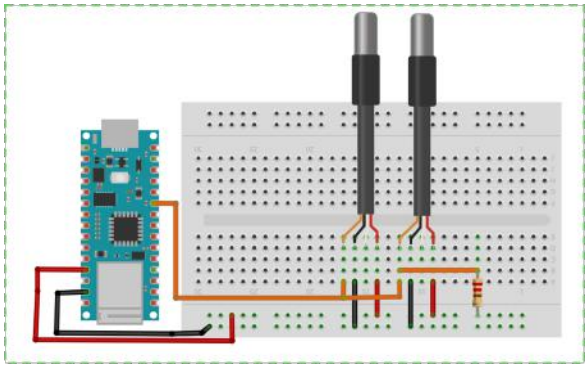


Figure 4.6: Multiple DS18B20 wiring to MCU.

To measure the temperature of appliance hoses that use water, we are going to use **DS18B20** digital waterproof temperature sensor. It supports 1-Wire communication, meaning that we can connect multiple sensors under the same bus. We will use two temperature sensors under the same bus. One is dedicated

to water intake hose temperature measuring, and the other is for the water drain hose. The wiring of multiple sensors to the Arduino Nano IoT 33 is illustrated in Figure 4.6, where the black wire is ground, the red wire is 5V, and the orange wire is 1Wire bus of two sensors connected to a D7 pin in MCU, this setup also requires a pull-up resistor of 4.7kΩ. Temperature range: -55°C to +125°C, with an accuracy of ±0.5°C.

To measure ambient temperature, we use **DHT22** digital sensor. It is convenient to use because it has integrated humidity and temperature sensors. The sensor can measure in temperature range: -40°C to 80°C, humidity range 0% to 100%. The sensor resolution is 16-bit. The sensor is connected to the D4 pin of an MCU, as illustrated in Figure 4.7.

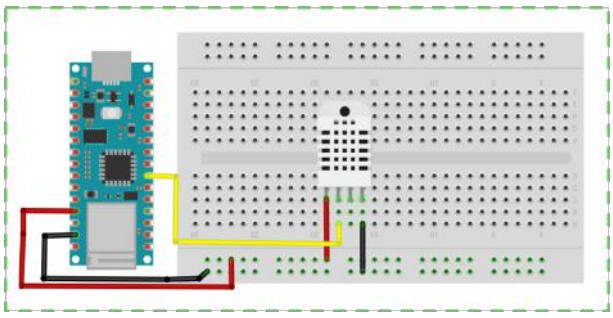


Figure 4.7: DHT22 wiring to MCU.

Dock development for all sensors

Although breadboard is convenient for rapid development, sensor testing, and temporary circuit testing, it is not useful to securely connect all the sensors to the MCU board for appliance testing because of breadboard connection fragility, especially carrying from one place to another. That is why we came up with the dock to securely and efficiently connect all the sensors into the MCU board. An illustration of all connector sensors to the breadboard can be found in Figure 4.9. The dock is based on this schematic. For all sensors that we use, we created connectors based on *GROVE 2MM 4PIN* male and female connectors, manufactured by *Seeed Technology Co., Ltd*. The sensors possess male connectors, while the MCU dock uses female connectors. Groove female connector dimensions are illustrated in Figure 4.8. All measures mentioned in the Figure are in mm. They are perfect because the maximum number of connectors used by sensors is four, a minimum of three. Each current connector rated 2A, and voltage rates 250V, which is fully sufficient for our application. The board itself was made based on a custom Printed Circuit Board (PCB).

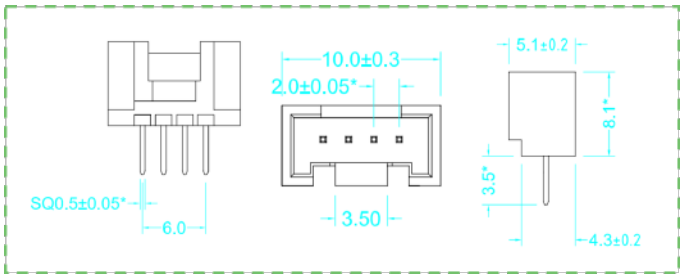


Figure 4.8: Female ports used to connect sensors to the dock.

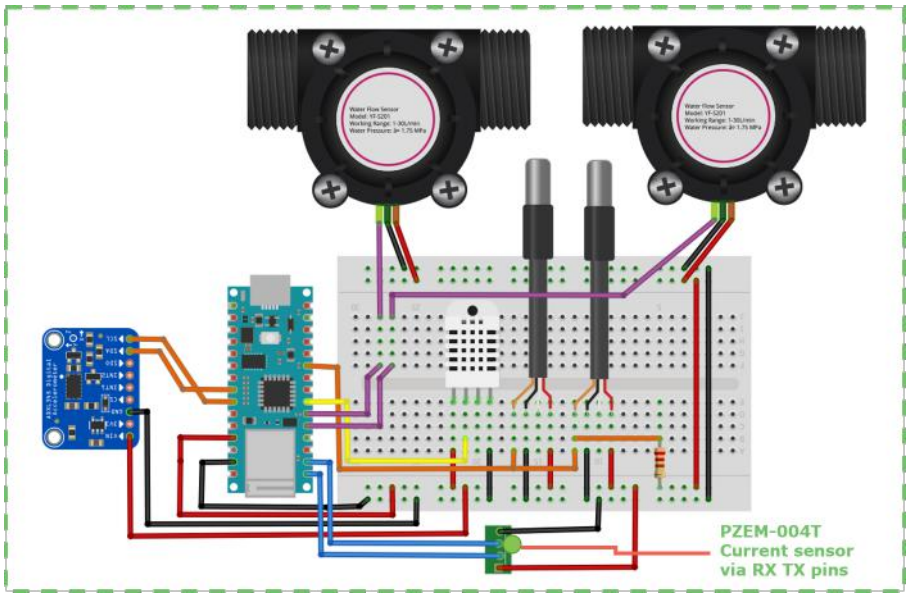


Figure 4.9: Schematic of node sensor wiring to MCU board.

The dock has all the ports necessary to connect the sensors used in the project. As illustrated in Figure 4.10, each sensor has a dedicated port on the dock. Ports D2 and D3 are used for water-flow sensors, D4 is used for temperature/humidity DHT22 sensors, I^2C port is used for accelerometers MPU-6050, ADXL-345, RX TX UART is used for current PZEM-004T sensor, D7 is used for DS18B20 water-proof temperature sensor, a 1-Wire communication protocol is allowing to use multiple temperature sensors in the same port. Hence we made an adapter to connect multiple temperature sensors exploiting a single port. Adapter is illustrated in Figure 4.11, the adapter wiring is based on Figure 4.6. Dock also has an integrated RGB LED that is used to inform about MCU's operational state. Dock has a free Analogue port A2 that is currently not used, but we might use it for an additional analog sensor in the future.

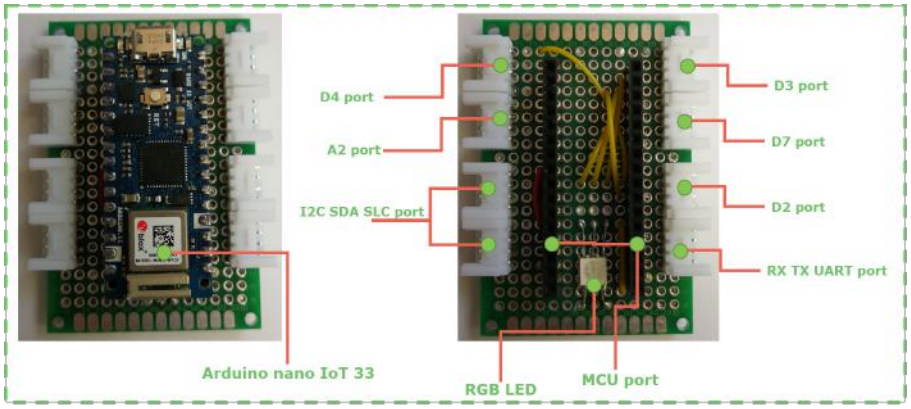


Figure 4.10: Prototype dock for Arduino Nano IoT 33.

All sensors have a male connector to easily connect the sensors to the dock in plug and play manner. An example is illustrated in Figure 4.12, where we have a DHT22 temperature/humidity sensor that is going to be simply going to be connected to the dock D4 port. The rest of the sensors are connected in the same manner into their dedicated ports, but they have much longer cables for easy access to the appliance. Wires length of other sensors is 2 meters.

Figure 4.11: Multiplexer using single MCU port for multiple DS18B20 sensors.

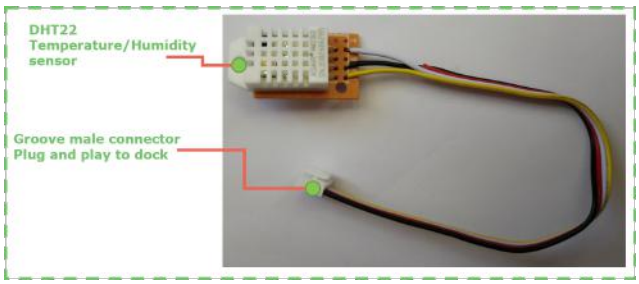


Figure 4.12: DHT22 temperature/humidity sensor with groove based male four pin connector.

4.2 Software

For the Arduino board to "talk" with the IC of the sensors, we need to write firmware and upload it to an Arduino board. For this task, we can use various libraries and write "Arduino" code that is based on C language. Arduino has an enormous community, and there are plenty of libraries for supporting a variety of sensors. The ecosystem has an open-source dedicated integrated development environment (IDE) [125] for writing and uploading the code to the MCU. Furthermore, it facilitates serial communication between an MCU board and a computer. The Arduino IDE has an integrated serial monitor for receiving the data from MCU and using it as a debug tool. Code libraries used in this project are under X11 license, meaning permission is at this moment granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software") to deal in the software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software is furnished to do so.

For the Raspberry Pi, a python code has been written called Agend.py. The Agent communicates with Arduino Nano IoT 33 board via MQTT messages simultaneously while receiving the data from the Arduino board. The code also does some data prepossessing and facilitates storing the data appliance consumption data into local MongoDB.

Software for an Arduino MCU

To work with the DHT22 sensor, we have used an Arduino library for the DHT series of low-cost temperature/humidity sensors, written by Adafruit Industries [126]. This library allows us to communicate with the DHT22 IC by providing the DHT22 data pin of the Arduino board, which in our case is D4. The library supports DHT11, DHT22 temperature and humidity sensors. After setting it up, we can read the temperature in Celsius and relative humidity (RH) captured by the sensor. The listing 4.1 snippet shows temperature and humidity each second via serial monitor when the sensor is connected to the D4 pin:

```
1 #include <DHT.h>;
2
3 #define PERIOD  1000 // frequency of reading the sensor
4 #define DHTPIN 4     // what pin we're connected to
```

```
#define DHTTYPE DHT22   // DHT 22  (AM2302)
DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor

void setup() {
    Serial.begin(115200);
    dht.begin();
}

void loop() {
  if(millis() - StartTime > PERIOD){
    StartTime = millis();
    hum = dht.readHumidity();
    temp= dht.readTemperature();
    Serial.println(String("DHT sensor") + hum + "% " + temp "C"
    }
}
```

Listing 4.1: Arduino code to read temperature/humidity sensor

To be able to read values from DS18B20 temperature sensor, a *OneWire* [127] that is combined with *DallasTemperature* [128] libraries were used. To set it up, the library is requiring to provide the digital pin of the MCU. DallasTemperature library automatically detects connected even multiple sensors because each out of the factory has a unique ID and identifies itself in 1-Wire Bus. After detecting the sensors, the temperature value is being asked by sensors index. While MCU is communicating with the specified sensor, the other temperature sensor is on "hold." A listing 4.2 snippet is represented to read multiple DS18B20 sensors:

```
#include <DallasTemperature.h>
#include <Wire.h>

#define PERIOD  1000 // frequency of reading the sensor in ms
#define ONE_WIRE_BUS 7 // data wire plugged into D7
OneWire oneWire(ONE_WIRE_BUS);
// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);
int deviceCount = 0;

void setup() {
    Serial.begin(115200);
    sensors.begin();
    // Gets the number of devices and uses them as index
    // If single sensor used, we can just pass 0 as sensor index.
    deviceCount = sensors.getDeviceCount();
}

void loop() {
  if(millis() - StartTime > PERIOD){
    for (int i = 0;  i < deviceCount;  i++)
        {
          Serial.print("Sensor ");
          Serial.print(i+1);
          Serial.print(" : ");
          tempC = sensors.getTempCByIndex(i);
          Serial.print(tempC);
          Serial.print((char)176);//shows degrees character
```

```
        }
    }
}
```

Listing 4.2: Arduino code to read DS18B20 temperature sensor

To read MEMS MPU-6050 and ADXL-345 accelerometers we are using *Adafruit-MPU6050* and *Adafruit-ADXL345* libraries. When the sensor is connected to I^2C via SDA, SLC pins, the library automatically discovers the sensor. These libraries allow us to set the measuring scale range of the sensor, and selectable ranges include +-2g/ +-4 g/ +-8 g/ +-16 g/. Furthermore, the data rate of a sensor is selectable from 0.10Hz up to 3200Hz. It means a value update rate of a sensor itself. For example, if MCU is reading values from the accelerometer at a rate of 200Hz, but the sensor value update rate is 20Hz, the sensor will update the value only after 20 readings by the MCU, creating square waves. A listing 4.3 code snippet is of reading ADXL-345:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>

/* Assign a unique ID to this sensor at the same time */
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

void setup() {
    Serial.begin(115200);
    accel.setRange(ADXL345_RANGE_16_G);
    // Gets the number of devices and uses them as index
    // If single sensor used, we can just pass 0 as sensor index.
    deviceCount = sensors.getDeviceCount();
}

void loop() {
  sensors_event_t event;
  accel.getEvent(&event);
  /* Display the results (acceleration is measured in m/s^2) */
  Serial.print("X: "); Serial.print(event.acceleration.x);
  Serial.print("Y: "); Serial.print(event.acceleration.y);
  Serial.print("Z: "); Serial.print(event.acceleration.z);
}
```

Listing 4.3: Arduino code to read ADXL345 accelerometer sensor

For interaction with the current sensor PZEM-004T, we have used a *PZEM004Tv30* library. Originally this library was written for ATmega328P microcontrollers by J. Mandula [129]. However, it was not compatible with Arduino Nano IoT 33 microcontroller that is based on SAMD21, we have made some compatibility adjustments in the source code of the library, and it is available in [130]. To interact with the sensor, open a serial UART communication with the sensor via RX, TX pins. In the Arduino code *Serial* is for communications via USB connection and *Serial1* is for onboard serial communication. First, we need to start Serial1 communication by providing a bound rate. Then we provide the address of the sensor to the PZEM004TV3 library. The current sensor has memory in which it stores the energy consumption of the session. We can programmatically wipe the memory by sending a request *pzem.resetEnergy()* which will reset the sensor. In the main loop, we require to return sensor readings from its IC. A listing 4.4 snippet to read the PZEM-004T sensor:

```
#include <PZEM004Tv30.h>
```

```
2
3 #define PERIOD  1000
4 PZEM004Tv30 pzem(Serial1, PZEM_DEFAULT_ADDR);
5
6 void setup() {
7     Serial.begin(115200);
8     Serial1.begin(9600);
9     pzem.setAddress(0x42);
10    pzem.resetEnergy();
11 }
12
13 void loop() {
14    if(millis() - StartTime > PERIOD){
15        StartTime = millis();
16        float volt = pzem.voltage();
17        float cur = pzem.current();
18        float powe = pzem.power();
19        float ener = pzem.energy();
20        float freq = pzem.frequency();
21        float pf = pzem.pf();
22    }
23 }
```

Listing 4.4: Arduino code to read PZEM-004T sensor

For water flow measurement, we use FS300A G3/4 and YF-DN50 G2. The main idea is to count the pulses that the sensor is sending to interact with the water-flow sensor. The sensor is based on Hall Effect. When water starts passing through the sensor, a little magnetized wheel triggers the Hall sensor, creating square wave pulses. These pulses are then counted by the MCU board and converted into water quantity. By counting the presence of pulses, we can also determine if the water was passing or not by the time the MCU took the measurement. A sensor interrupt command is used for this application. It allows the microcontroller to react to a pulse event. When a pulse is detected, an event is triggered calling *pulseCounter* function that will increment *pulseCount*. A listing 4.5 code example is illustrated for pulse counting and conversion to water quantity:

```
1 byte sensorInterrupt = 2;  // 0 = digital pin 2
2 byte sensorPin       = 2;
3 // Sensor outputs approximately 5.942 pulses
4 // per second per litre/minute of flow.
5 float calibrationFactor = 5.942;
6 volatile byte pulseCount;  // counts pulses from sensor rotations
7 float flowRate;
8 unsigned int flowMilliLitres;
9 unsigned long totalMilliLitres;
10 unsigned long oldTime;
11 bool water_flowing = 0;
12
13 void setup() {
14     Serial.begin(115200);
15     attachInterrupt(sensorInterrupt, pulseCounter, FALLING);
16 }
17
18 void loop() {
19     if(millis() - StartTime > PERIOD){
20         detachInterrupt(sensorInterrupt);
```

```
21
22          flowRate = ((1000.0 / (millis() - oldTime)) * pulseCount)...
23          ... / calibrationFactor;
24          oldTime = millis();
25          flowMilliLitres = (flowRate / 60) * 1000;
26          totalMilliLitres += flowMilliLitres;
27          flowRate > 0 ? water_flowing = 1 : water_flowing = 0;
28          pulseCount = 0;
29          attachInterrupt(sensorInterrupt, pulseCounter, FALLING);
30      }
31  }
32
33  void pulseCounter()
34  {
35    pulseCount++;
36  }
```

Listing 4.5: Arduino code to measure water flow using Hall based water flow sensor

In the Arduino firmware, we are using *Scheduler* library. This library was created for Arduino SAM and SAMD architectures, and it allows multiple tasks to run simultaneously without interrupting each other. This is a cooperative scheduler in the sense that the CPU switches from one task to another [131]. We used a scheduler to run multiple loops independently, where the first loop is to read sensors such as water flow, current, and temperature. The second loop was for reading the accelerometer sensor. It simplifies the implementation of reading sensors. It helps avoid data loss from the accelerometer because accelerometer data must be read in higher frequencies than the rest of the sensors.

The Arduino Nano IoT 33 firmware starts by checking its EEPROM long-term memory. If memory is empty, it starts a new node registration sequence where the MCU needs to be registered in the edge device. To achieve this, our MCU starts acting as a wireless access point (WAP) to which any client having WiFi and a web browser can connect. In MCU firmware every MCU IP is 192.168.2.1, as illustrated in Figure 4.13 (a). For all WiFi operations, we are using *WiFiNINA.h* library can serve as either a server accepting incoming connections or a client making outgoing ones [132]. When the web client access the MCU via its IP address in the browser, MCU replies with a web application that contains HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and Javascript code inside a single HTML file. However, inside MCUs firmware, that web application is converted into base64 encoding that represents binary data (more specifically, a sequence of 8-bit bytes) in an ASCII string format by translating the data into a radix-64 representation. To convert our web application, we used a Linux command *cat page.html | gzip | base64 -w0* which returns a converted array from regular HTML into a base64 encoding. *Page.html* in this command is our HTML page that is being later loaded into the MCU.

Web client and MCU is communicating via WebSocket that is being opened between client and MCU the moment web browser access the IP address and the web application is loaded at the client-side. To be able to communicate using WebSocket an Arduino library *WebSocketServer.h* was used [133]. Bi-directional communication is established over a single TCP protocol, meaning that both parties can communicate by sending messages. The client does not need to be refreshed every time a message is received.

We can supply MCU with the necessary information from the web client to function as an appliance monitor. Via web client, we send the SSID of a WiFi network, its password and appliance type, and the model of an appliance that we will monitor. Furthermore, we select the necessary sensors that are going to be used for a particular appliance. We can also provide the MQTT server IP address or host-name

as illustrated in Figure 4.13 (a,b). Before loading the web application to the client, MCU searches for available WiFi networks and creates an SSID list of available WiFi networks. The available WiFi can be selected from the drop-down menu inside the web application. In the web application, we can provide all this information and set up a monitoring node. After the information is provided and send to the MCU, we still have 15 seconds to cancel the set-up and change any of the parameters if needed, illustrated in Figure 4.13 (d). If an accelerometer module is required, for example, for a washing machine, an accelerometer module has to be selected, and it involves calibration to set the offset from the values. The accelerometer has to be placed in a tranquil position during the calibration as illustrated in Figure 4.13 (c), each second an MCU sends a message to the web client informing the completeness of the calibration inside the pop-up. It takes 40 seconds to calibrate. Each second we take measurements of X, Y, Z-axis during this 40 second period and create an average of the accelerometers measurements, which will offset all future values. This offset is saved in EEPROM memory, meaning that we do not lose the offset data even if MCU loses power or restarts. In Figure 4.13 (a), we can see a green-colored text saying connected, which means that the WebSocket is successfully established between the client and the MCU AP. Furthermore, in Figure 4.13 (b), we can see a list of sensor modules that we can select. It is an important step in node set-up because MCU will generate MQTT JSON messages that will later be sent to the edge device based on the preferences.

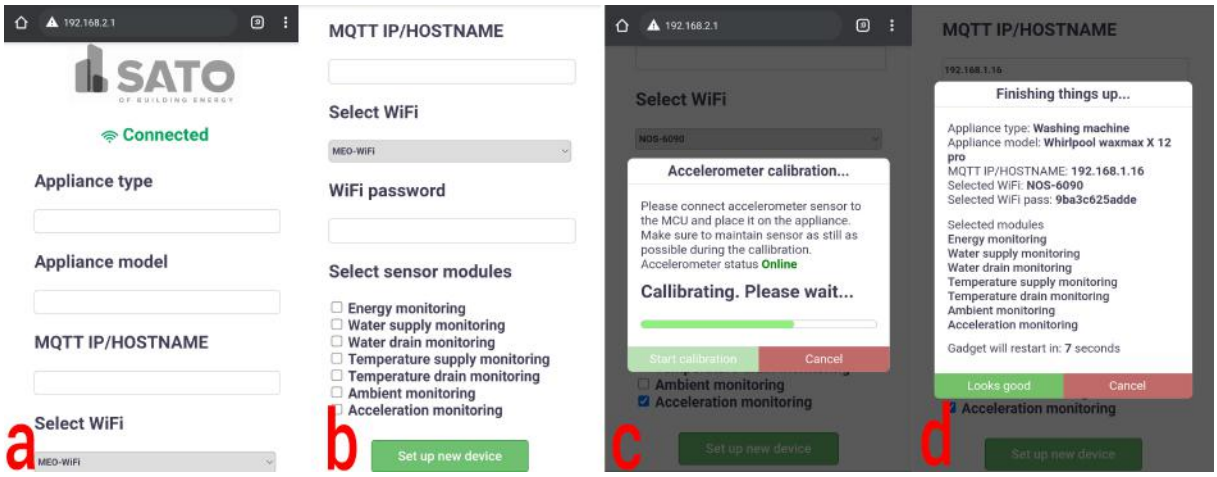


Figure 4.13: Developed web application, loaded inside MCU flash for an AP.

After set-up is complete, MCU restarts, and when it boots initiatory time after the set-up, it connects to the provided WiFI network. Next, it connects to the MQTT broker. Then MCU sends a registration message to the edge device via MQTT broker with the data of monitored appliance, activated sensor modules. Moreover, its unique ID, which is made of *SATO_MCU_MAC_address* e.g., sato _7c:9e:bd:ec:aa:dc, registration message is sent via dedicated MQTT topic called "registration" when edge device receives the registration request, it saves all this data to a local Mongo database.

When Mongo database returns a success message that the registration record has been saved, edge device agent.py replies to MCU via MQTT message. Every MCU is listening to its unique topic of registration to avoid broadcast the registration acknowledgment. For instance, an MCU is listening for "registration/MCU _UNIQUE _ID, e.g.: "registration/ sato _7c:9e:bd:ec:aa:dc" topic, meaning that when registration is requested edge device knows the MCUs unique ID, which is then used as a reply topic. When MCU receives an acknowledgment, it saves the registration status inside its EEPROM long-term memory, meaning the node will be aware of its registration status. After registration is done, the MCU

node starts a monitoring routine.

The firmware checks and sets up necessary sensors that have been selected from sensor modules during the set-up by the web application. First, it checks if the accelerometer is needed and is connected to the MCU. If so, it reports accelerometers parameters that are saved in accelerometers IC. After that, the code sets the oscillation range by using the G value. An independent loop starts for sending accelerometer readings using MQTT "accelerometer" topic. Independent loop is being initiated using *Scheduler* library. If the accelerometer sensor is not connected, the Arduino code reports that it was not detected. Then code initiates the rest of the sensors, DS18B20 temperature sensor, DHT22 temperature/humidity sensor, and PZEM004T current sensor. An interrupt is also being initiated for pulse event occurrence from the water-flow sensor. If water flow is detected, the interrupt will call a function that counts square wave pulses by incrementing *pulseCount* variable. The main Arduino loop is reporting sensor values each second via MQTT "sensors" topic. Inside the loop, we check if one second has passed since the last report. If so, the code then detaches interrupts of the water-flow sensor, requests sensor values of the current sensor, temperature sensors, and calculates water flow by calculating the pulses if there were any, if no pulse was detected. It reports that water was not flowing at that moment and vice-versa. After reporting the sensor readings via the MQTT topic, MCU firmware resets *pulseCount* variable and attaches the interrupts for the water-flow sensor, waiting one second to repeat the cycle of the main loop. Moreover, firmware is running a Watchdog timer (WDT).

Many AVR's® contain an 'Enhanced Watchdog Timer,' which runs independently on a separate on-chip 128KHz oscillator. The Watchdog Timer is typically used for pre-defined system reset duties (i.e., a fail-safe or protection mechanism against system crashes). However, given that it has its interrupt vector, the WatchDog Timer may also be used as a regular (albeit relatively restricted) time counter for managing a user-defined Interrupt Service Routine (ISR) [134].

To start the watchdog, we use a command *WatchDog::start();*, the WDT library has different timers to choose from, starting at 16ms up to 8s, meaning that in case it crashed, the watchdog timer will consider a particular timer and will reset the MCU node.

Software for Raspberry Pi computer

For data collection and storage, an Agent written in Python was created. The agent is responsible for many tasks of communication. It is responsible for connecting to the MQTT server and listen to three main different MQTT topics "sensor," "accelerometer," and "registration." If the accelerometer is used, for example, for washing machine monitoring, the agent receives accelerometer data of X, Y, Z-axis. Since we are collecting ≈ 200 samples per second, the agent collects these values inside arrays. We have three arrays for each axis. Each second we receive an MQTT message via the "sensor" topic, triggering few events. One of them is to calculate the FFT of each axis using accumulated accelerometer measurements inside arrays. The second is to create statistical data about each axis, meaning we measure the maximum, minimum, and mean recorded accelerometer data points. When these tasks are finished agent generates a specific JSON object and stores this data inside the Mongo database. Nevertheless, not all JSON objects are the same. It all depends on the selected modules during node registration via the web application. If we measure only a simple device like a TV, we will measure only energy values provided by the current sensor. Data models are illustrated in Figure 4.14. 4.14-A it is An example of a JSON object when we are monitoring, e.g., a washing machine. In that case, all sensors are being used. Furthermore, we can see that statistical data of the accelerometer is also being saved in the same object. In Figure 4.14-B, we show an example of an object when we are monitoring only a simple device like a TV or any other

appliance that requires only electricity data monitoring. In Figure 4.14-C, we can see an example record of an accelerometer.

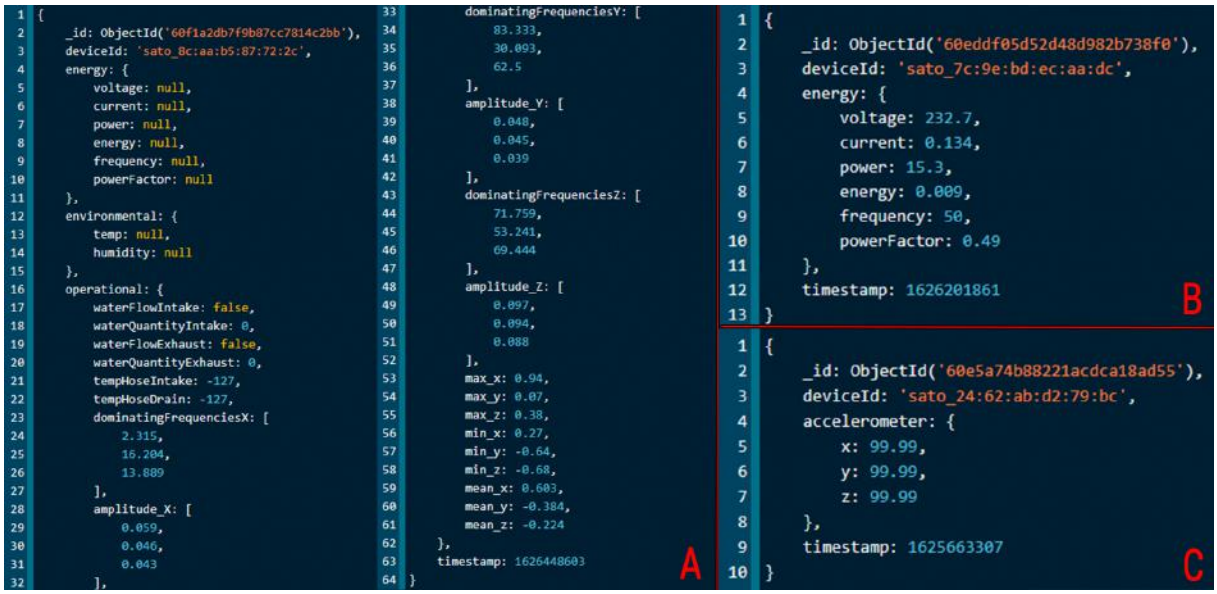


Figure 4.14: Records saved to Mongo database by the Agent.

The other task that Agent is responsible for is the registration of a new MCU monitoring node. The Agent is listening to the "registration" MQTT topic, and when it received the request from a new MCU node, it saves all the information about the node inside the Mongo database. The information that is being received includes MCU unique ID, appliance type and model, and the modules that are being used by that node.

Agent forwards incoming messages to SATO platform via Kafka publisher, topics such as "registration" and "sensor" are being forwarded. We do not send each accelerometer reading to the Kafka server because it is inefficient and would produce enormous amounts of data. The agent is responsible for converting JSON objects incoming from the MCU node before forwarding them to Kafka to the correct JSON object mapping understood by the SATO platform.

4.3 Appliance classification

For appliance classification, we implemented the KNN, RF, XGBoost, and SVM classification algorithms. Taking into consideration the appliance energy consumption pattern dataset called ACS-F2, which has been used as a reference for similar works [135, 136, 137, 138], Table 4.2 refers to the features that are considered for the implementation that we did concerning to the appliance classification. Feature set F1 contains original features from the data set ACS-F2. It has six-dimensional feature set: P - power, Q - reactive power, I - current, f - frequency in the power line, V - voltage of a power line, ϕ - phase angle. The F2 feature set is a modification of an F1 feature set where we engineer features such as S - apparent power and pf - power factor used as an indicator of electrical circuit efficiency. Furthermore, feature set F3 contains original features provided by our current sensor. The F2 feature set was created to make unity between both data-sets so that we can do experiments with the integration of our dataset to the ACS-F2 dataset. We also recorded the classification accuracy, comparing the original feature set with the updated one. We describe the formulas and how these features are being calculated in the results chapter.

Table 4.2: A list of feature sets that are used in our classification experiments.

Feature-set	Containing features
F1	P, Q, I, f, V, ϕ
F2	$P, pf, I, f, V, Q, S, \phi$
F3	P, pf, I, f, V

We have used K-fold cross-validation in our implementation of machine learning algorithms instead of splitting the data into 80% - 20% for training and testing. In cross-validation, we can split data more times. Those splits are called folds. We have used K = 10. Meaning that model will be tested during ten iterations, 9/10 will be used for training the rest for testing. An illustration showing the principle of K-Fold cross-validation is illustrated in Figure 4.15, when K = 4, it shows four iterations with different training and testing data. Using Cross-validation, we will have a better idea of how well the learner will generalize to unseen data. To automate the K Folds, we are using *sklearn.KFold* python library. It works by passing the data we want to fold and defining a random state for result reproduction. In our experiments, the random state is "1".

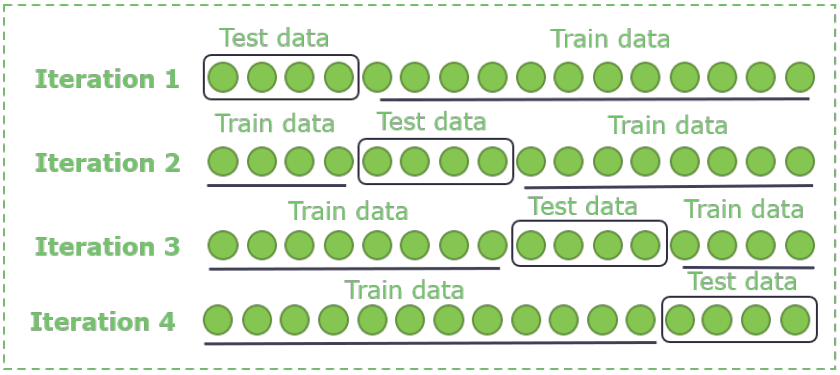


Figure 4.15: K-Fold cross validation when K = 4.

4.3.1 K-nearest neighbors

To use K-nearest neighbors (KNN), we were using *sklearn.neighbors* python library integrated into *jupyter notebook*. In this algorithm, the most important hyperparameter is the value of K, which tells the algorithm how many surrounding neighbors we are going to take into consideration to make a classification prediction on a given data point. In our experiments, we iteratively provide K from 1 to 30, and each time we record the prediction accuracy to choose the best K value. We will use prediction accuracy as a baseline to other ML algorithms. Due to its simplicity and performance, it is a good benchmark criterion for classification algorithm comparison. We have noticed that the bigger the K, the worse the evaluation score, almost linearly. Table 4.3 describes the properties used in the algorithm.

4.3.2 Random forest

For random forest (RF) implementation, we have used *sklearn.RandomForestClassifier* python library. The algorithm has multiple hyper-parameters, and when properly tuned, it might increase the model's accuracy. For hyper-parameter tuning, we have implemented a *sklearn.RandomizedSearchCV* which

Table 4.3: KNN hyper-parameters used in our implementation.

Hyperparameter	Value
weight	*uniform*
algorithm	*auto*
distance metric	*minkowski*
metric params	*none*
n neighbors (k)	*1 to 30*

automatically provides cross-validation. Randomized grid search is requiring to provide grid values, the number of iterations and K folds for cross-validation. The algorithm will try to find randomized parameters from the provided grid by multiple iterations after the last iteration algorithm returns the best-found hyper-parameters and the accuracy score. In this algorithm, the most important two parameters are the number of trees and the depth per tree. Although it has more hyperparameters that can be adjusted. *N-estimators* means number of trees in the forest, *max-features* means max number of features considered for splitting a node, *max-depth* means maximum number of levels in each decision tree, *min-samples-split* min number of data points placed in a node before the node is split, *min-samples-leaf* means minimum number of data points allowed in a leaf node, *bootstrap* method for sampling data points (with or without replacement). The grid parameters that we have used inside the grid search:

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60,
               70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000,
                  1200, 1400, 1600, 1800, 2000]}
```

4.3.3 XGBoost

Like RF, XGBoost has multiple hyperparameters that need to be tuned for higher classification accuracy of a model. To implement XGBoost, we have used *xgboost.xgb* python library to implement the model. For grid-search of XGBoost, we used *hyperopt.hp* python library. *Max-depth* means maximum number of levels in each decision tree, *learning-rate* Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and the learning rate shrinks the feature weights to make the boosting process more conservative, *min-child-weight* means a minimum sum of instance weight (hessian) needed in a child. Suppose the tree partition step results in a leaf node with the sum of instance weight less than min-child-weight. In that case, the building process will give up further partitioning, *gamma* means minimum loss reduction required to make a further partition on a leaf node of the tree, *subsample* means a ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data before growing trees. Moreover, this will prevent overfitting, *colsample-bytree* means the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed [139]. The grid parameters that were used in our experiments:

```
'learning_rate': hp.uniform("learning_rate", 0.0, 1),
'n_estimators':hp.quniform("n_estimators", 50, 400, 50),
'max_depth':hp.quniform("max_depth", 3, 15, 1),
```

```
'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1),
'gamma':hp.quniform("gamma", 0, 10, 1),
'subsample':hp.uniform("subsample", 0.7, 1),
'colsample_bytree' : hp.uniform('colsample_bytree', 0.1, 1)
```

4.3.4 Support-vector machine

For support-vector machine (SVM) implementation we have used *sklearn.svm* python library. As previously mentioned algorithms, SVM also contains multiple hyperparameters that can be tuned to perform classification better. We were experimenting with "C," "gamma," and different kernels. C value determines the miss-classifying of each training example. A smaller C value is telling the SVM to find a larger-margin separating the hyperplane. Higher *C* value makes it more prone to be influenced by outliers because the hyperplane might be divided closer towards an opposite class of outlier points, leaving smaller margins. *gamma* is used with RBF kernel. In linear or polynomial kernel, gamma is not being used. When trying to divide two clusters, higher gamma means more curvature. However, we must be careful because too high gamma makes the learner too much "tailored" for the data we insert. Nevertheless, the number of gamma is dependent on the data we use. After the grid search in our experiments, we were using settings described in Table 4.4.

Table 4.4: SVM hyperparameters used in our implementation.

Hyperparameter	Value
C	*1000*
gamma	*1*
kernel	*rbf*

4.4 Integration with SATO platform

In Appendix C.1 is illustrated overall SATO system architecture integration. In the orange field, we are marking the scope of this project report. The architectural project components are outside the SATO platform. The edge device used in this work is communicating with the integration solution of the SATO platform via Apache Kafka broker. Inside edge device, Agent is programmed to forward data to the platform. Edge device sends messages via two topics: registration and data. When a new monitoring node is being set up, it requests to be registered locally in the gateway. After the registration is successful, the gateway forwards a registration to the platform, so the platform is aware of the registered user and the appliances being monitored from a single gateway. The registration data format is being adapted by the requirements of the integration solution platform before forwarding. A registration message contains SATO platform user id, monitoring node id, monitoring node capabilities, and data about the monitored device like type and model. The monitoring node capabilities depend on the monitoring node's set-up, e.g., if we monitor TV, the only active module will be "energy"; if we monitor a complex appliance like a washing machine, it will have additional modules activated. The object of registration example of a washing machine is illustrated in Figure 4.16. To be able to send messages that platform can consume Agent.py needs server IP address called bootstrap server e.g.: *bootstrap-servers='94.62.39.186:9091'*, *bootstrap-servers='94.62.39.186:9092'*, registration and data producers are running on separate ports,

9091 and 9092. It also requires the client id of producer and consumer. In our testing we used *client-id="ZygisGateway"*.

```
{
    id: 'sato_8c:aa:b5:87:72:2c',
    sato_user_id: 'XXXXXXXXXXXXXXXXXX',
    appliance: [
        'Washing machine',
        'Whirlpool NX741'
    ],
    modules: [
        'energy',
        'water supply status',
        'water supply quantity',
        'water drain status',
        'water drain quantity',
        'water supply temperature',
        'water drain temperature',
        'ambient temperature',
        'ambient humidity',
        'vibration monitoring'
    ],
    timestamp: 1626437216
}
```

Figure 4.16: JSON object example for registration to SATO platform.

```
{
    deviceId: 'sato_7c:9e:bd:ec:aa:dc',
    id: 'energy',
    energy: {
        voltage: 232.9,
        current: 0.552,
        power: 64.2,
        energy: 0,
        frequency: 50,
        powerFactor: 0.5
    },
    timestamp: 1626977489
}
```

Figure 4.17: JSON object example of energy reading forwarded to SATO platform.

When the edge device receives energy readings from the node, it transforms this data into the JSON object format that the SATO platform can understand. Depending on the data we are receiving from the monitoring node, if it is a simple device, we send energy readings in a format illustrated in Figure 4.17. If we take a look at the example, every energy reading record has an id. This id represents the data that is being sent. For current sensor data, the id is *energy*. In this implementation, we have many ids representing different kinds of sensor readings. They are described in Table 4.5.

Table 4.5: Ids for different sensor data that is forwarded to SATO platform.

Id	Sensor data
energy	Current sensor readings
water supply status	Water flow sensor status (True/False)
water supply quantity	Water flow sensor readings of water flow quantity of intake hose
water drain status	Water flow sensor status (True/False) for exhaust hose
water drain quantity	Water flow sensor readings of water flow quantity for exhaust hose
water supply temperature	Water intake hose temperature
water drain temperature	Water intake hose temperature
ambient temperature	Temperature data from a multi-sensor of humidity/temperature
ambient humidity	RH data from a multi-sensor of humidity/temperature

Furthermore, we have experimented with the stress testing of an integration solution of a SATO platform that standardizes all the data across the platform, where messages are received. We did several tests. One of them was sending as many new monitoring node registration messages via Kafka producer as possible. In our experiments, we sent 70 000 per minute from a single gateway to stress test it. The gateway was not able to send more than 70 000 messages per minute. Each message size was approximately 253 bytes.

Chapter 5

Results

To evaluate the performance and capabilities of the proposed solution, we did a set of experiments that varies from sensor data acquisition to the classification part. Since we are interested in real-time classification, we defined experiments to test the data acquisition and pre-process the data, which allows us to understand the classification accuracy provided by chosen machine learning algorithms.

In these experiments, we got measurements from the current sensor and send them to the classification algorithms. Each experiment ran for at least 10 minutes. Every 2 seconds, it sends a measurement to the classification module and logs the true positives and false positives during its span.

We also performed experiments on multi-functional appliance cycle classification and system performance as a whole, where the maximum throughput of MQTT messages from nodes is determined. To simulate the message generation, we used a PC that generated MQTT messages, sent them to the edge device (raspberry pi), counted the number of messages it can process, and simultaneously saved them into the local Mongo database. We were sending different size packet messages and measuring the throughput performance based on its size. The experiment was repeated ten times per message packet to get the average number of message throughput. Furthermore, our edge device has to make some calculations regarding the most relevant frequencies from accelerometer data of Z, Y, Z-axis. We measured the execution time of the FFT transform-based based on the number of points collected. Nevertheless, we use ≈ 200 Hz sampling frequency for accelerometer data for appliance monitoring.

5.1 Data acquisition and pre-processing

With the proposed solution, we recorded the electricity consumption of common appliances using the current sensor. Some appliances were recorded during different states of functioning. For example, laptop energy consumption data was acquired while it was used for normal operations like writing text and browsing the internet. In a second session of data acquisition, we were running games on it. Similarly, we used a monitor normally on medium brightness, and the second recording was when the monitor was put on idle mode. The same has been done with the TV. The list of appliances is in table 5.1, the list of captured features is in table 5.2. Arduino Nano IoT 33 was communicating with Raspberry Pi using a serial communication via USB connection. Each appliance was recorded for one hour with a data acquisition frequency of 1Hz (1 sample per second). We were creating 3600 samples by the end of the session. Our sensor provided voltage, frequency, current, power and power factor.

In Figure 5.1 we compare power, current, and power factor of *HP pavilion, Asus rog GL503E normal usage, Asus rog Gl503E gaming*. We observed that when comparing HP and Asus computers using normal activities like browsing the internet and reading emails, HP consumes more energy and looking at

Type	Model	Release date	State	Session length	N^o of samples
Air purifier	Airfree P100	2019	ON	60 min	3600
Laptop	Asus rog GL503GE	2018	ON	60 min	3600
Coffee machine	Krups XN 7001	2012	ON	60 min	3600
Laptop	Laptop HP Pavilion	2012	ON	60 min	3600
Monitor	AOC27B1H	2019	ON	60 min	3600
Monitor	AOC27B1H	2019	SLEEP	60 min	3600
Shaver	Philips S3133	2020	CHARGING	60 min	3600
Smartphone	Huawei Mate 9	2016	CHARGING	60 min	3600
TV	Samsung UE40C6510UW	2010	ON	60 min	3600
Tablet	Apple iPad 10.2	2019	CHARGING	60 min	3600
Washing machine	Aqualtis AQ114D69D	2010	STANDBY	60 min	3600
Washing machine	Aqualtis AQ114D69D	2010	WASHING	60 min	3600
Total:					43 200

Table 5.1: Appliances that have been recorded using our prototype.

the power factor, the electronics are not as efficient as Asus computers. Although it makes it explainable, HP consumes more energy because it is older (released around 2012), and Asus was released around 2018. In general, hardware tends to be more energy-efficient. Furthermore, we recorded Asus computer while gaming, which requires more energy to support graphics card and processor for heavy work. It consumes way more than using it just for simple tasks (average three times more energy), and it also consumes much more than the HP computer.

Table 5.2: List of features captured with our prototype.

Feature	Symbol	Unit	Sensor precision
Real power	P	Watts	0.001
Power factor	pf	-	.01
rms Current	I	Ampere	0.001
Frequency	f	Hertz	0.1
rms Voltage	V	Volt	0.001
energy	Wh	watt-hour	1

Table 5.3: List of recorded washing machine cycles.

Washing cycle	N^o of sessions	Session length	N^o of samples
Prewashing	3	80 min	2400
Baby clothes	3	120 min	3700
Coloured	3	70 min	2100
Rinsing	3	35 min	1000
Mix	3	40 min	1200
7 day clothes	3	83 min	2500
Delicate clothes	3	60 min	1800
Total:			40 500

Furthermore, we have recorded a few washing machine cycles that we will try to classify. Table 5.3 presents the washing cycles that have been recorded. Each cycle was recorded three times, one for training the model, the second one for testing the model, and a third one for classifying to use as previously unseen data to the Machine Learning model. We monitored an *Aqualtis AQ114D69D* washing machine during seven different washing cycles. Each cycle was recorded three times.

In our further experiments, we are also using a third-party open-source appliance energy consumption database. ACS-F2 released in (2014) [140] is an appliance energy consumption pattern database. It provides individual electricity consumption data taken from hundreds of Swiss houses. Data-set comprise I, V, P, Q, f and ϕ, described in table 5.5. It consists of 15 different appliance categories and holding 15 different appliances per category, generating a dataset of 225 appliances in total. ACS-F2 appliance categories are described in table 5.4. Appliances were recorded by using PLOGG smart plug. Data was collect at a frequency of 10^{-1}Hz (1 sample per 10 seconds) rate over two sessions of 1 hour each, creating 360 records per appliance. The reason why we are paying attention to this database, is because we want to make an extension of this dataset with our new data. ACS-F2 had laid a strong ground of

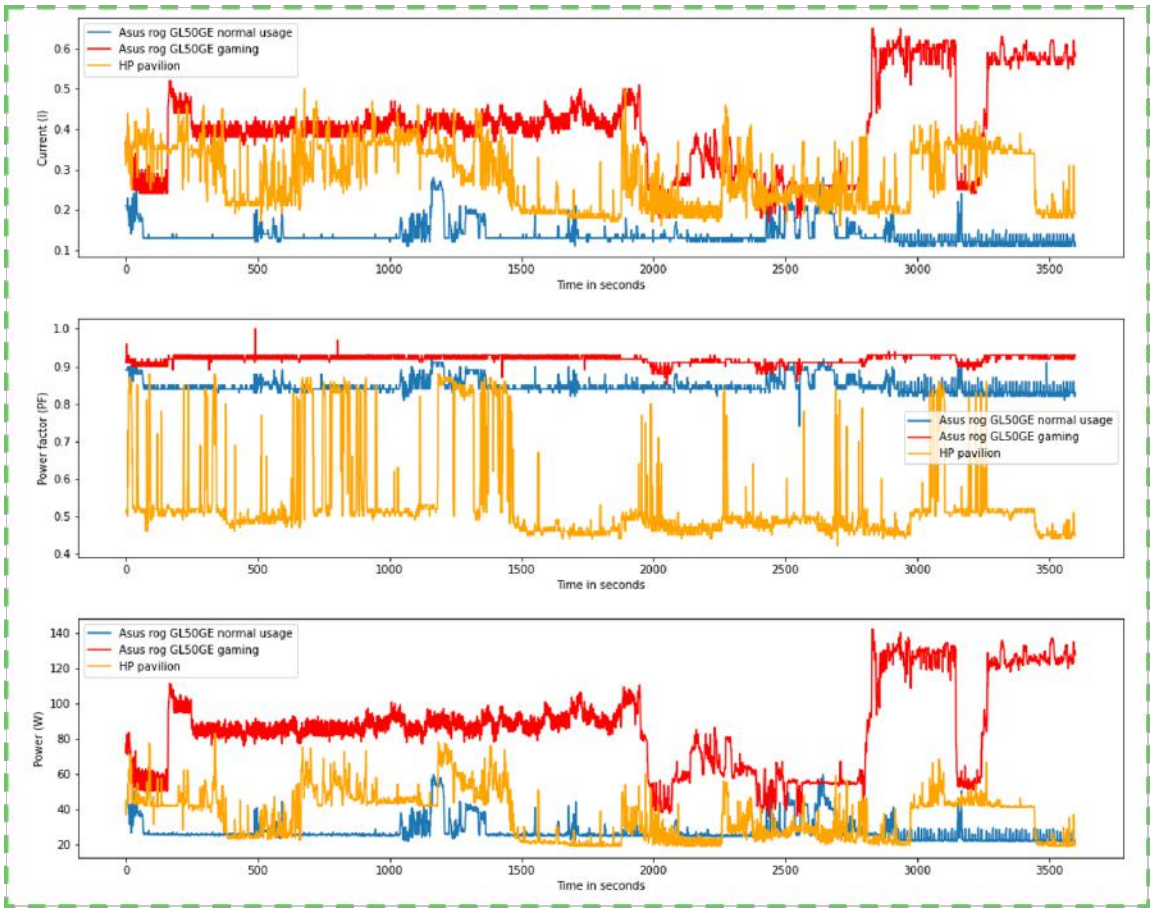


Figure 5.1: Power, Power factor, and current comparison of laptops.

different appliance data that we can use to recognize legacy appliances since the database was released in 2014. Moreover, it extends our experiments regarding machine learning with more appliance energy monitoring data.

Before using any machine learning classification, we have cleaned the dataset. We would not consider the record if it is $P \leq 0$ and $I \leq 0$. When these values are less than or equal to 0 (zero), it meant that at the moment of capturing a measurement appliance was consuming no energy (being idle or power saving), or the sensor was not sensitive enough to be able to capture the power and current traces.

In electricity data, we have some features that have high magnitude values in comparison with others. For example, power has a substantial magnitude comparing with the current. It is essential to scale these values to avoid this underlying assumption that larger values have a higher impact. We use a standard scaler from *sklearn.preprocessing* python library. The Standard Scaler assumes data is usually distributed within each feature and scales it in such a way that the distribution centered around 0, with a standard deviation of 1, a mathematical expression can be seen in Equation: 5.1.

$$\overline{x} = \frac{x - \mu}{\sigma} \tag{5.1}$$

Since the ACS-F2 dataset was released in 2014, we might be able to integrate this data to classify older appliances because appliance energy consumption traits are changing throughout the years. We would not be able to recognize legacy appliances without this data. We are planning to combine it with the new datasets generated by our solution. To do this task, we need to normalize these datasets and

Table 5.4: List of appliances that contained in ACS-F2 dataset.

Category	Appliances	Session	Records
Fridges/freezers	15	2	360
TVs (LCD)	15	2	360
Hi-Fi systems (with CD players)	15	2	360
Laptops	15	2	360
Computer stations (with monitors)	15	2	360
Compact fluorescent lamps (CFL)	15	2	360
Incandescent Lamps	15	2	360
Microwaves	15	2	360
Coffee machines	15	2	360
Mobile phones (via battery charger)	15	2	360
Printers	15	2	360
Fans	15	2	360
Shavers	15	2	360
Monitors	15	2	360
Kettles	15	2	360
Total:	225		162 000

Table 5.5: List of features of appliances in ACS-F2 dataset.

Feature	Symbol	Unit	Sensor precision
Real power	P	Watts	0.001
Reactive power	Q	VoltAmps	0.001
rms Current	I	Ampere	0.001
Frequency	f	Hertz	0.1
rms Voltage	V	Volt	0.001
phase angle	ϕ		1

create a standard feature set for both ACS-F2 and our new dataset to integrate it.

Our sensor automatically supplies us with power factor value, although the ACS-F2 dataset does not have it. Nevertheless, we can calculate the power factor in ACS-F2 by already known data. We can calculate the power factor by using the following formula:

$$pf = \frac{P}{S} \tag{5.2}$$

where pf means power factor, P is real power, and S is apparent power. ACS-F2 does not have an apparent power value. To calculate it, we can use the following formula:

$$S = \sqrt{P^2 + Q^2} \tag{5.3}$$

or

$$S = VI \tag{5.4}$$

where Q is reactive power.

ACS-F2 has some features that are not apparent in our current data set. However, we can feature

engineer it by using the power triangle used in physics. For example, our dataset does not contain apparent power, reactive power, or phase angle, but we can calculate it with the data we already have to enrich our feature set. As mentioned previously, we can calculate apparent power by using Equation 5.4. We already have current (I) and voltage (V). To calculate reactive power, we can use the following formula:

$$Q = \sqrt{S^2 - P^2} \tag{5.5}$$

to for phase angle calculation we can use following formula:

$$\phi = cos^{-1}(\frac{P}{S}) \tag{5.6}$$

We will be trying to do the classification using features that are common to both datasets. The feature set is in table 5.6, and we call it F2. That contains power (P), power factor (pf), current (I), frequency (f), and voltage (V).

Table 5.6: A list of feature sets that are common within ACS-F2 and our new dataset after feature engineering

Feature-set	**Containing features**
F2	$P, pf, I, f, V, Q, S, \phi$

Furthermore, when we merge databases, we need to normalize the data to avoid introducing bias in the data. To do this, we reduce our dataset to the size of ACS-F2 data. Each appliance has 720 records of energy observation inside ACS-F2. To normalize our dataset to these numbers, we reduce our dataset by filtering current data from table 5.1 by picking a record in the 5-second interval, meaning we will reduce from 3600 records to 720 records for the experiments.

5.2 Classification experiments

In this section, we discuss classification experiments that were done with our dataset and ACS-F2 dataset. In our experimentation, we use four different classification algorithms: KNN as a baseline for other classification algorithms, XGBoost, SVM, and RF.

One of the first, experiment A (EA), we have done was trying to classify appliances from the data that was obtained during data collection, and the recorded appliances are mentioned in table 5.1, the data was not under-sampled in this experiment. The experiment goal is to see how accurately we can classify appliances that were recorded using our monitoring prototype. For the classification we were using feature set that is mentioned in table 4.2 (F3). We have used KNN as the base classification algorithm following XGBoost, SVM, and RF classification.

ACS-F2 database contains multiple categories of appliances. We are going to experiment with first layer appliance classification as represented in Figure 3.7 in a hybrid-model approach. This experiment B (EB) aims to know the accuracy we can achieve of trying to classify an appliance type. All appliances per particular appliance type will get a type class. For instance, if we have multiple TVs, all of them will be classified as a TV. This will be done with the rest of the appliances.

Since the ACS-F2 database contains multiple appliance classes, we experimented with specific appliance classification based on that appliance type class. As described in table 5.4 each appliance type contains 15 different appliances, meaning that each ML model will be trained to classify specific appliances per appliance category. We present results based on different feature sets that we have defined.

It allows seeing the trends of accuracy based on the feature set. Feature sets are described in the table 4.2. We call this experiment C (EC). The classification results represent the second layer of model classifiers responsible for appliance brand/model classification, as represented in Figure 3.7 in a hybrid-model approach.

In the upcoming experiment that we name experiment D (ED), we take every appliance and classify them. When preparing ACS-F2 data for this experiment, we have noticed that some of the appliances do not have information about the appliance model, but only the type. Due to this reason, we do not include unknown devices in this experiment. Theoretically, from this dataset, we should have had 250 classes of different appliances. However, since some data is simply missing, we could do this experiment only with 198 appliances, meaning that the classification algorithm will classify that number of classes.

In the experiment with a name experiment E (EE), we measure the correctness of real-time classification. We are sending unseen appliance readings from our monitoring node into the ML model. For this experiment, we use a random forest trained model since the classification score was the highest. Furthermore, we will be sending unseen appliance current sensor readings to the classifier for 10 minutes period. The monitoring frequency of this experiment is 2 seconds, meaning that by the end of 10 minutes of an experiment, we will have 300 classification results for each investigated appliance.

In the experiment F (EF), we are joining two data sets: ACS-F2 and our data set that is described in Table 5.4 and Table 5.1 respectively, both data-sets share a common feature set F2 that is described in Table 4.2. First, we join two datasets based on the type of the device. For example, every laptop recorded in our dataset was treated as a class Laptop. This was done with all the appliances, and they had a label based on their type. Since ACS-F2 has a higher number of data and types of devices, we adapted our data set class names based on class names used in ACS-F2. For example, Smartphone was labeled as Mobile-phone. To understand better how the class names were changed, a Table 5.7 describes the conversion of class names before injecting it into the ACS-F2.

Some preparation has to be made in terms of data observations. Originally our data had 3600 samples per appliance. We reduced the sample size per appliance to 720 samples to avoid introducing bias as it was initially in the ACS-F2 dataset. Although some of the appliances types do not exist in the ACS-F2 database, due to this reason, we left the original sample size for Tablet, Washing-machine, and Air-purifier classes.

Furthermore, following this experiment, we perform appliance classification by model among other same type appliances, similarly to experiment EC, but this time we are injecting our data set into ACS-F2, this means that we will be adding additional appliances into to appliance categories: laptops, coffee machine, monitor, shaver, TV. Moreover, we will measure models accuracy with updated datasets.

Experiments regarding machine learning training and testing were done on a laptop. The laptop specifications are illustrated inside the Table 5.8.

In the fig. 5.2 we are showing the data samples per recorded appliance after the data set was cleaned for the following experiment. In experiment **EA** we used our dataset, which is described in Table 5.1, KNN performed 97% of accuracy, and it takes 1.5s to train. XGBoost gave us 99% and took 3.8s to train. The SVM accuracy was 99.8%, and the training took 11.3s, while the RF reported 99.99% of accuracy and took 1.5 s for training. Execution accuracy that is mentioned here is not including hyper-parameter tuning. We consider the training time when using already tuned parameters. From the results, we can say that the RF algorithm obtained the best accuracy. Furthermore, the worst was KNN. Nevertheless, its accuracy is great than 97%. In the illustration 5.3, we show a confusion matrix from appliance classification experiment using RF classification since it performed best in our results, and in Figure 5.4, we show a classification report. Features that were used to train models are described in table 4.2:F3 the

Table 5.7: Our data recorded data classes adapted for ACS-F2 dataset experiment

Our appliance name	Adapted class name to ACS-F2
Airpurifier-Airfree-P100	Airpurifier
Coffee-Krups-XN-7001	Coffee-machine
Laptop-Asus-rog-GL503GE	Laptop
Laptop-hp-pavilion-dv7	Laptop
Monitor-AOC27B1H	Monitor
Monitor-AOC27B1H-sleep	Monitor
Shaver-Philips-S3133	Shaver
Smartphone-Huawei-Mate-9	Mobile-phone
Tablet-Ipad-10.2	Tablet
TV-Samsung-UE40C6510UW	TV
Washing-machine-Aqualtis-AQ114D69D	Washing-machine
Washing-machine-Aqualtis-AQ114D69D-standby	Washing-machine

Table 5.8: Laptop specifications used in experiments.

Host	Model	CPU	RAM	OS
Laptop	Asus Rog GL553-GE	i7 8th gen 12 CPUs	16GB	Windows 10

original feature set provided by the current sensor from our monitoring node, and F2 updated feature set by using feature engineering.

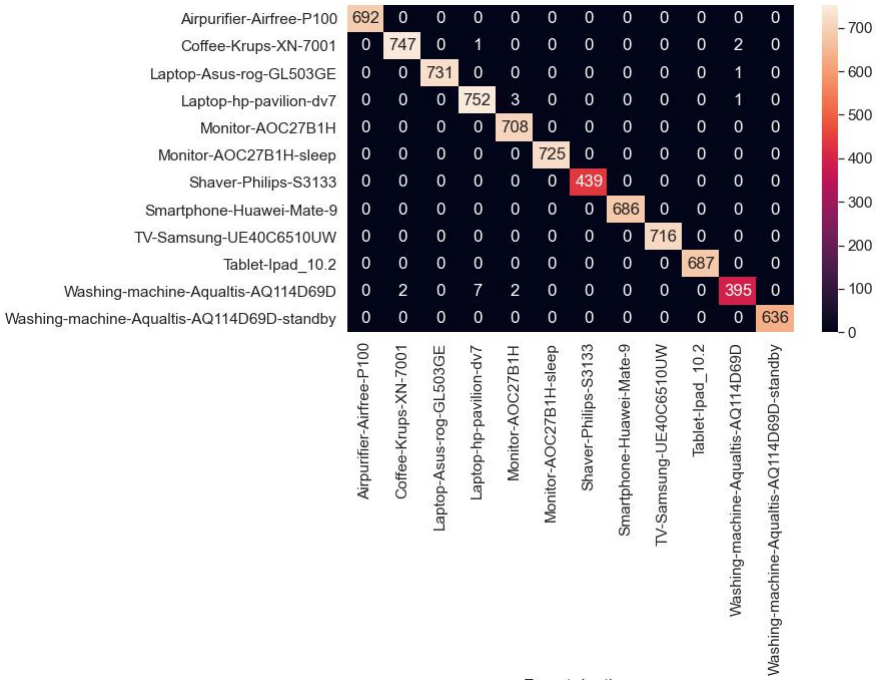


Figure 5.3: Random forest confusion matrix from the data used in fig. 5.2 using F3 feature set.

In this experiment, we show the comparison of classification performance using the original feature set called F3 that our used PZEM-004T current sensor provides an updated feature set with additional features called F2. In Figure 5.5 we compare the F3 (original from the current sensor feature set) with

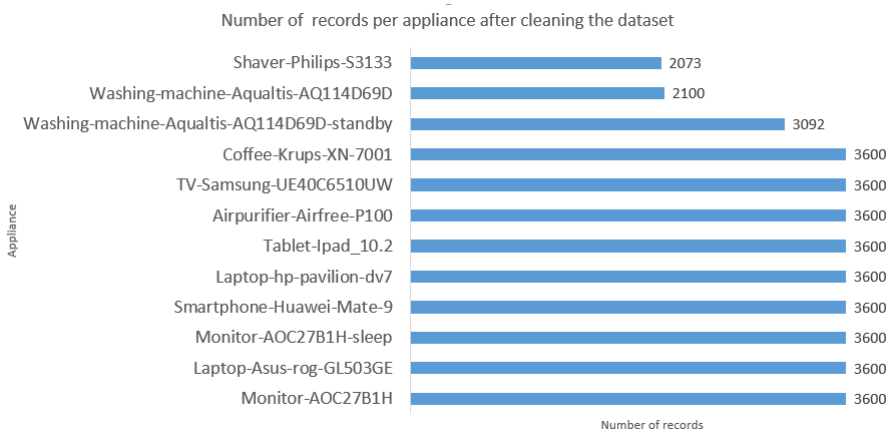


Figure 5.2: Sample size per appliance after cleaning the dataset. Source: author.

	precision	recall	f1-score	support
Airpurifier-Airfree-P100	1.00	1.00	1.00	692
Coffee-Krups-XN-7001	1.00	1.00	1.00	750
Laptop-Asus-rog-GL503GE	1.00	1.00	1.00	732
Laptop-hp-pavilion-dv7	0.99	0.99	0.99	756
Monitor-AOC27B1H	0.99	1.00	1.00	708
Monitor-AOC27B1H-sleep	1.00	1.00	1.00	725
Shaver-Philips-S3133	1.00	1.00	1.00	439
Smartphone-Huawei-Mate-9	1.00	1.00	1.00	686
TV-Samsung-UE40C6510UW	1.00	1.00	1.00	716
Tablet-Ipad_10.2	1.00	1.00	1.00	687
Washing-machine-Aqualtis-AQ114D69D	0.99	0.97	0.98	406
Washing-machine-Aqualtis-AQ114D69D-standby	1.00	1.00	1.00	636
accuracy			1.00	7933
macro avg	1.00	1.00	1.00	7933
weighted avg	1.00	1.00	1.00	7933

Figure 5.4: Random forest classification report from the data used in fig. 5.2.

the updated feature set F2 with feature engineering techniques. Moreover, from the graphic, we have obtained some interesting results. KNN was improved from 96% up to 99%. Furthermore, we can see that XGB had a slight improvement as well, from 99.3% to 99.7%. Moreover, we noticed a negligible degradation in SVM results with the original data set. It was reporting 99.8%, but with the updated feature set, it falls to 99.2%. The random forest does not seem to have either improvement or degradation in the accuracy report of 99.9%.

In the following Figure 5.6 we are showing the result of classification **EB** experiment, where appliances are being classified by appliance type. From the results, we can claim that the F2 feature set with some engineered features gave us a small margin of increased classification accuracy. From the results, we can claim that the RF classification algorithm is in the lead with provided accuracy of 99.3%.

The following results are from experiment **EC** where we classify specific appliances in the same belonging group of appliances. In figure 5.7 we show results of different classification algorithms using the F1 feature set and in Figure 5.8 we show results when using F2 feature sets. Furthermore, in Figure 5.9 we present the average performance of classification algorithms that we obtained in the experiment throughout F1 and F2. We can claim that the F2 feature set increases the average accuracy of appliance model identification among the same type of appliances.

In the experiment, called **ED**, where we are classifying each appliance among existing appliances using data from the ACS-F2 dataset. This means that a single ML model would be used to classify any

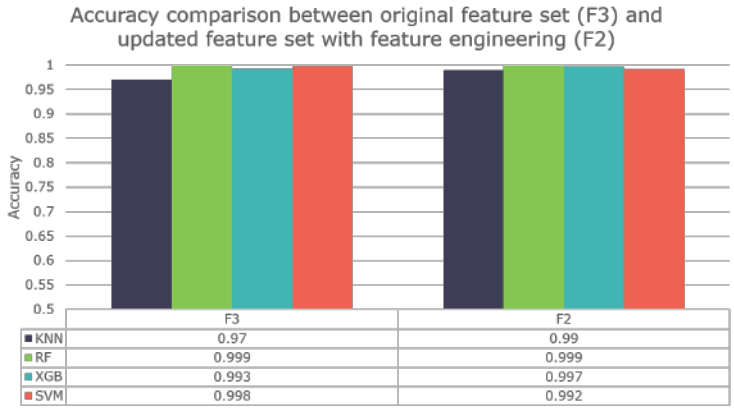


Figure 5.5: Comparison between F3 original feature set and update feature set with feature engineering.

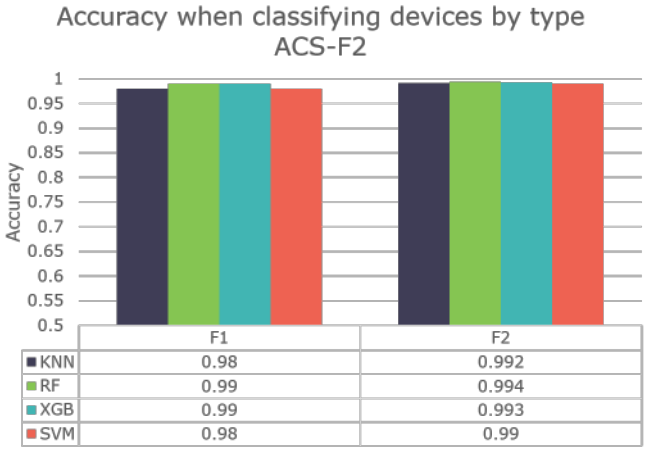


Figure 5.6: Comparison of classification accuracy in ACS-F2, when classifying by device type.

appliance. Results of using different classification algorithms are illustrated in Figure 5.10. Although the classification result has a decent score, we must not be deceived by the accuracy. Let us investigate this experiment further, regardless of good average accuracy. Some of the appliances are poorly classified, and the accuracy based on RF F2 experiment from Figure 5.10, classification report shows that there are appliances classified as poorly as 59% accuracy rate. Although it is not a majority of appliances with a high rate of fallacious classification, we can not claim that having one model for all appliances is suitable for high accuracy appliance classification.

In the experiment **EE**, we were classifying recorded appliances in real-time. We send appliance data to the classification module for 10 minutes with a sampling frequency of 2 seconds. The classification results are illustrated in Figure 5.11.

	Coffee_machines	Computers_stations_(with_monitors)	Fans	Fridges_and_freezers	Hi-Fi_systems_(with_CD_players)	Kettles	Lamps_(compact_fluorescent)	Lamps_(incandescent)	Laptops_(via_chargers)	Microwave_ovens	Mobile_phones_(via_chargers)	Monitors	Printers	Shavers_(via_chargers)	Televisions_(LCD_or_LED)
KNN	0.94	1	1	0.99	0.99	0.89	0.98	1	0.98	0.98	0.96	0.99	0.98	0.98	0.99
RF	0.97	1	0.99	0.99	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
XGBoost	0.97	0.99	0.99	1	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
SVM	0.96	1	0.99	0.99	0.98	0.99	1	0.98	0.98	1	0.99	0.99	0.99	0.96	0.99

Figure 5.7: Comparison of appliance classification per category using F1 feature set.

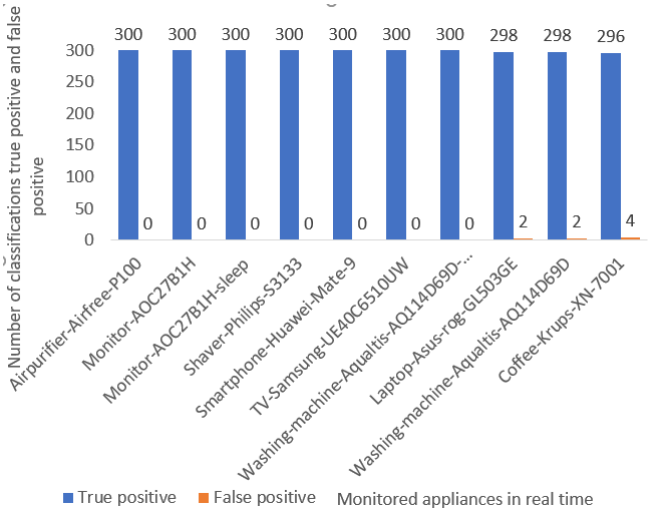
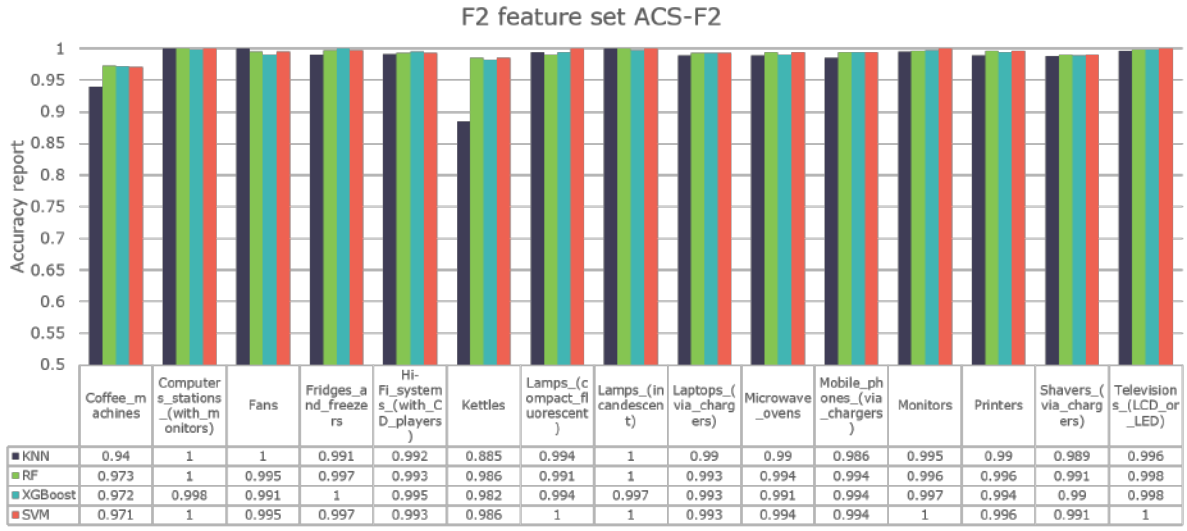


Figure 5.11: Real time classification evaluation results.

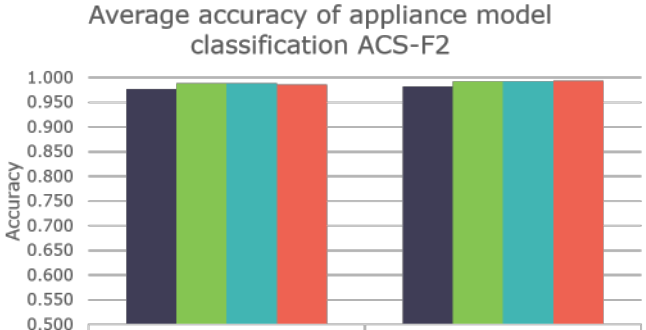
From the obtained results, most of the appliances had a 100% accurate classification. When recording the Asus Rog GL503GE laptop computer, we had two false-positive evaluations in which it was classified as an HP Pavilion dv7 laptop computer. During 10 minute monitoring period, the washing machine Aqualtis AQ114D69D was mistaken with Coffee machine Krups XN 7001, and the opposite happened where the coffee machine was mistakenly classified as the washing machine Aqualtis AQ114D69D.9. In this experiment, the obtained classification score is 99.73% as true positives and 0.26% as identified as false positives. During the experiment, we noticed that the monitored washing machine is being recognized in a washing state regardless of the washing cycle selected. The trained model has been trained only on one cycle, but it treats other cycles in the same classification as simply "washing." We were expecting that the washing will fail to be classified as washing using different cycles than the one it was trained on. This means that the washing cycles are almost identical in terms of data clustering.

When we turned on the washing machine, our classifier can report that the state has been changed of the appliance from IDLE to washing, and when the cycle finished, it goes back to being IDLE, which



	Coffee_machines	Computers_stations_(with_monitors)	Fans	Fridges_and_freezers	Hi-Fi_systems_(with_CD_players)	Kettles	Lamps_(compact_fluorescent)	Lamps_(incandescent)	Laptops_(via_chargers)	Microwave_ovens	Mobile_phones_(via_chargers)	Monitors	Printers	Shavers_(via_chargers)	Televisions_(LCD_or_LED)
KNN	0.94	1	1	0.991	0.992	0.885	0.994	1	0.99	0.99	0.986	0.995	0.99	0.989	0.996
RF	0.973	1	0.995	0.997	0.993	0.986	0.991	1	0.993	0.994	0.994	0.996	0.996	0.991	0.998
XGBoost	0.972	0.998	0.991	1	0.995	0.982	0.994	0.997	0.993	0.991	0.994	0.997	0.994	0.99	0.998
SVM	0.971	1	0.995	0.997	0.993	0.986	1	1	0.993	0.994	0.994	1	0.996	0.991	1

Figure 5.8: Comparison of appliance classification per category using F2 feature sets.



	F1	F2
KNN	0.977	0.983
RF	0.989	0.993
XGB	0.989	0.992
SVM	0.986	0.994

Figure 5.9: Comparison of appliance averaged classification results of EC experiment.

means that we can use the change of state as our timestamp of the beginning of the cycle and we can observe when the cycle has been finished. Having this information, we can pass washing cycle data to a dedicated classifier. We discuss the way we can classify different cycles in the following section.

Here we present the results of the EF experiment, where we merge two appliance monitoring data sets: ACS-F2 and our dataset. Figure 5.12 shows a confusion matrix using a random forest algorithm that provided us with a 99.31% accuracy report. In this experiment, KNN reported an accuracy of 99.1%, SVM reported 99%, and XGB reported 99.1%. Furthermore, similarly to merging of appliances per their type, we merged datasets per appliance model among other same type models. In Figure 5.13, we show the classification results when both data sets are merged together. Based on average results, the best classification was reported by random forest with 99.33%. Since we have only one washing machine in the experiment, all models performed very well classifying if a washing machine is on IDLE or washing states.

In the last experiment, we are comparing the real-time classification accuracy when both data sets ACS-F2, and our data sets are merged. Here we are going to test two approaches in the first one, where

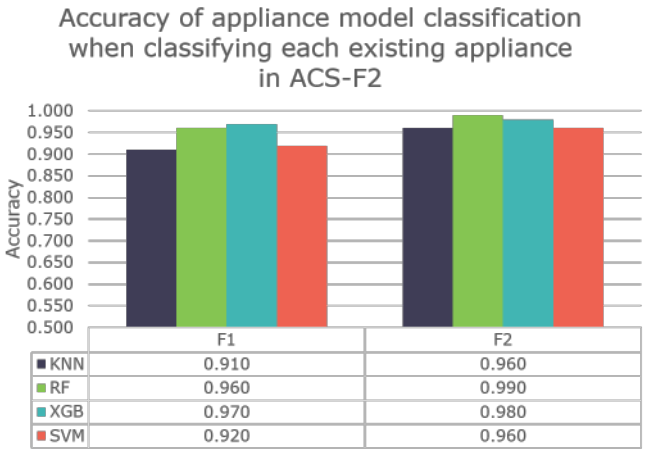


Figure 5.10: Appliance classification when each appliance has its own class in ACS-F2 database.

we have a single trained model for all existing appliances and a hybrid model approach where we have multiple models that are split in a tree approach. The first layer model is classifying the type of appliance and forward the data to the appropriate model to classify the appliance model based on its type. To test a single model for real-time classification with merged data sets, we chose random forest because, based on our experiments, it performed best with a classification result of 99.3%. KNN reported 99.1% accuracy, SVM reported 99% accuracy, and XGB reported 99%. The single model contains for all appliances contains 198 different appliance classes from the ACS-F2 data set plus 12 new appliance classes that we have introduced from our data set containing in total 210 classes. The experimental approach is illustrated in Figure 5.14, appliance classification based on two different approaches, A - single-model approach results, B - hybrid model approach results. Each appliance was monitored for 10 minutes with a sampling frequency of 2 seconds. By the end of this time, we have 300 classification results per monitored appliance, and we count true positives and false positives.

Monitoring results are illustrated in Figure 5.15, from the obtained results, we can tell that the hybrid model approach has a higher advantage providing us with more accurate real-time classification in comparison with the single model classification approach, which has increased the number of false-positive classifications in comparison with the hybrid model approach. Based on this experiment, the classification accuracy of a single model was 52.2%, and the accuracy provided by the hybrid model approach provided us with 96.4%.

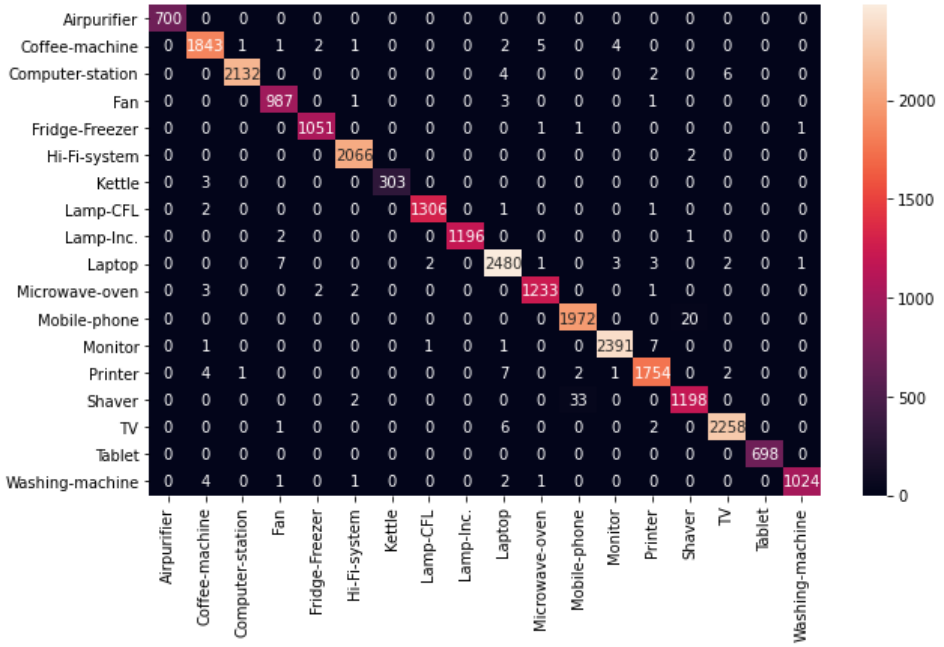


Figure 5.12: Confusion matrix of appliance type classification when ACS-F2 and our dataset are merged using common feature set called F2.

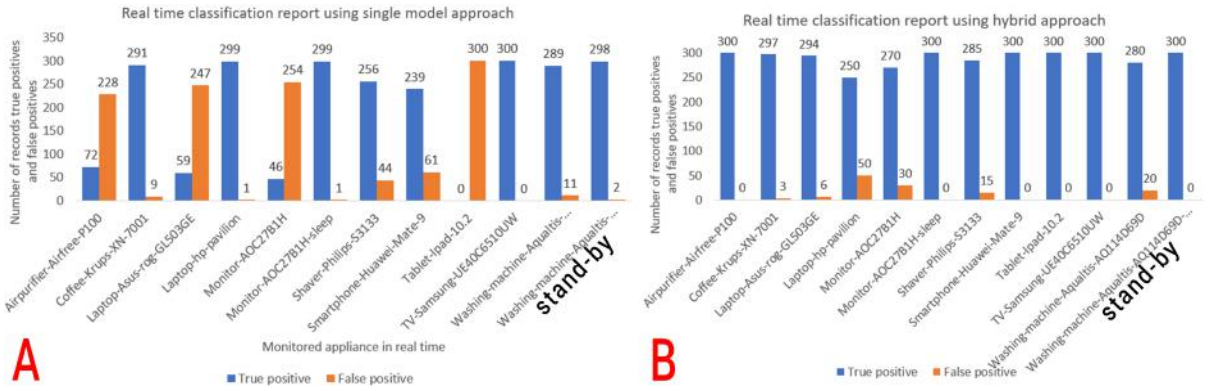


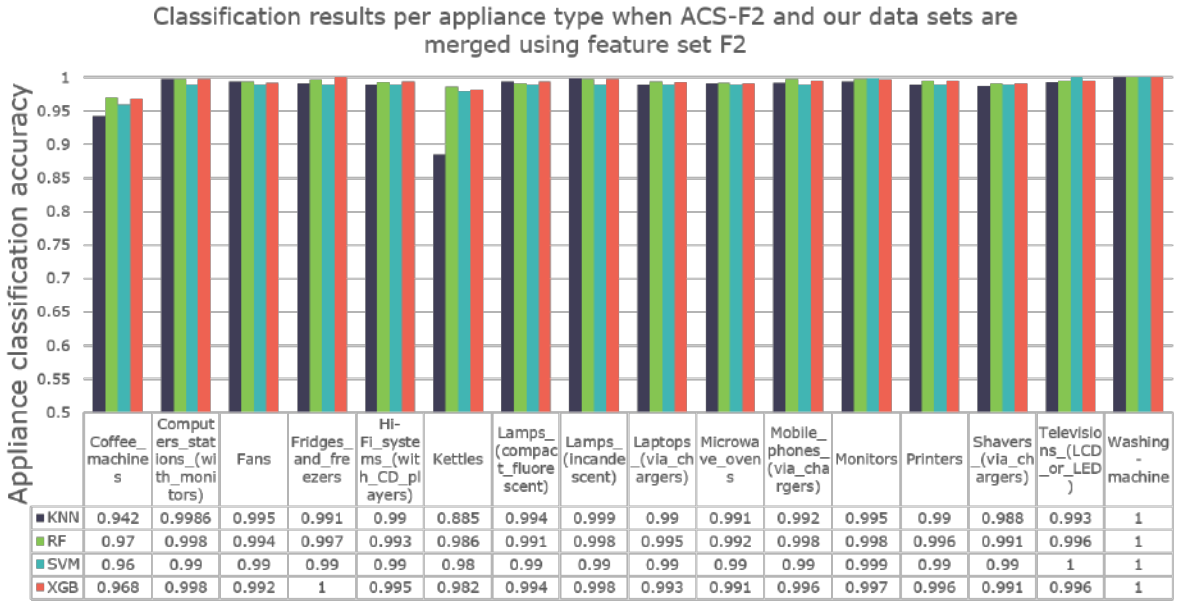
Figure 5.15: Classification results. A - single model classifying all existing classes, B - hybrid model that splits the classification.

5.3 Multi-functional appliance cycle classification

In this experiment, we monitored an *Aqualtis AQ114D69D* washing machine during different washing cycles. Cycles that were recorded and used in this experiment are mentioned in table 5.3.

The first thing we have tried was using the same approach as appliance classification, labeling the data, and classifying it. This approach was not able to classify different cycles, meaning that the washing cycle signal amplitudes are very similar and can not be clustered for differentiation. In Figure 5.16 we plot electricity current (A) readings of two washing cycles from our sensor to have an idea of how different cycles look. Looking at the plot, we can tell that the amplitudes are very similar due to this reason. We can not use a traditional classification approach.

Every program cycle produces a slightly different energy consumption pattern during its functioning



	Coffee_machines	Computers_stations_(with_monitors)	Fans	Fridges_and_freezers	Hi-Fi_systems_(with_CD_players)	Kettles	Lamps_(compact_fluorescent)	Lamps_(Incandescent)	Laptops_(via_chargers)	Microwave_ovens	Mobile_phones_(via_chargers)	Monitors	Printers	Shavers_(via_chargers)	Televisions_(LCD_or_LED)	Washing-machine
KNN	0.942	0.9986	0.995	0.991	0.99	0.885	0.994	0.999	0.99	0.991	0.992	0.995	0.99	0.988	0.993	1
RF	0.97	0.998	0.994	0.997	0.993	0.986	0.991	0.998	0.995	0.992	0.998	0.998	0.996	0.991	0.996	1
SVM	0.96	0.99	0.99	0.99	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.999	0.99	0.99	1	1
XGB	0.968	0.998	0.992	1	0.995	0.982	0.994	0.998	0.993	0.991	0.996	0.997	0.996	0.991	0.996	1

Figure 5.13: Classification results when both data-sets are merged and classification is done by appliance model among same type appliances.

cycle, meaning that we have to treat this data as a time series observation. Unfortunately, due to the nature of time-series data, we cannot detect the washing cycle program in real-time. The classification can be made only after half of the washing cycle or at the end of the cycle. Our approach was to divide a washing cycle in half. We have a water heating pattern on the left side signal window, and on the right side window, we have an actual washing pattern and perform feature extraction from a time-series signal. To extract multiple features, we have used a TSFEL python feature extraction library [141], which in our case extracted around 325 different features per one time-series signal feature. For feature extraction, we have used all our time series features from Table 5.2. It uses time-series signals and extracts temporal, statistical, and spectral features from a time-series signal. One time series window creates one row with features. In our case, we divide the signal into two parts, meaning that one washing cycle provides us with two rows of features per class. In total, we had 1951 extracted features per single window.

We have extracted features from 7 different washing cycles. Each cycle has been recorded three times. One cycle is used for training, the second one is for testing for the model, and the third is for classification testing to prove that this method works. Washing cycles are described in table 5.3

Our experiment used a KNN algorithm to classify by extracted features from the washing cycle based on recorded patterns. We ran the algorithm with a hyper-parameter K = 1. We used all the features that the TSFEL library provided. From obtained results, we can demonstrate that our approach of washing cycle classification works as planned. A confusion matrix is illustrated in figure 5.17. In the future, we will perform automated feature selection to reduce the number of features that possess the least importance.

To test the model with unseen washing cycle data, first, we need to perform feature extraction from the signal and supply extracted features to the ML model. The model was able to recognize a new unseen washing cycle and successfully classify it from our testing.

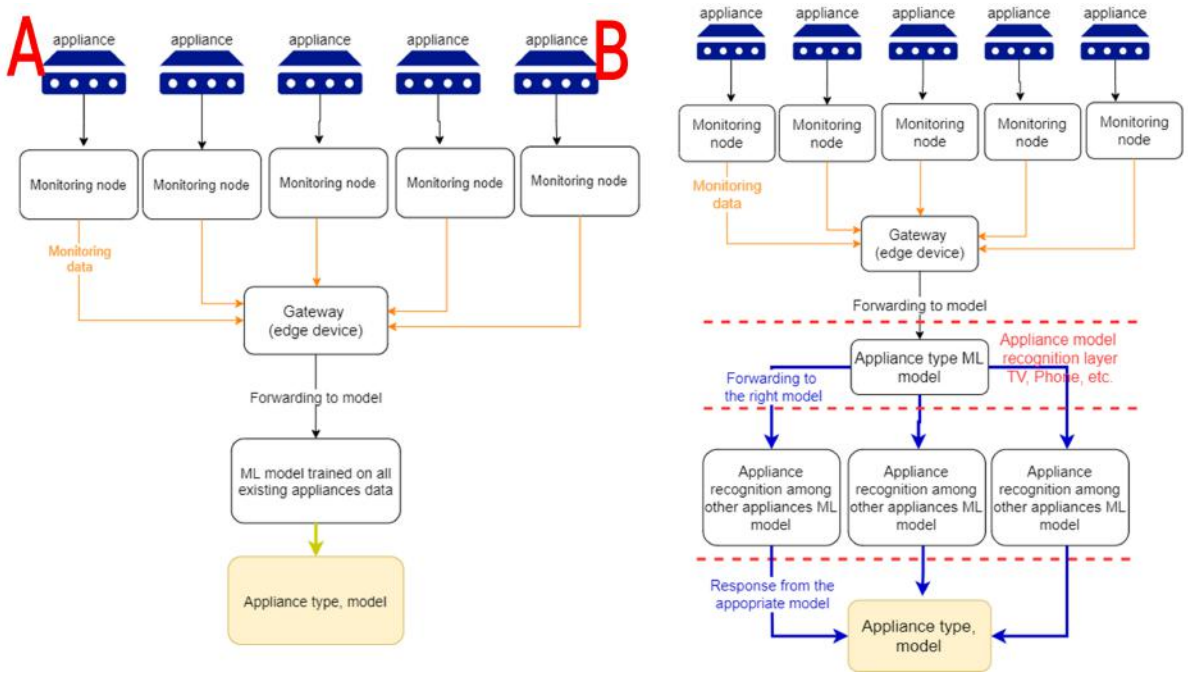


Figure 5.14: Classification approaches. A - single model classifying all existing classes, B - hybrid model that splits the classification.

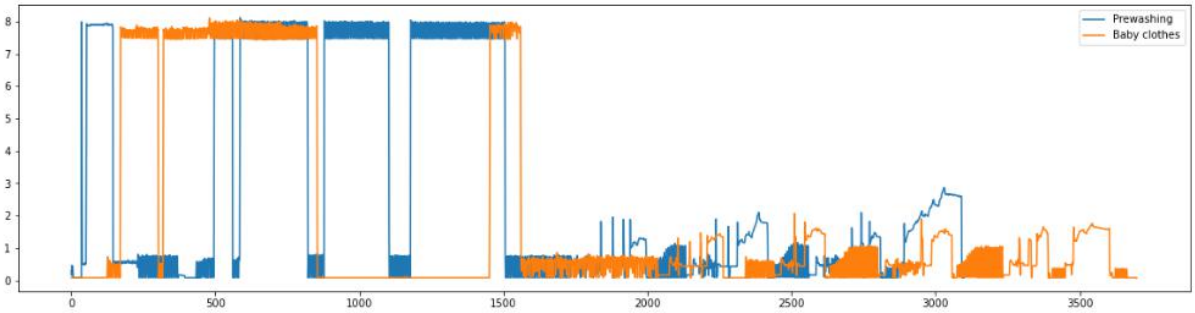


Figure 5.16: Two washing cycles. Blue line - prewashing, orange line - baby clothes.

5.4 System performance evaluation

In figure 5.18 (A), we can see that message size is influencing the maximum throughput. The maximum number of MQTT messages that Raspberry Pi could handle was 35000 per minute when the message size is 378 bytes, meaning that we could send 13.23MB per minute via MQTT. Interestingly, when the message size was reduced to 103 bytes, we could send 46000 messages, although the number of messages is more extensive. If we convert it to MB, it makes 4.7MB. If we look at the CPU load in figure 5.18 (B), during the experiment, the MQTT broker uses 15.45% of the total CPU load and other services are consuming the rest of the CPU.

Message size from nodes can vary depending on the device we are monitoring. For example, if we obtain data from TV, the single message size is 161 bytes that will be sent once per 2 seconds, which means that we need only 30 messages per minute. If we take an example of washing machines monitoring, message size is more significant because it has more sensors connected to it and the message size is 378 bytes per 2 seconds. Furthermore, when the washing machine is in use, we use an accelerometer for

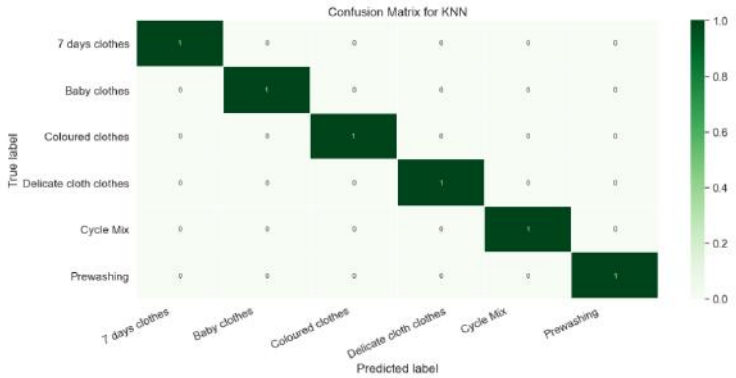


Figure 5.17: Feature extraction pipeline using TSFEL library of a single washing cycle.

oscillation monitoring, which sends 200 messages per second with a payload size of 103 bytes.

Based on this data, we can define how many devices we can monitor at the same time. If we consider a household with a washing machine and a dishwasher in use simultaneously, we would need to send 24000 messages per minute (due to high volume accelerometer data), which still leaves us with available 11000 messages per minute. Meaning that theoretically, we could monitor around 360 appliances using a single Edge device.

We calculated the overhead of execution time that is required by the FFT algorithm. If we acquire 200 sensor readings of the X, Y, Z-axis, we need to calculate 600 data points where 200 is for each axis. By our estimation, the overhead to convert the signal from the time domain to frequency domain is 4.3ms for 600 points, as illustrated in figure 5.18 (C).

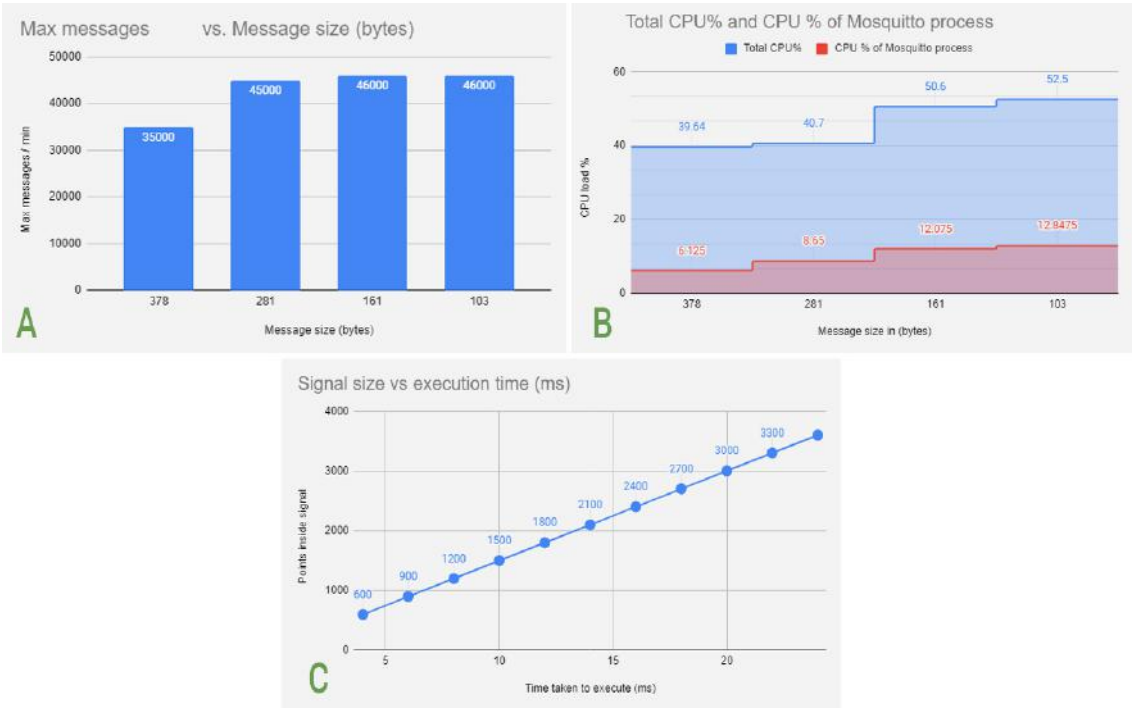


Figure 5.18: Prototype evaluation. A) Maximum message throughput based on message size via MQTT. B) SBC CPU load while executing maximum message throughput experiment. C) Execution time of FFT based on signal size.

Chapter 6

Conclusion

Summary

While the level of comfort and lifestyle quality is growing, energy consumption is increasing in parallel. Soon enough, energy consumption growth will become a massive problem in Europe and all the globe if we keep exploiting fossil fuels like coal, oil, or natural gasses. Rising energy consumption levels became problematic, and Europe is sharing quite a big part of it. Growing CO_2 gas emissions that provoking earth with greenhouse effect are reaching 36%, sharing more than one quarter with all the world. Moreover, Europe represents 40% of final energy consumption. Although statistics might sound frightening, we can build a bright future if we act now. Europe has created a goal of reducing the CO_2 emissions and exploiting renewable energy sources, and its objective is to share at least 32.5% of renewable energy sources by 2030.

Every bit helps in reducing the energy. Although many buildings in Europe are not constructed to be energy efficient, we can not simply demolish them and build new ones because it is simply unsustainable and not environmentally friendly. Appliances are sharing quite a big part in overall building energy consumption. To estimate the energy consumption EU introduced energy efficiency by labeling appliances with energy efficiency ranking, meaning that consumers can purchase more energy-efficient devices. Nevertheless, it is not enough to measure the real-life energy consumption per apartment or per building of many apartments. A solution is to use our existing communication infrastructure and implement an IoT solution for appliance monitoring and resource monitoring.

Users receiving a detailed energy consumption will become more aware of how they use the appliances and the various cycles of appliances. This might help in increased energy savings. Furthermore, the solution will help draw a realistic picture of how much energy is consumed by domestic appliances during day-to-day life. Energy consumption awareness must be spread, and consumers should be informed about the resources consumed by the surrounding devices, not just an overall monthly bill.

Generally, there are two types of appliance monitoring in the literature: non-intrusive load monitoring NILM and intrusive load monitoring ILM. We are proposing a flexible appliance monitoring system prototype that is following the ILM approach. However, this approach requires to use of a sensor for every appliance. Nowadays, it is a viable option due to low-cost but highly accurate sensors. Our implemented solution provides two types of monitoring: simple appliances and complex ones. In simple appliances, we monitor only energy consumption using a current sensor that provides us with various energy readings about the appliance. Furthermore, a complex appliance monitoring solution, in this category, we have appliances such as a washing machine or a dishwasher, which requires electricity and water supply. In a complex appliance, with the help of various sensors, we can monitor water intake

and water drain, the water temperature of the intake and exhaust hoses, and oscillations because of the machinery inside that is spinning tub of the washing machine or spinning arms inside the dishwasher, we can monitor the quantity of water that has been consumed during the washing cycle. Furthermore, the system can be used with legacy and new appliances.

Reached goals

We successfully have implemented a low-cost, highly flexible appliance monitoring platform prototype that can simultaneously monitor multiple appliances using multiple nodes. Based on our experiments, we can monitor more than 360 unique appliances using a single gateway, including high volume data-producing nodes for complex appliances like a washing machine or a dishwasher. Our method of setting up and registering a new node simplifies the setup for new appliance monitoring. We can connect a new appliance and monitor it using any device with built-in WiFi and an internet browser. Using obtained data, we can classify monitored appliances, including devices functioning in multiple states, with excellent accuracy using the Random Forest classification.

Two classification approaches have been tested: a single model approach where the same machine learning model classifies any existing appliance. Our implemented hybrid model uses multiple ML models and splits problems into smaller ones obtaining a higher accuracy compared to the single model approach. Since our recorded appliance dataset was relatively small, we have complemented the appliance classification experiments using the ACS-F2 publicly available appliance monitoring data set.

From the experiments, we can claim that the multiple ML model approach is more advanced and provides better classification results than the single model classification. From the experiment results, we could see that appliance classification is highly accurate using a single model only when the number of classes is low. However, when we considering hundred of different devices single model becomes an ill-suited solution. In testing and training, the random forest classification algorithm was performing best in most use cases. Due to this reason, we used this model in our real-time classification experiment. The single model provided us with only 52.2% classification accuracy, and the hybrid-model approach provided us with 96.4%. The false-positive results were due to miss-classification in the first layer model where we classify appliance type. The next layer, where a model is trained for classifying appliance models among the same type of appliances, did not return false positives.

Future work

- Prototype security measures:

  In this work, we have not reviewed security measures that could have been implemented. In order to implement our solution into production and to integrate it with the SATO platform, security implementation must be further explored. A possible solution is SSL certificates integration for data protection and confidentiality.

- Prototype new monitoring node setup:

  For setting up a new monitoring node, an application should be developed to make the appliance registration and setup is rapid and straightforward. However, the current solution for setting up a new node is quick and relatively simple to perform.

- Fault detection and prevention:

Due to the pandemic, we did not have an opportunity to fully explore the patterns that implementation would have helped us to collect and experiment with. In the future, we do plan to implement a fault detection module. One of the proposed solutions is a further exploration of auto-encoders and pattern recognition for fault detection techniques.

- Monitoring node independence:

  In this prototype, we propose a solution of monitoring system prototype that consists of a monitoring node and edge device. At this moment, we are successfully exploiting the edge device and its computing power to generate necessary feature extraction like FFT transform from time-series data to the frequency domain. To create fully independent monitoring nodes from the edge device solution is to transfer computations from the edge into monitoring nodes themselves. More specifically, computations modules would make a part of MCUs firmware. Solutions are integrating more potent MCUs that have higher computational power. One of many candidates could be the ESP8266 board containing a dual-core microprocessor and has WiFi capabilities.

- Appliance ML model improvements:

  To constantly improve and update machine learning models, we need more data. Due to this reason, we will be using data from the SATO platform ontology when it is available to us.

Bibliography

[1] DA Pohoryles, C Maduta, DA Bournas, and LA Kouris. Energy performance of existing residential buildings in europe: A novel approach combining energy with seismic retrofitting. *Energy and Buildings*, 223:110024, 2020.

[2] European Commission. Fourth report on the state of the energy union. `https://ec.europa.eu/info/publications/4th-state-energy-union_en`. Accessed: 2020.09.02.

[3] Anna Carolina Menezes, Andrew Cripps, Dino Bouchlaghem, and Richard Buswell. Predicted vs. actual energy performance of non-domestic buildings: Using post-occupancy evaluation data to reduce the performance gap. *Applied energy*, 97:355–364, 2012.

[4] Vincenzo Corrado and Ilaria Ballarini. Refurbishment trends of the residential building stock: Analysis of a regional pilot case in italy. *Energy and Buildings*, 132:91–106, 2016.

[5] Graziano Salvalai, Marta Maria Sesana, and Giuliana Iannaccone. Deep renovation of multi-storey multi-owner existing residential buildings: A pilot case study in italy. *Energy and Buildings*, 148:23–36, 2017.

[6] European commission. About the energy label and ecodesign. `https://ec.europa.eu/info/energy-climate-change-environment/standards-tools-and-labels/products-labelling-rules-and-requirements/energy-label-and-ecodesign/about_en`. Accessed: 2020.09.04.

[7] Alessandro Biglia, Andrew J Gemmell, Helen J Foster, and Judith A Evans. Energy performance of domestic cold appliances in laboratory and home environments. *Energy*, 204:117932, 2020.

[8] J Palmer, N Terry, and T Kane. Further analysis of the household electricity survey. *Early Findings: Demand Side Management*, 2013.

[9] Georgina Wood and Marcus Newborough. Energy-use information transfer for intelligent homes: Enabling energy conservation with central and local displays. *Energy and buildings*, 39(4):495–503, 2007.

[10] Pervez Hameed Shaikh, Nursyarizal Bin Mohd Nor, Perumal Nallagownden, Irraivan Elamvazuthi, and Taib Ibrahim. A review on optimized control systems for building energy and comfort management of smart sustainable buildings. *Renewable and Sustainable Energy Reviews*, 34:409–429, 2014.

[11] Daniela Pasini, Francesco Reda, and Tarja Häkkinen. User engaging practices for energy saving in buildings: Critical review and new enhanced procedure. *Energy and Buildings*, 148:74–88, 2017.

[12] A Longjun Wang, B Xiaomin Chen, C Gang Wang, and D Hua. Non-intrusive load monitoring algorithm based on features of v–i trajectory. *Electric Power Systems Research*, 157:134–144, 2018.

[13] Bing Zhang, Shengjie Zhao, Qingjiang Shi, and Rongqing Zhang. Low-rate non-intrusive appliance load monitoring based on graph signal processing. In *2019 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pages 11–16. IEEE, 2019.

[14] Yassine Himeur, Abullah Alsalemi, Faycal Bensaali, and Abbes Amira. Efficient multi-descriptor fusion for non-intrusive appliance recognition. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[15] Antonio Ridi, Christophe Gisler, and Jean Hennebert. User interaction event detection in the context of appliance monitoring. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 323–328. IEEE, 2015.

[16] I Abubakar, SN Khalid, MW Mustafa, Hussain Shareef, and M Mustapha. Application of load monitoring in appliances' energy management–a review. *Renewable and Sustainable Energy Reviews*, 67:235–245, 2017.

[17] Sam Moayedi, Fares AlJuheshi, Ahmad Almaghrebi, Jan Haase, Hiroaki Nishi, Kim Fung Tsang, and Mahmoud Alahmad. An overview of technologies for lower energy consumption in smart buildings. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 4693–4698. IEEE, 2018.

[18] Antonio Ridi, Christophe Gisler, and Jean Hennebert. A survey on intrusive load monitoring for appliance recognition. In *2014 22nd international conference on pattern recognition*, pages 3702–3707. IEEE, 2014.

[19] Ming-Tang Chen and Che-Min Lin. Standby power management of a smart home appliance by using energy saving system with active loading feature identification. *IEEE Transactions on Consumer Electronics*, 65(1):11–17, 2018.

[20] Jie Yuan and Xiaoyong Li. A reliable and lightweight trust computing mechanism for iot edge devices based on multi-source feedback information fusion. *Ieee Access*, 6:23626–23638, 2018.

[21] Jiheon Kang and Doo-Seop Eom. Offloading and transmission strategies for iot edge devices and networks. *Sensors*, 19(4):835, 2019.

[22] Wafa'a Kassab and Khalid A Darabkh. A–z survey of internet of things: Architectures, protocols, applications, recent advances, future directions and recommendations. *Journal of Network and Computer Applications*, 163:102663, 2020.

[23] Jacob Fraden. *Handbook of Modern Sensors. Physics, Designs, and Applications*, chapter 1. Springer, 2015.

[24] Cristian González García, Daniel Meana-Llorián, Juan Manuel Cueva Lovelle, et al. A review about smart objects, sensors, and actuators. *International Journal of Interactive Multimedia & Artificial Intelligence*, 4(3), 2017.

[25] Ruengwit Khwanrit, Somsak Kittipiyakul, Jasada Kudtonagngam, and Hideaki Fujita. Accuracy comparison of present low-cost current sensors for building energy monitoring. In *2018 International Conference on Embedded Systems and Intelligent Technology & International Conference on Information and Communication Technology for Embedded Systems (ICESIT-ICICTES)*, pages 1–6. IEEE, 2018.

[26] Allegro MicroSystems Inc. Fully integrated, hall effect-based linear current sensor with 2.1 kvrms voltage isolation and a low-resistance current conductor. https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf, 2007. Accessed: 2020.11.05.

[27] Winson. Wcs1800. hall effect base linear current sensor. http://www.winson.com.tw/uploads/images/WCS1800.pdf, 2020. Accessed: 2020.10.13.

[28] Albert Young. Non-invasive sensor: Yhdc sct013-000 ct used with arduino. (sct-013). https://www.poweruc.pl/blogs/news/non-invasive-sensor-yhdc-sct013-000-ct-used-with-arduino-sct-013. Accessed: 2020.10.13.

[29] RUENGWIT KHWANRIT. Accuracy comparison and auto-calibration algorithm of present low-cost current sensors for building energy monitoring. http://ethesisarchive.library.tu.ac.th/thesis/2017/TU_2017_5922040638_9167_8282.pdf.

[30] PrimalCortex. Measuring home energy consumption with the pzem004t and esp8266. https://primalcortex.wordpress.com/2019/07/06/measuring-home-energy-consumption-with-the-pzem004t-and-esp8266/. Accessed: 2020.10.13.

[31] Zakriya Mohammed, Ibrahim Abe M Elfadel, and Mahmoud Rasras. Monolithic multi degree of freedom (mdof) capacitive mems accelerometers. *Micromachines*, 9(11):602, 2018.

[32] Mohd Ismifaizul Mohd Ismail, Rudzidatul Akmam Dziyauddin, Noor Azurati Mohd Salleh, Robiah Ahmad, Marwan Hadri Azmi, and Hazilah Mad Kaidi. Analysis and procedures for water pipeline leakage using three-axis accelerometer sensors: Adxl335 and mma7361. *IEEE Access*, 6:71249–71261, 2018.

[33] Emir Husni and Folkes Laumal. The development of an earthquake early warning system using an adxl335 accelerometer. In *2018 21st Saudi Computer Society National Computer Conference (NCC)*, pages 1–5. IEEE, 2018.

[34] Marcus Varanis, Anderson Langone Silva, and Arthur Guilherme Mereles. On mechanical vibration analysis of a multi degree of freedom system based on arduino and mems accelerometers. *Revista Brasileira de Ensino de Física*, 40, 2017.

[35] Basith Adli and Pranoto Hidaya Rusmin. Vibration measuring tools for rotary pumping machine with accelerometer mems sensor. In *2020 FORTEI-International Conference on Electrical Engineering (FORTEI-ICEE)*, pages 69–74. IEEE, 2020.

[36] Dileep Kumar Soother, Jawaid Daudpoto, and A Shaikh. Vibration measurement system for the low power induction motor. *Engineering Science And Technology International Research Journal*, 2(4):53–57, 2018.

[37] Emerson Galdino and Alexandre Cury. Development of low-cost wireless accelerometer for structural dynamic monitoring. *Rev Interdiscip Pesqui Eng RIPE*, 2:10–19, 2017.

[38] Analog Devices. Small, low power, 3-axis ±3 g accelerometer adxl335. https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf. Accessed: 2020.11.29.

[39] Marcus Varanis, Anderson Silva, Arthur Mereles, and Robson Pederiva. Mems accelerometers for mechanical vibrations analysis: A comprehensive review with applications. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 40(11):1–18, 2018.

[40] Analog Devices. 3-axis, ±2 g/±4 g/±8 g/±16 g digital accelerometer adxl345. https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf. Accessed: 2020.11.29.

[41] Md Hasibuzzaman, Abu Shufian, Rubayat Kabir Shefa, Rifat Raihan, Joy Ghosh, and Avijit Sarker. Vibration measurement & analysis using arduino based accelerometer. In *2020 IEEE Region 10 Symposium (TENSYMP)*, pages 508–512. IEEE, 2020.

[42] Yum Ji Chan and Jing-Wei Huang. Multiple-point vibration testing with micro-electromechanical accelerometers and micro-controller unit. *Mechatronics*, 44:84–93, 2017.

[43] I InvenSense. Mpu-6000 and mpu 6050 product specification revision 3.4. *United States*, 2013.

[44] Nikhil Agnihotri. How to use the adxl345 accelerometer sensor. https://www.engineersgarage.com/electronic-projects/adxl345-accelerometer-sensor-how-to-use/, 2021. Accessed: 2021.05.17.

[45] Ria Sood, Manjit Kaur, and Hemant Lenka. Design and development of automatic water flow meter. *International journal of computer science, engineering and applications*, 3(3):49, 2013.

[46] Pushkar Singh and Sanghamitra Saikia. Arduino-based smart irrigation using water flow sensor, soil moisture sensor, temperature sensor and esp8266 wifi module. In *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 1–4. IEEE, 2016.

[47] J Lalnunthari and HH Thanga. Dependence of hall effect flow sensor frequency on the attached inlet and outlet pipe size. In *2017 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 56–60. IEEE, 2017.

[48] Anif Jamaluddin, Dewanto Harjunowibowo, Dwi Teguh Rahardjo, Egy Adhitama, and Syamsul Hadi. Wireless water flow monitoring based on android smartphone. In *2016 2nd International Conference of Industrial, Mechanical, Electrical, and Chemical Engineering (ICIMECE)*, pages 243–247. IEEE, 2016.

[49] Gaurav Gosavi, Gajanan Gawde, and Gautam Gosavi. Smart water flow monitoring and forecasting system. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 1218–1222. IEEE, 2017.

[50] Waleed Ali Badawi. Underground pipeline water leakage monitoring based on iot. *International Journal of MC Square Scientific Research*, 11(3):01–08, 2019.

[51] Deeksha Srivastava, Awanish Kesarwani, and Shivani Dubey. Measurement of temperature and humidity by using arduino tool and dht11. *International Research Journal of Engineering and Technology (IRJET)*, 5(12):876–878, 2018.

[52] Warren Gay. *Advanced Raspberry Pi. Raspbian Linux and GPIO Integration. Second edition*, chapter 22. Apress, 2018.

[53] Milica Lekić and Gordana Gardašević. Iot sensor integration to node-red platform. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–5. IEEE, 2018.

[54] Mihai Bogdan. How to use the dht22 sensor for measuring temperature and humidity with the arduino board. *ACTA Universitatis Cibiniensis*, 68(1):22–25, 2016.

[55] Adafruit. Digital relative humidity and temperature sensor am2302/dht22. `https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf`. Accessed: 2020.04.01.

[56] Paul Macheso, Tiwonge D Manda, Sylvester Chisale, Nelson Dzupire, Justice Mlatho, and Didacienne Mukanyiligira. Design of esp8266 smart home using mqtt and node-red. In *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, pages 502–505. IEEE, 2021.

[57] Ihsan Jabbar Hasan, Nahla Abdul Jalil Salih, Nadhir Ibrahim Abdulkhaleq, and Mohannad Jabbar Mnati. An android smart application for an arduino based local meteorological data recording. In *IOP Conference Series: Materials Science and Engineering*, page 042014. IOP Publishing, 2019.

[58] Texas Instruments. Lm35 precision centigrade temperature sensor. `https://www.ti.com/lit/ds/symlink/lm35.pdf`. Accessed: 2020.04.01.

[59] Yusuf Mulge Poonam and Yusuf Mulge. Remote temperature monitoring using lm35 sensor and intimate android user via c2dm service. *International Journal of Computer Science and Mobile Computing*, 2(6):32–36, 2013.

[60] Salomi S Thomas, Amar Saraswat, Anurag Shashwat, and Vishal Bharti. Sensing heart beat and body temperature digitally using arduino. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 1721–1724. IEEE, 2016.

[61] Maxim Integrated. Ds18b20 programmable resolution 1-wire digital thermometer. `https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf`, 2019. Accessed: 2020.04.05.

[62] Mohamed Fezari and Ali Al Dahoud. Exploring one-wire temperature sensor 'ds18b20'with microcontrollers. *Badji Mokhtar Annaba University, University of Al-Zaytoonah Faculty of IT, Jordan*, 2019.

[63] Ali Nur Fathoni, Noor Hudallah, Riana Defi Mahadji Putri, Khusnul Khotimah, Tri Rijanto, and Miftahul Ma'arif. Design automatic dispenser for blind people based on arduino mega using ds18b20 temperature sensor. In *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, pages 1–5. IEEE, 2020.

[64] Win Hlaing, Somchai Thepphaeng, Varunyou Nontaboot, Natthanan Tangsunantham, Tanayoot Sangsuwan, and Chaiyod Pira. Implementation of wifi-based single phase smart meter for internet of things (iot). In *2017 International Electrical Engineering Congress (iEECON)*, pages 1–4. IEEE, 2017.

[65] Ruby Jain, Nidhi Tiwari, and Mukesh Yadav. A comparison study of wifi 6 and wifi 5. *choice*, 5:6, 2020.

[66] Lei Qiao, Zhe Zheng, Wenpeng Cui, and Liang Wang. A survey on wi-fi halow technology for internet of things. In *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pages 1–5. IEEE, 2018.

[67] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, volume 20, 2017.

[68] A Ramelan, F Adriyanto, BAC Hermanu, MH Ibrahim, JS Saputro, and O Setiawan. Iot based building energy monitoring and controlling system using lora modulation and mqtt protocol. In *IOP Conference Series: Materials Science and Engineering*. IOP Publishing, 2021.

[69] Komkrit Chooruang and Kraison Meekul. Design of an iot energy monitoring system. In *2018 16th International Conference on ICT and Knowledge Engineering (ICT&KE)*, pages 1–4. IEEE, 2018.

[70] Mostafa Labib, Atef Ghalwash, Sarah Abdulkader, and Mohamed Elgazzar. Networking solutions for connecting bluetooth low energy devices-a comparison. In *MATEC Web of Conferences*, volume 292, page 02003. EDP Sciences, 2019.

[71] F John Dian, Amirhossein Yousefi, and Sungjoon Lim. A practical study on bluetooth low energy (ble) throughput. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 768–771. IEEE, 2018.

[72] Tutun Juhana and Arif Indra Irawan. Non-intrusive load monitoring using bluetooth low energy. In *2015 9th International Conference on Telecommunication Systems Services and Applications (TSSA)*, pages 1–4. IEEE, 2015.

[73] Majid Karami, Gabrielle Viola McMorrow, and Liping Wang. Continuous monitoring of indoor environmental quality using an arduino-based data acquisition system. *Journal of Building Engineering*, 19:412–419, 2018.

[74] Ievgeniia Kuzminykh, Arkadii Snihurov, and Anders Carlsson. Testing of communication range in zigbee technology. In *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, pages 133–136. IEEE, 2017.

[75] Nanda Kishor Panda, Mayank Senapati, S Meikandasivam, D Vijaykumar, and Jaganatha Pandian. Zigbee based clamp-on iot energy meter. In *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*, pages 126–131. IEEE, 2019.

[76] Oratile Khutsoane, Bassey Isong, and Adnan M Abu-Mahfouz. Iot devices and applications based on lora/lorawan. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 6107–6112. IEEE, 2017.

[77] Afef Mdhaffar, Tarak Chaari, Kaouthar Larbi, Mohamed Jmaiel, and Bernd Freisleben. Iot-based health monitoring via lorawan. In *IEEE EUROCON 2017-17th International Conference on Smart Technologies*, pages 519–524. IEEE, 2017.

[78] Chang-Sic Choi, Jin-Doo Jeong, Il-Woo Lee, and Wan-Ki Park. Lora based renewable energy monitoring system with open iot platform. In *2018 international conference on Electronics, Information, and Communication (ICEIC)*, pages 1–2. IEEE, 2018.

[79] Akram Syed Ali, Zachary Zanzinger, Deion Debose, and Brent Stephens. Open source building science sensors (osbss): A low-cost arduino-based platform for long-term indoor environmental data collection. *Building and Environment*, 100:114–126, 2016.

[80] Milan Matijevic and Vladimir Cvjetkovic. Overview of architectures with arduino boards as building blocks for data acquisition and control systems. In *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, pages 56–63. IEEE, 2016.

[81] Chung-Hyuk Yim, Tae-Gue Oh, and Gyu-Sik Kim. Applicaton of usb serial communication to radon measuring system. *International Journal on Recent and Innovation Trends in Computing and Communication*, 5(1):06–09, 2017.

[82] Jan Mallari. How to set up uart communication on the arduino. `https://www.circuitbasics.com/how-to-set-up-uart-communication-for-arduino`. Accessed: 2020.10.13.

[83] Robotics Back-End. Raspberry pi arduino serial communication – everything you need to know. `https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/`. Accessed: 2020.10.13.

[84] André Glória, Francisco Cercas, and Nuno Souto. Comparison of communication protocols for low cost internet of things devices. In *2017 South Eastern European Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, pages 1–6. IEEE, 2017.

[85] Pavel Hubenov. Development of i2c communication based on logics, 2017.

[86] Simon Monk. *Programming Arduino Next Steps: Going Further with Sketches*. McGraw-Hill Education, 2 edition, 2016.

[87] Farid Baskoro, Miftahur Rohman, and Aristyawan Putra Nurdiansyah. Serial peripheral interface (spi) communication application as output pin expansion in arduino uno. *INAJEEE (Indonesian Journal of Electrical and Electronics Engineering)*, 3(2):34–40, 2020.

[88] Simon Monk. *Programming Arduino Next Steps: Going Further with Sketches*. McGraw-Hill Education, 2 edition, 2016.

[89] Simon Monk. *Programming Arduino Next Steps: Going Further with Sketches*. McGraw-Hill Education, 2 edition, 2016.

[90] Md Zakaria Islam, MA Kader, and Chowdhury Mohammad Masum. Design and implementation of data acquisition and control system using 1-wire protocol, 2021.

[91] Suxian Cai, Yunfeng Wu, Ning Xiang, Zhangting Zhong, Jia He, Lei Shi, and Fang Xu. Detrending knee joint vibration signals with a cascade moving average filter. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 4357–4360. IEEE, 2012.

[92] Hadi Kordestani, Yi-Qiang Xiang, and Xiao-Wei Ye. Output-only damage detection of steel beam using moving average filter. *Shock and Vibration*, 2018, 2018.

[93] Pengfei Luo, Min Zhang, Yile Liu, Dahai Han, and Qing Li. A moving average filter based method of performance improvement for ultraviolet communication system. In *2012 8th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)*, pages 1–4. IEEE, 2012.

[94] H Azami, B Bozorgtabar, and M Shiroie. Automatic signal segmentation using the fractal dimension and weighted moving average filter. *Journal of Electrical & Computer science*, 11(6):8–15, 2011.

[95] Tinouna Asma, Ghanai Mouna, Ouali Mohammed Assam, and Chafaa Kheireddine. Efficient filtering framework for electrocardiogram denoising. *International Journal Bioautomation*, 23(4):403, 2019.

[96] Dipta Chaudhuri, Moloy Mukherjee, Mofazzal H Khondekar, and Koushik Ghosh. Simple exponential smoothing and its control parameter: A reassessment. In *Recent Trends in Signal and Image Processing*, pages 63–77. Springer, 2019.

[97] Deboleena Sadhukhan and Madhuchhanda Mitra. Ecg noise reduction using fourier coefficient suppression. In *Proceedings of The 2014 International Conference on Control, Instrumentation, Energy and Communication (CIEC)*, pages 142–146. IEEE, 2014.

[98] Andre Dekker Pieter Kubben, Michel Dumontier. *Fundamentals of Clinical Data Science*. Springer, 1 edition, 2019.

[99] Çiğdem Polat Dautov and Mehmet Siraç Özerdem. Wavelet transform and signal denoising using wavelet method. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2018.

[100] Anil Prashanth Rodrigues, Grynal D'Mello, and P Srinivasa Pai. Selection of mother wavelet for wavelet analysis of vibration signals in machining. *Journal of Mechanical Engineering and Automation*, 6(5A):81–85, 2016.

[101] Manel Rhif, Ali Ben Abbes, Imed Riadh Farah, Beatriz Martínez, and Yanfang Sang. Wavelet transform application for/in non-stationary time-series analysis: a review. *Applied Sciences*, 9(7):1345, 2019.

[102] Alaa A Jaber and Robert Bicker. Real-time wavelet analysis of a vibration signal based on arduino-uno and labview. *International Journal of Materials Science and Engineering*, 3(1):66–70, 2015.

[103] Tanuj Yadav and Rajesh Mehra. Denoising and snr improvement of ecg signals using wavelet based techniques. In *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, pages 678–682. IEEE, 2016.

[104] Lizhe Tan and Jean Jiang. *Digital signal processing: fundamentals and applications*. Academic Press, 2018.

[105] Damien Zufferey, Christophe Gisler, Omar Abou Khaled, and Jean Hennebert. Machine learning approaches for electric appliance classification. In *Machine learning approaches for electric appliance classification*, pages 740–745. IEEE, 2012.

[106] Andreas Reinhardt, Paul Baumann, Daniel Burgstahler, Matthias Hollick, Hristo Chonov, Marc Werner, and Ralf Steinmetz. On the accuracy of appliance identification based on distributed load metering data. In *2012 Sustainable Internet and ICT for Sustainability (SustainIT)*, pages 1–9. IEEE, 2012.

[107] Vibhatha Abeykoon, Nishadi Kankanamdurage, Anuruddha Senevirathna, Pasika Ranaweera, and Rajitha Udawalpola. Real time identification of electrical devices through power consumption pattern detection. *Pervasive Comput*, 10(1):40–48, 2016.

[108] Antonio Ridi, Christophe Gisler, and Jean Hennebert. Automatic identification of electrical appliances using smart plugs. In *2013 8th International Workshop on Systems, Signal Processing and their Applications (WoSSPA)*, pages 301–305. IEEE, 2013.

[109] Christophe Gisler, Antonio Ridi, and Jean Hennebert. Appliance Consumption Signature Database and Recognition Test Protocols. In *WOSSPA2013 The 9th International Workshop on Systems, Signal Processing and their Applications 2013*, pages 336–341, 2013. Some of the files below are copyrighted. They are provided for your convenience, yet you may download them only if you are entitled to do so by your arrangements with the various publishers.

[110] Zeynep Duygu Tekler, Raymond Low, Yuren Zhou, Chau Yuen, Lucienne Blessing, and Costas Spanos. Near-real-time plug load identification using low-frequency power data in office spaces: Experiments and applications. *Applied Energy*, 275:115391, 2020.

[111] Raghunath Shivram Reddy, Niranjan Keesara, Vikram Pudi, and Vishal Garg. Plug load identification in educational buildings using machine learning algorithms. In *Proceedings of BS2015: 14th Conference of International Building Performance Simulation Association, Hyderabad, India*, pages 1940–1946, 2015.

[112] Innocent Mpawenimana, Alain Pegatoquet, Win Thandar Soe, and Cécile Belleudy. Appliances identification for different electrical signatures using moving average as data preparation. In *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*, pages 1–6. IEEE, 2018.

[113] Andrea Tundis, Ali Faizan, and Max Mühlhäuser. A feature-based model for the identification of electrical devices in smart environments. *Sensors*, 19(11):2611, 2019.

[114] Antonio Ridi and Jean Hennebert. Hidden markov models for ilm appliance identification. *Procedia Computer Science*, 32:1010–1015, 2014.

[115] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, chapter 5. O'Reilly, 2019.

[116] Shan Suthaharan. *Machine Learning Models and Algorithms for Big Data Classification*, chapter 9. Springer, 2016.

[117] Jaime Lynn Speiser, Michael E Miller, Janet Tooze, and Edward Ip. A comparison of random forest variable selection methods for classification prediction modeling. *Expert systems with applications*, 134:93–101, 2019.

[118] Zhongheng Zhang, Yiming Zhao, Aran Canes, Dan Steinberg, Olga Lyashevska, et al. Predictive analytics with gradient boosting in clinical medicine. *Annals of translational medicine*, 7(7), 2019.

[119] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3):1937–1967, 2021.

[120] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4), 2015.

[121] Watchdog timer. https://os.mbed.com/cookbook/WatchDog-Timer. Accessed: 2021.05.01.

[122] U blox team. Nina-w10 series (open cpu). https://www.u-blox.com/en/product/nina-w10-series-open-cpu. Accessed: 2021.10.25.

[123] Arduino nano 33 iot. https://store.arduino.cc/arduino-nano-33-iot. Accessed: 2021.04.21.

[124] Fritzing software. https://fritzing.org/download/. Accessed: 2021.04.25.

[125] Arduino. Arduino ide 1.8.15. https://www.arduino.cc/en/software. Accessed: 2021.10.25.

[126] Adafruit Industries. Dht sensor library. https://github.com/adafruit/DHT-sensor-library. Accessed: 2021.10.25.

[127] Arduino. Dallas semiconductor's 1-wire protocol. https://playground.arduino.cc/Learning/OneWire/. Accessed: 2021.10.26.

[128] Vladimir Savchenko. Arduino library for maxim temperature integrated circuits. https://github.com/vlast3k/Arduino-libraries/tree/master/DallasTemperature. Accessed: 2021.10.26.

[129] Jakub Mandula. Pzem-004t v3.0. https://github.com/mandulaj/PZEM-004T-v30. Accessed: 2021.10.26.

[130] Žygimantas Jasiūnas. Pzem-004t-v30-samd21. https://github.com/zygisjas/PZEM-004T-V30-SAMD21. Accessed: 2021.10.27.

[131] Arduino library scheduler. https://www.arduino.cc/reference/en/libraries/scheduler/. Accessed: 2021.05.30.

[132] Wifinina library. https://www.arduino.cc/en/Reference/WiFiNINA. Accessed: 2021.04.15.

[133] Branden Hall. Websocket client and server for arduino. https://www.arduino.cc/en/Reference/WiFiNINA. Accessed: 2021.04.16.

[134] Nadav Matalon. Watchdog timer. https://github.com/nadavmatalon/WatchDog. Accessed: 2021.04.15.

[135] Antonio Ridi, Christophe Gisler, and Jean Hennebert. Acs-f2—a new database of appliance consumption signatures. In *2014 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, pages 145–150. IEEE, 2014.

[136] Emir Salihagić, Jasmin Kevric, and Nejdet Doğru. Classification of on-off states of appliance consumption signatures. In *2016 XI International Symposium on Telecommunications (BIHTEL)*, pages 1–6. IEEE, 2016.

[137] Antonio Ridi, Christophe Gisler, and Jean Hennebert. Processing smart plug signals using machine learning. In *2015 IEEE wireless communications and networking conference workshops (WCNCW)*, pages 75–80. IEEE, 2015.

[138] Saeed Mian Qaisar, Futoon Alsharif, Abdulhamit Subasi, and Ahmed Bensenouci. Appliance identification based on smart meter data and event-driven processing in the 5g framework. *Procedia Computer Science*, 182:103–108, 2021.

[139] Xgboost parameters. `https://xgboost.readthedocs.io/en/latest/parameter.html`. Accessed: 2021.03.20.

[140] A. Ridi, C. Gisler, and J. Hennebert. ACS-F2 - A new database of appliance consumption signatures. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 145–150. IEEE, Aug 2014.

[141] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. Tsfel: Time series feature extraction library. *SoftwareX*, 11:100456, 2020.

[142] Matej Andrejašic. Mems accelerometers. In *University of Ljubljana. Faculty for mathematics and physics, Department of physics, Seminar*, 2008.

[143] Erik Grahn. Evaluation of mems accelerometer and gyroscope for orientation tracking nutrunner functionality, 2017.

[144] Jaromír Jezný and Miloslav Čurilla. Position measurement with hall effect sensors. *American Journal of Mechanical Engineering*, 1(7):231–235, 2013.

[145] Dejan. Dht11 and dht22 sensors temperature and humidity tutorial using arduino. `https://howtomechatronics.com/tutorials/arduino/dht11-dht22-sensors-temperature-and-humidity-tutorial-using-arduino/`. Accessed: 2020.04.01.

[146] Intan Sugiyanti. Design of atm crime monitoring system based on mqtt protocol using sim800l and arduino mega 2560, 2019.

[147] Kumar Gaurav. Bluetooth connectivity in ios – part 1. `https://frugalisminds.com/bluetooth-connectivity-ios-part-1/`. Accessed: 2021.05.25.

[148] Arduino nano 33 iot pinout sheet. `https://content.arduino.cc/assets/Pinout-NANO33IoT_latest.pdf`. Accessed: 2021.04.21.

[149] Raspberry pi 4 computer. `https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf`. Accessed: 2021.05.30.

[150] Raspberry pi 4 gpio. `https://www.raspberrypi.org/documentation/usage/gpio/README.md`. Accessed: 2021.05.30.

Appendix A

Sensor working principles

A.1 MEMS accelerometer working principles

MEMS term took shape around 1989, and it is describing a research field where mechanical elements such as membranes and cantilevers are manufactured at a scale of microelectronics. These devices are equipped with parts in size of less than 100µm. Such systems are manufactured using a technique called micro-fabrication. MEMS, unlike simply microelectronics, have special characteristics that draw the boundary between such systems and microelectronics. Microelectronics are solid and compact objects, MEMS have holes, cavities, membranes, cantilevers, mechanical parts on a micro-scale. MEMS Accelerometer is an electromechanical device that measures acceleration forces. These forces split into two classes: static and dynamic. The static constant force of gravity pulling down (1G) or 9.80665m/s^2, the dynamic force that might be caused by an oscillation of an object [142].

To understand the basics of MEMS accelerometer, we can take a look at the second-order spring-mass-damper system in a Figure A.1 which is operating on Newton's second law which states that the algebraic sum of all the forces equals the inertial force of the proof mass. When an acceleration (a) is applied to proof mass (m) suspended by springs with a spring constant (k),and having a damping (b), then the force ($F_{applied}$) acting upon the proof mass [31, 143].

Typical MEMS accelerometer contains a movable proof of mass, like in Figure A.1, with plates that is attached through a mechanical suspension system to reference frame, as shown in Figure A.2. It has two kinds of plates, fixed ones and movable ones that are attached to springs. Those plates represent capacitors. If there is no acceleration, capacitance's C_1 and C_2 are equal. If the proof of mass is displaced, acceleration is measured by tracking capacitance difference. We will not discuss further how capacitance difference is being calculated because it is not our primary research goal, and it is out of our work scope.

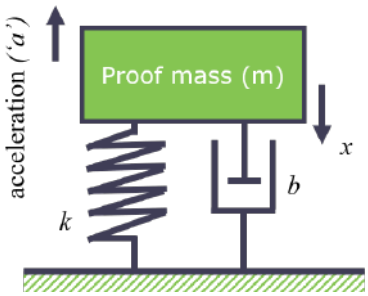


Figure A.1: Accelerometer model based on spring-mass-damper.

We are mentioning this to have an overall understanding of how MEMS accelerometer sensors function [142].

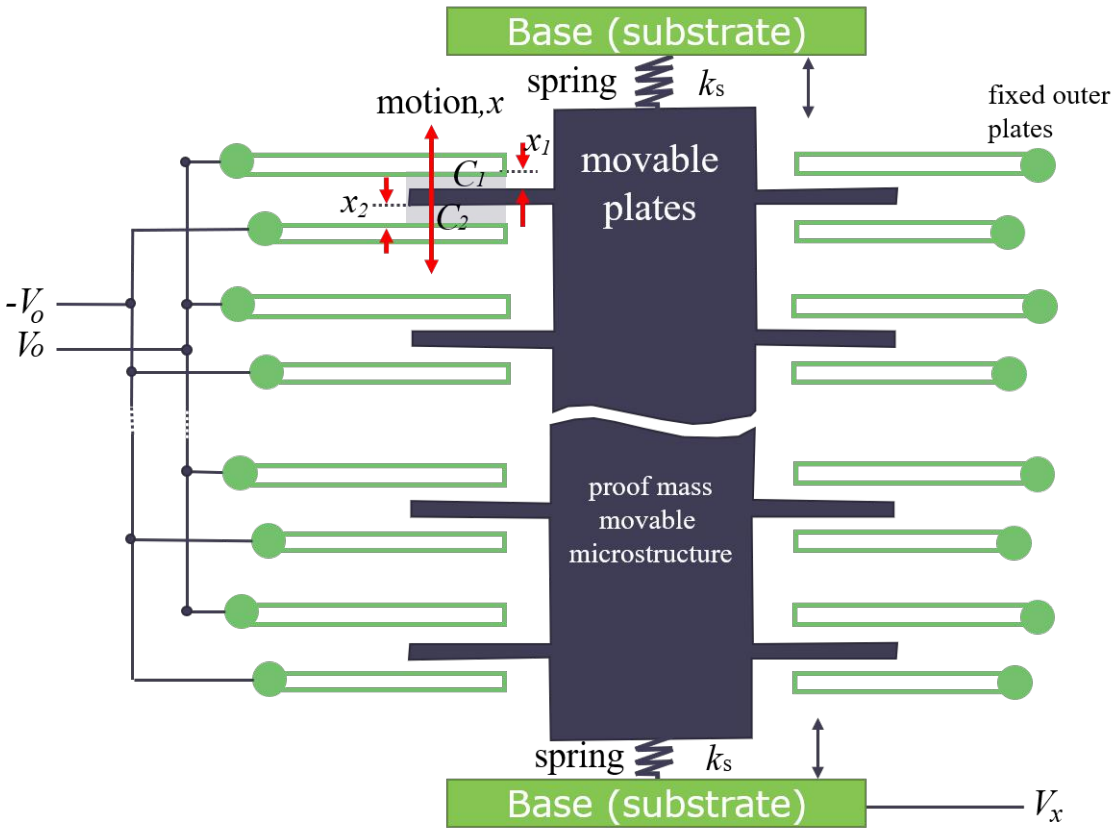


Figure A.2: Single axe accelerometer structure. Proof of mass is attached to the base through springs. In this example proof of mass can only move up and down, meaning, that it can measure the oscillation of single axis.

If we take a single-axis accelerometer part demonstrated in Figure A.2 and place set of them in perpendicular direction, we get multiple-axis accelerometer, this approach can be implemented to measure even three axis: x/y/z. There are two approaches to implement multiple axis MEMS accelerometer. First one is to have separate proof of mass modules implemented under the same chip or use single proof of mass to measure 3D axis.

A.2 Hall-effect water flow sensor working principle

The way Hall Effect transducer works is illustrated in Figure A.3, Hall conductive plate outputs voltage depending on two factors power supply and magnetic field. As illustrated in the left side, when plate is not affected by any magnetic field, the plate will not return any voltage. In the right side of the illustration we can see, when plate is affected by magnetic force, "Hall" voltage is developing across the sides of the plate based on Lorentz force. This charge can be measured and its strength is proportional to magnetic field force strength. Sensors that are using hall effect can be digital or analogue. In analogue type sensor, the voltage is directly measured after being amplified. Digital sensor based on hall effect acts more like "ON", "OFF" signal, or "HIGH", "LOW" [144]. For example, previously mentioned water-flow sensor has digital response, that is how it is generating square wave signal output.

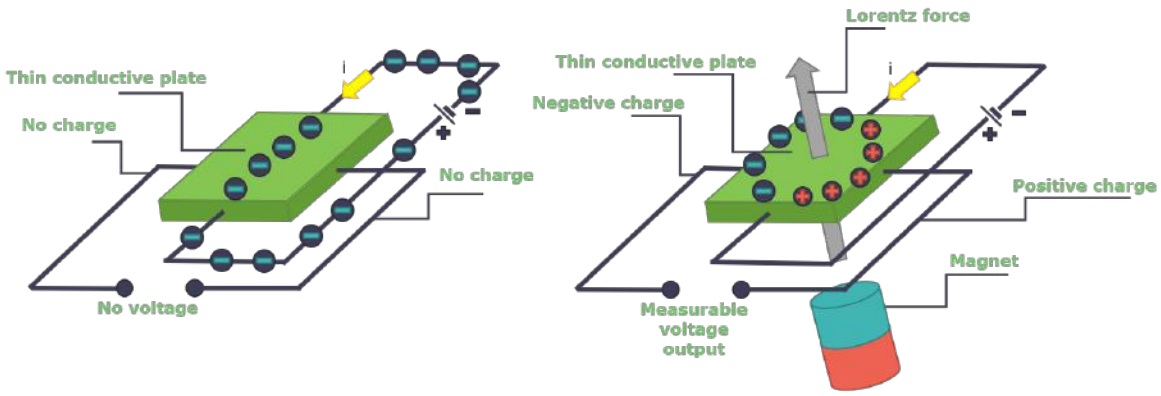


Figure A.3: Hall effect principle. Left side when plate is not affected by a magnetic field. Right side when plate is affected by a magnetic field creating Lorentz force.

A.3 Temperature and humidity sensors DHC11/DHC22 working principle

Both sensors (DHT11,DHT22) work in the same principle, but DHT22 has an improved accuracy and bandwidth of measuring temperature and humidity in the Figure A.4, we can see the inner part of the sensor without the protective layer.

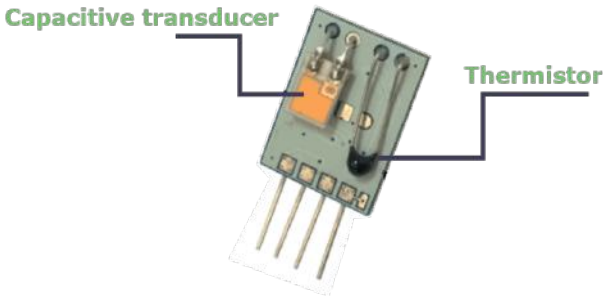


Figure A.4: The inner part of DHT22 sensor.

Humidity is being sensed by using capacitive transducer, as illustrated in Figure A.5 (a), we have two thin electrode plates and in between is a moisture pad, as the moisture changes in pad, the conductivity between electrode plates changes as well, Internal Circuitry (IC) is measuring the resistance then converting this into ready to read data for microcontroller [145].

For temperature measurement sensor has a build in thermistor. In other words, it is a variable resistor which has an ability to change resistance with the change of temperature. As illustrated in A.5 (b), DHC11 and DHC22 contains sintered semi-conductive material such as ceramics or polymers for providing higher resistance changes with even a small change in the temperature. When temperature increases, the sensor resistance decreases, IC is responsible for measuring the resistance level and converting it into temperature data [145].

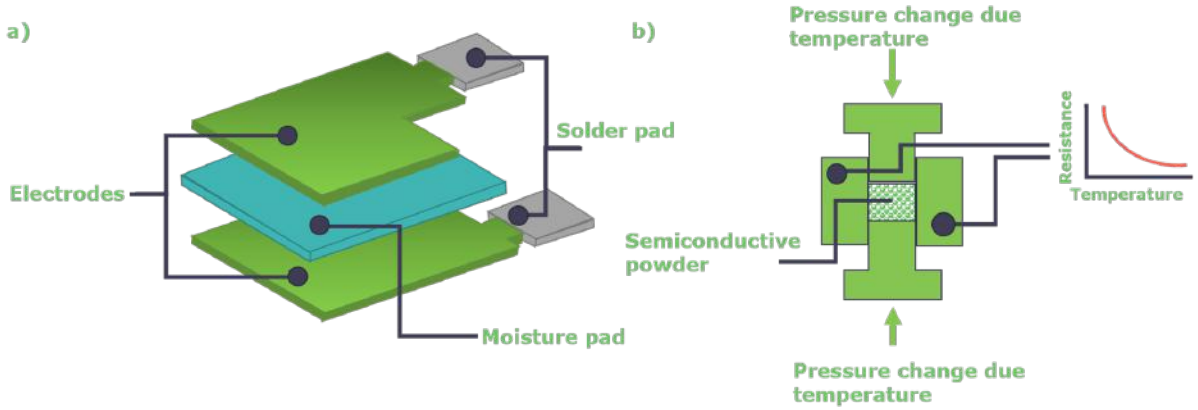


Figure A.5: a) capacitive transducer working principle, b) thermistor working principle.

Appendix B

Appliance monitoring related architectures from the literature

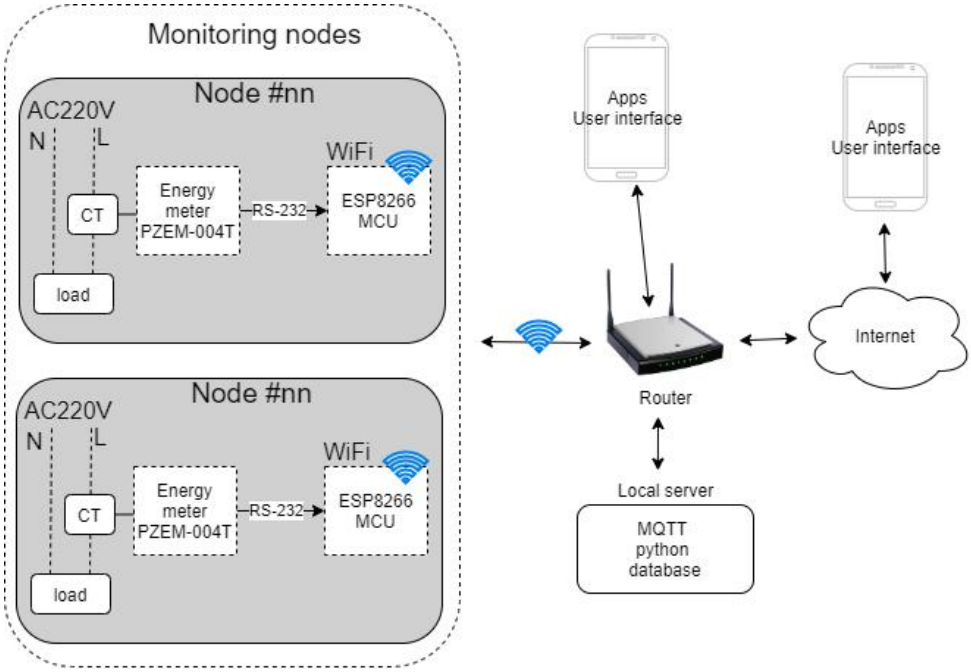


Figure B.1: IoT energy monitoring architecture.

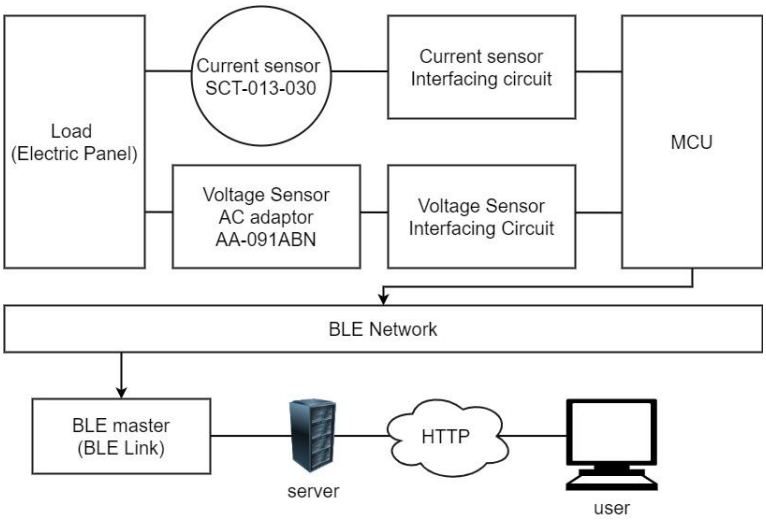


Figure B.2: Load monitoring architecture based on BLE.

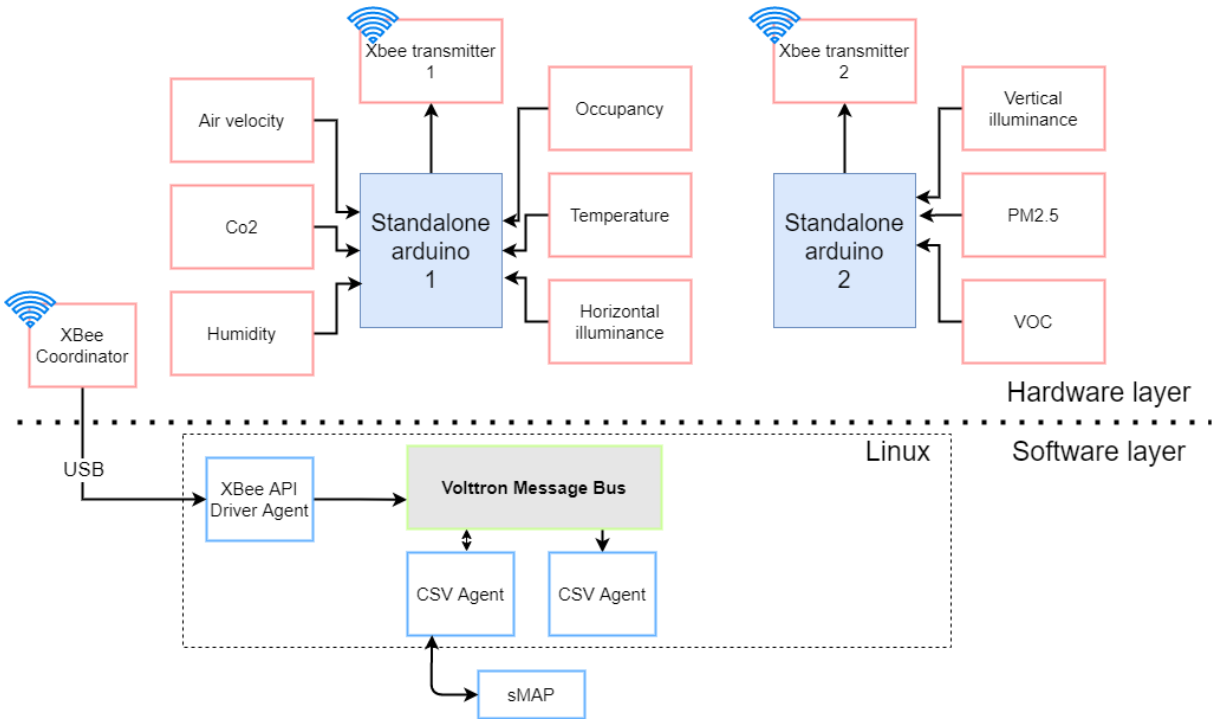


Figure B.3: Indoor environment monitoring solution based on ZigBee.

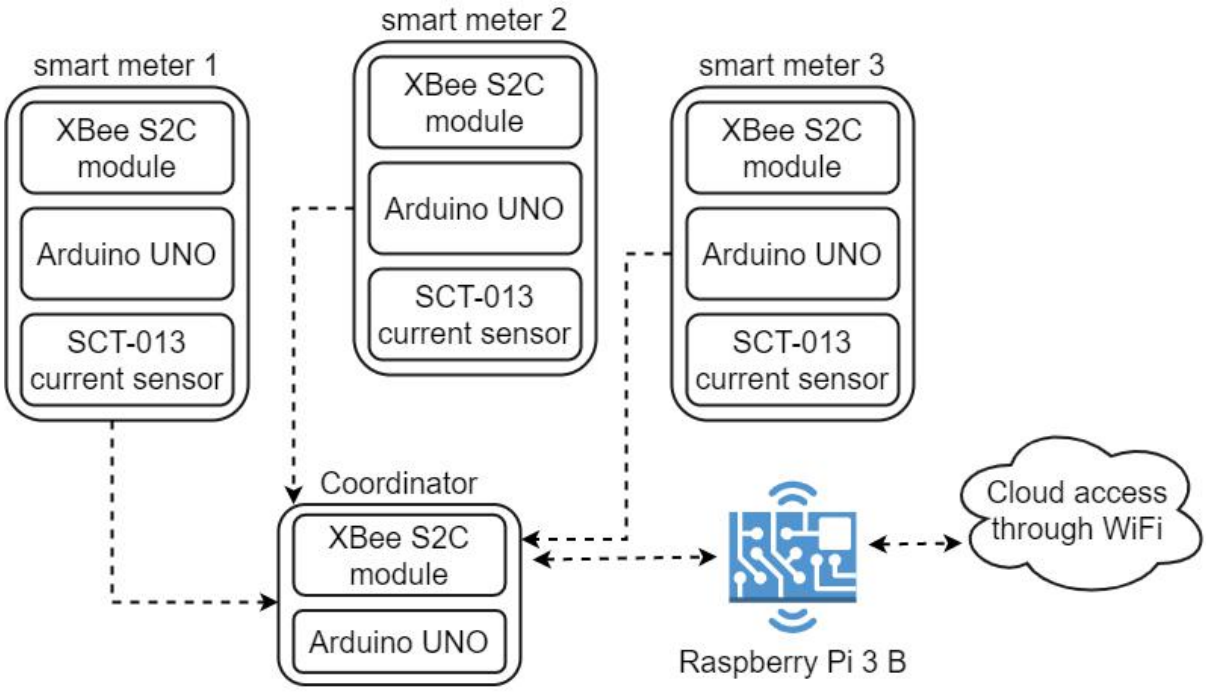


Figure B.4: Xbee based smart energy metering design

Appendix C

SATO platform system architecture

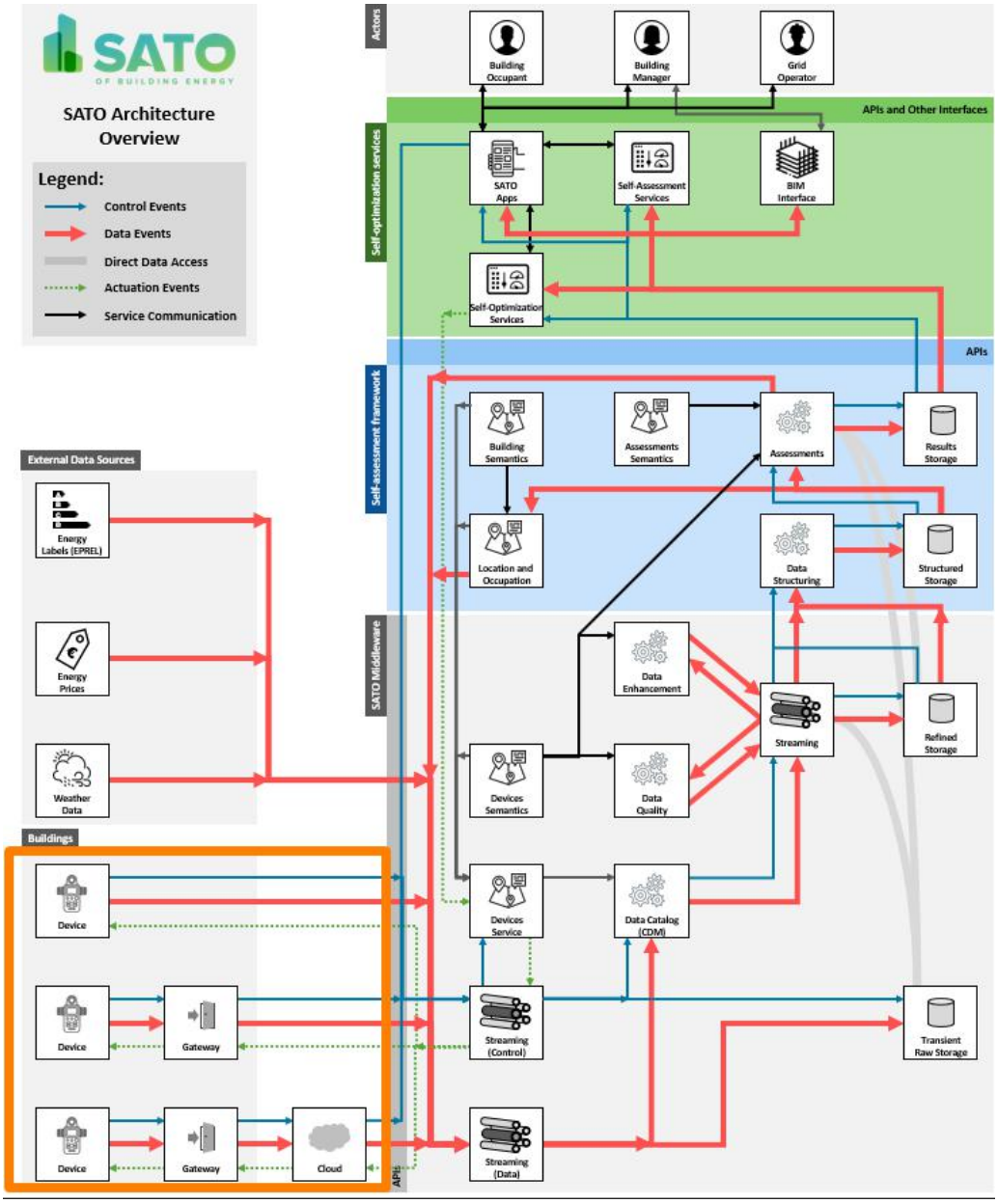


Figure C.1: Overall SATO system architecture. In the orange area we are marking the scope of this project report.