Master's thesis

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

Even Vestland

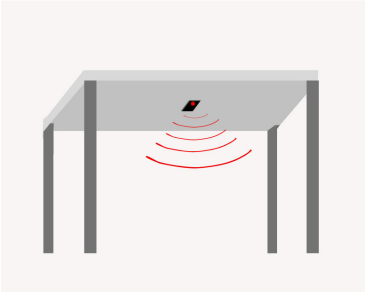
Proof-of-Concept for Work-Station Monitoring System

Master's thesis in Cybernetics and Robotics
Supervisor: Sverre Hendseth
June 2023

NTNU
Norwegian University of
Science and Technology

Even Vestland

Proof-of-Concept for Work-Station Monitoring System



Master's thesis in Cybernetics and Robotics
Supervisor: Sverre Hendseth
June 2023

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

NTNU
Norwegian University of
Science and Technology

Preface

To make it to the end of a Master's Program in Cybernetics and Robotics is no easy feat and is a proud achievement of mine. However, I would never have made it this far without the support from friends, classmates, family and the bouquet of characters that make up the Department of Engineering Cybernetics. I would therefore like to dedicate this final academic work of mine to my sparring partners through 5 years, Simen and Oscar, my supervisor Sverre, my ever supportive family and my biggest fan, my partner Johanne.

Abstract

In recent years, IoT (Internet of Things) applications have steadily become a regular part of our everyday use of technology. In this thesis, a proof-of-concept system is developed for an IoT system detecting occupancy at university workstations. The motivation is to provide students with a convenient way to identify available desk space without physically visiting the location. The specifications and design of the system take into account relevant theory, implementations of similar systems, and current hardware and software standards. Additionally, the system considers constraints such as cost, privacy considerations, scalability, and physical limitations.

A temperature sensor is used to identify occupancy at the workstation. Together with an ESP32 microcontroller and accompanying schematics, it makes up the physical module of the system. The ESP32 is responsible for processing the sensor data, establishing a WiFi connection, and publishing status messages via the MQTT protocol when occupancy is detected. To conserve power, the microcontroller employs sleep modes during periods between measurements and between 1800 and 0700 hours. On the subscribing end of the MQTT protocol, a web dashboard will display the occupancy status and error detection. The interface of the dashboard visually represents the layout of the room, highlighting the status of each workstation in terms of occupancy.

As a proof-of-concept, the system successfully displays the intended functionality of the product, although there is potential to optimize the system further. It should be viewed as a first step toward a final product, which includes a physical setup that can be replicated and placed at campus workstations. The further development of the system is encouraged as it can potentially fill a gap in the market for student-centric logistic solutions.

Sammendrag

I løpet av de siste årene har IoT -løsninger blitt en vanlig del av samhandlingen vår med teknologi i hverdagen. I denne oppgaven utvikles et proof-of-concept for et IoT-system som detekterer bruk av leseplasser på universiteter. Motivasjonen er å gi studentene en praktisk måte å finne ledig leseplass uten å måtte fysisk besøke rommet. Spesifikasjonene og utformingen av systemet tar hensyn til relevant teori, lignende systemer og gjeldende maskinvare- og programvarestandarder. I tillegg er det lagt begrensninger på kostnader, personvernhensyn, skalérbarhet og fysiske begrensninger.

En temperatur sensor brukes for å identifisere tilstedeværelse ved leseplassene. Sammen med en ESP32-mikrokontroller og tilhørende skjematikk, utgjør den den fysiske modulen i systemet. Mikrokontrolleren er ansvarlig for å behandle sensordata, etablere en WiFi-forbindelse og publisere statusmeldinger via MQTT-protokollen når tilstedeværelse oppdages. For å spare strøm, bruker mikrokontrolleren sleep mode i perioder mellom målingene og mellom klokken 1800 og 0700. På mottakersiden av MQTT-protokollen vil et web dashbord vise status og feilmeldinger. Grensesnittet på dashbordet viser rommets utforming, og markerer statusen for hver leseplass.

Som et proof-of-concept viser systemet den tiltenkte funksjonaliteten til produktet, selv om det finnes potensial til å optimalisere systemet ytterligere. Systemet bør betraktes som et første steg mot et endelig produkt, som inkluderer en fysisk oppsett som kan reproduseres og plasseres på leseplassene. Videre utvikling av systemet oppfordres, da det potensielt kan fylle en rom i markedet for logistiske løsninger rettet mot studenter.

Contents

Chapter 1

Introduction

1.1 Motivation

The recent COVID-19 pandemic has significantly affected people's perceptions of home-based work. As a result, many organizations and educational institutions have partially utilized office spaces and desks. Therefore, implementing systems capable of monitoring occupancy and activity in these workplaces is crucial for optimizing their utilization. In addition, this information is relevant to factors such as safety protocols, energy utilization, and general logistics.

The use of sensor technology to survey workplaces and desks is something that already exists in different forms and configurations. Several companies already offer solutions often targeted toward businesses or universities looking for occupancy insights, like IOTSpot (IOTSpot, 2023) and Disruptive Technologies (Boothman, 2022). These solutions often include integrated platforms that provide dashboard solutions, climate monitoring, and other sensor insights.

These solutions are often costly and may offer more functionality than required. Moreover, disclosing sensitive information to a third party can be unsettling, despite their compliance with security guidelines and assurances of data protection.

In this thesis, a system similar to the ones described will be developed as a proof of concept. It will feature a real-time interface that displays information on work-station availability without storing the data. The system will be aimed for use at Gløshaugen Campus, NTNU. There exists a number of free-to-use desks and workplaces on this Campus. For students to be able to see whether or not a desk is available before commuting to school would optimize time planning and reduce frustration.

Such a system will need to be able to differentiate between people using the workplace and

other obstacles, such as a chair or a backpack. Also, the system's scalability to encompass a significant number of workplaces requires careful deliberation regarding practicality, cost, power consumption, communication systems, and user-friendliness.

1.2 Problem Statement

The goal of this thesis is to develop a proof-of-concept system for perceiving presence at workstations on campus and displaying the availability through a web interface. The focus will lie on developing a cost-effective, practical, and scalable system.

1.2.1 Objectives

- Develop a functioning sensor and hardware system capable of detecting presence at a workstation.
- Develop an intuitive interface that allows the user to see if workstations on campus are available.
- Develop the system with a basis in user requirements and current theory.
- Keeping the complexity and cost of the system at a level that ensures that it can be further developed to a prototype and scaled for multiple systems.
- Make sure the privacy concerns of the users are respected.

1.2.2 Limitations

As a proof of concept, the product of this thesis will only consider the functionality of the system. All though all choices will be made with regard to a future prototype. This means that a power supply and an implementable physical system will not be developed in this thesis but theoreticized and discussed.

Chapter 2

Background

This chapter will present previous works related to desk occupancy monitoring. It will also introduce theory on hardware and software protocols that will be implemented and discussed in later chapters.

2.1 Related Works

2.1.1 Current Solutions for Work-Place Monitoring

Many different solutions already exist when it comes to sensor systems designed for workplace monitoring. Teixera et al. (2010, p.33) suggest three sensor-based solutions that will be relevant in the future; a number of motion sensors, camera sensors, and the use of mobile phones for tracking. Even though this study looks at general human sensing and not specifically towards workstations, the first two solutions have been explored in several papers in recent years. Maiti et al. (2022) propose in their paper a system utilizing three sensors: two distance sensors and one tilt sensor placed in front of the desk facing the user. As mentioned in section 1.1, companies like Iotspot (IOTSpot, 2023), and Disruptive Technologies (Boothman, 2022) already deliver solutions for desk monitoring. These solutions often include integrated platforms that provide dashboard solutions, climate monitoring, and other sensor insights relevant to businesses. It is worth noting that the cost of these solutions is relatively high. Although many of these businesses only provide their price range upon request, Disruptive Technologies publicly disclose their starter kit with a price of 6500kr. It includes five sensors and a one-year subscription to their cloud solution (DisruptiveTechnologies, 2023).

The motivations for companies that purchase these types of services may vary. Ball (2010,

p.93) proposed three main reasons why companies wish to monitor their employees: increasing productivity, protecting company interests, and protecting against legal liabilities. However, other studies suggest additional motivations. For instance, the aforementioned study by Maiti et al. (2022) proposes their desk monitoring solution with the aim of enhancing employee health benefits. The sensor systems used are often characterized by their motivations; therefore, many systems that aim to increase health benefits use cameras or multiple sensors to capture sitting posture. An example of this is the work by Sikkandhar and Lim (2021). They implemented a pressure-sensing substrate in a chair to monitor prolonged sitting and posture.

The solutions with these motivations in mind often require expensive and comprehensive solutions, whereas the motivation in this thesis is more narrow in scope and only focuses on sensing presence for the sake of determining occupancy in universities. This sets it apart from several of the solutions mentioned and makes it unique in its simplicity and low cost.

2.1.2 Privacy and Security Concerns in Desk-Occupancy Surveillance

Systems that track the presence and activity of people can be subject to misuse and invasion of privacy. Therefore, it is necessary to ensure the data is anonymous and respect current privacy regulations, such as GDPR in Europe (EPRS, 2020) and CCPA in the US (Bonta, 2023).

Not many studies have been conducted on the effects of monitoring workstation occupancy on students and employers. The few publications that have been conducted mainly focus on more intrusive surveillance than what is relevant for this thesis, like performance monitoring and screen monitoring. Lazar et al. (2022) performed a meta-analysis concluding that this kind of surveillance has a negative impact on employees' stress levels and work satisfaction.

However, the integration of real-time monitoring of occupancy in universities and businesses has been tried before. In October 2022, it was introduced on campus at Northwestern University, US, as part of a study on 'desk usage'. It faced backlash from the student body as they were not informed of the implementation beforehand (Ongweso Jr, 2022). A similar incident happened at The Daily Telegraph Headquarters in 2016 when management failed to inform the staff of the monitoring beforehand (Quinn and Jackson, 2016). In this case, the motivation for the desk monitoring was to gather environmental sustainability data. These incidents exemplify the issues surrounding sensor monitoring of people, even when surveillance is not the objective.

2.2 Theory on Relevant Standards and Protocols

In the following chapters, a number of different hardware and software protocols, standards, and products will be relevant. These will be described in this section.

2.2.1 Hardware Products and Protocols

Arduino

Arduino is a platform developed by the company with the same name. It includes a series of microcontrollers along with a software platform called Arduino IDE (DevicePlus, 2020). The Arduino IDE utilizes a variation of the C++ programming language, known as the Arduino programming language, to code the microcontrollers (Circuito, 2018). It includes a number of built-in libraries and functionalities, in addition to a compiler and an easily configurable serial monitor. A characteristic of the Arduino programming language is the integrated ”void setup” and ”void loop” functions. Their functionality follows what their names imply; the setup function runs once when the microcontroller starts up, and the loop iterates repeatedly. Figure 2.1 shows the Arduino IDE displaying a blank sketch that includes the aforementioned functions.

Figure 2.1: The Arduino IDE

An advantage of the Arduino IDE and programming language is that it is not exclusive to Arduino boards. As one of the most popular choices for embedded projects, Arduino as a platform has a plethora of forums, guides, and example projects online. This makes it a convenient development tool, especially for beginners in the field.

ESP32

The ESP32 is a series of SoC (System-on-a-Chip) microcontrollers developed by Espressif Systems (EspressifSystems, 2023a). Common for all the variants is that they include WiFi and Bluetooth functionality (EspressifSystems, 2023c) (EspressifSystems, 2022). By utilizing ultra-low-power consumption the microcontroller can use these functionalities in a sustainable way. It has four different sleep modes; modem sleep, light sleep, deep sleep, and hibernation, in addition to the active mode. This feature makes it easy to modify the amount of current the ESP32 draws according to its requirements. Hibernation mode exhibits the lowest power consumption. By relying solely on the RTC (Real Time Clock), it draws 4.5 μA of current. Deep sleep mode shares similarities with hibernation mode but additionally uses RTC memory to preserve important values while the ESP32 is in sleep mode. The RTC memory is a RAM (Random Access Memory) with a capacity of 8KB (EspressifSystems, 2023b). This additional feature leads to a higher current consumption of 6.5 μA. The light sleep mode and modem sleep mode consume 800μA and 3-30mA, respectively, as they utilize more processing power compared to the hibernation and deep sleep modes.

ESP32 is a popular choice for hardware development given its low cost and easy applicability. In addition, it is relatively small compared to other microcontrollers, which makes them practical to implement in physical environments. Although different configurations exist within the ESP32 series, most are compatible with communication buses like SPI, UART, and I2C.

The ESP32 series is compatible with the Arduino IDE, which allows it to utilize the Arduino programming language. As mentioned, this is a convenient feature for beginners and experienced developers alike, as the platform has a wealth of guides, forums, and documentation. Figure 2.2 shows the ESP32 Wroom 32 Devkit microcontroller subject to this thesis.

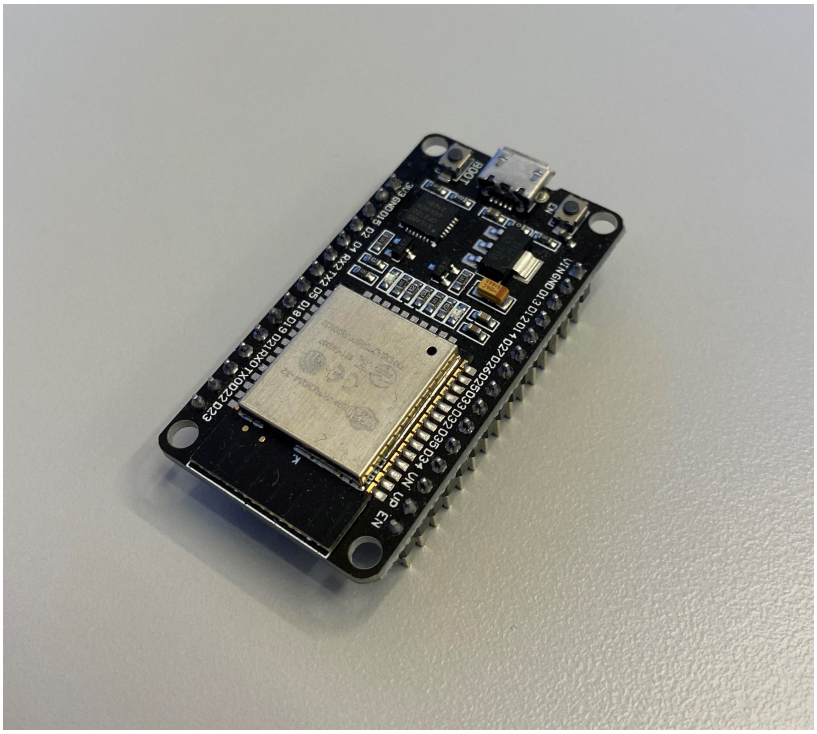


Figure 2.2: A picture of the ESP32 Wroom 32 Devkit Microcontroller utilized in this thesis

D6T Thermal Sensor

The D6T is a series of four temperature sensors developed by Omron Electronics (Omron, 2018). They utilize thermopile sensors that are able to capture infrared rays and calculate the temperature value by comparing it to an internal thermal sensor. This way, they are able to measure surface temperatures without direct contact. The series includes sensors with varying capabilities and costs. The D6T-8L-09 and D6T-44L-06 are the higher-priced options and employ a grid system with 8 and 16 channels, respectively, enabling detailed analysis of surface temperature distribution. The D6T-1A-01, depicted in figure 2.3, and D6T-1A-02 do not utilize this grid system. They are, however, less costly and are practical in situations where only single-point measurements are necessary. All four sensors are only compatible with the I2C bus. The temperature value that is transmitted by the sensors is a three-digit number that needs to be divided by 10 to obtain the correct Celsius value. Since the D6T sensors continuously perform measurements and do not include any power-saving measures, the current to the sensor has to be shut off for it not to draw power.

Figure 2.3: The D6T-1A-01 thermal sensor. The image is gathered from the D6T sensor data sheet (Omron, 2022, 11).

2.2.2 Software Standards and Protocols

I2C

Inter-integrated circuit, known as I2C, is a communication bus widely used in microcontrollers (Campbell, 2016). It bases its functionality on a master-slave relationship between one or more controllers and one or more peripherals. Unlike many other communication protocols like SPI, I2C only needs two signal wires which can lead to an easier physical hardware setup. Since it only utilizes one wire for data transfer the system is semi-duplex, and thus the speed of the communication can be limited. Figure 2.4 illustrates how the I2C protocol functions with one controller and one or two peripherals. The SDA line is the data line with 7-bit addressing, and the SCL line is the clock signal. As the figure shows, the voltage line needs to be connected to the signal lines through pull-up resistors to ensure stable voltage levels.

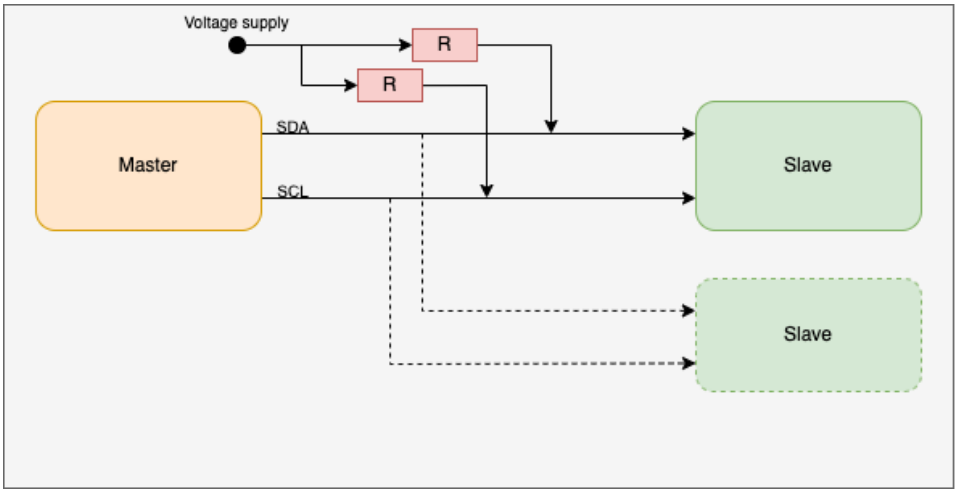


Figure 2.4: A visualization of the I2C communication bus with one controller and one or two peripherals.

MQTT

For IoT-based (Internet of Things) systems, MQTT is a widely used application layer communication protocol. It utilizes a publisher/subscriber model, where a broker handles incoming communication from a client and routes it toward its intended target (Oasis, 2019). This way, the subscriber client and publisher client are unaware of each other and only need the address information of the broker to transfer and receive data. The broker is a piece of software running on a computer or a microcontroller. It allows for a large number of clients to connect to it, making MQTT scalable for large systems. An additional benefit of using the MQTT protocol is its quality of service operation, commonly referred to as QoS. It is a specification that is determined by the client to ensure the level of certainty desired regarding the successful receipt of the transmitted message.

As the MQTT protocol requires few resources for machine-to-machine communication, it is a well-suited protocol for microcontrollers. It often utilizes WebSockets when communicating with browser clients. Websocket is an application layer communication protocol running over the transport layer protocol TCP (IETF, 2011).

Figure 2.5 shows a practical example of how the MQTT communication can look like with 4 clients. Two of the clients in this example are sensors that publish data to the broker on two different topics. A device subscribes to the data from one of the topics and publishes a status message on a different topic. The last client is a server subscribing to two topics and storing the data.

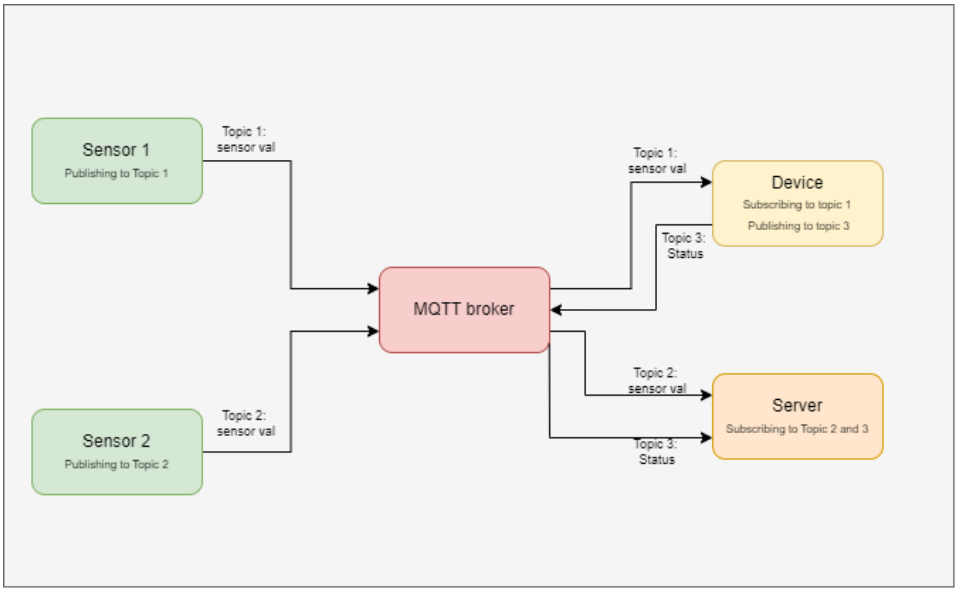


Figure 2.5: A visualization of an example of the MQTT communication protocol with four clients

Multiple providers offer options for establishing a local MQTT broker, such as Mosquitto (Eclipse, 2023). This approach involves manual setup and configuration of the broker on a microcontroller or computer. Alternatively, some providers offer external brokers that require minimal or no configuration but usually come at a cost. Cloud.MQTT is an example of the latter, providing monthly payment plans of 5$, 19$, 99$ and 299$ enabling twenty-five, one hundred, one thousand, and ten thousand client connections, respectively (CloudMQTT, 2023). The 5$ solution also has limitations on the transfer speed at 20kbit/s, while the others have no set limit.

React and Next

React is an open-source software library for building user interfaces in the Javascript and Typescript programming languages. These languages are often used in front-end development, and the difference between them is relatively small. TypeScript is a superset of JavaScript, which means that JavaScript code can also be valid TypeScript code (Microsoft, 2023). However, TypeScript can provide better editor support and code completion, which often makes it easier to work with.

React was developed by engineers working for Facebook, currently renamed Meta, and has since become an integral part of the Facebook website (Thakkar, 2020a). Since its introduction in 2013, it has become a prevalent tool for front-end developers due to its efficient utilization of a component-based structure. This structure permits the creation of individual components that can be combined to form the user interface. React components

utilize hooks, which provide functionality and logic to the components. The hooks are, like components, reusable, which helps developers to write modular and maintainable code.

Since React is exclusively a library, the official React website recommends employing frameworks such as Next.js or Remix to construct full-stack applications (Meta, 2023). Next.js is an open-source tool for rendering applications on the server side developed by the cloud platform company Vercel (Vercel, 2023) (Thakkar, 2020b). By downloading the framework and initializing a project, a number of example files running a local website appear. This is then readily available for editing and further development. Another advantage of the next.js framework is the adaptive creation of HTML files based on the typescript and javascript code. This enables the user to focus on development in these files without creating or editing HTML files.

Chapter 3

Specifications

This chapter will present the specifications for the system. It is divided into the sections Market Specification and Product Specifications. The first section will provide the system layout based on the end user's requirements, and the second section will describe the requirements for the product to realize this layout.

3.1 Market Specifications

The finished system at the end of this thesis is a proof-of-concept, but considerations are made with potential future prototypes in mind. This is especially relevant when determining the initial market specifications. The choices in this section aim to identify the optimal system for the user group within the constraint of cost, scalability, privacy considerations, and physical limitations.

The target market for the system is focused on campus workstations and the primary user group is therefore university students, with the potential deployer of an end-product being universities.

3.1.1 Accessibility

Even though university students are generally more technologically proficient than the average population, the intention is to make the system as user-friendly as possible. This is to make sure that the system will be utilized. This means that the interface needs to be easily accessible to the users. A solution that is available both on mobile devices and web platforms was therefore deemed the most appropriate approach for this purpose. This way

the students can decide whether to commute to school or study from home based on the occupancy of the workstations, as well as keep track of the availability on their phone en route to school.

3.1.2 Interface Layout

The objective of the interface was to create an intuitive user experience. To achieve this, it was decided to display the availability of workstations through a visual representation. A practical way to do this is to create an overhead look of the room containing the desks. This way, the workstations' occupancy status can be signaled using color combinations and symbols. However, it is important to consider color blindness and the effectiveness of different color combinations and symbols as part of the design choices for the interface.

3.1.3 Privacy Considerations and Physical Layout

Privacy is a significant issue in sensor systems designed for monitoring purposes, as demonstrated by the incidents mentioned in subsection 2.1.2. Consequently, it is important to address these concerns in a manner that is acceptable to the students. It was decided that the solution for this issue is to anonymize the data by refraining from storing any sensor data. The system was also decided to be designated for free seating workstations only so the occupancy could not be linked to individuals.

For the sake of transparency, it was also decided to sketch a theoretical physical system that is noticeable without being intimidating to the user. Since the sensor specifications, which will be presented in subsection 3.2.2, also impact the intended placement of the physical system, two solutions were considered applicable; on the desk pointing at the user's head and torso, or under the desk pointing at the user's legs. It was decided that the latter was a more appropriate solution as a sensor facing the user could be a distraction and feel like an intrusion of privacy. However, with this solution, the sensor system is not very noticeable. It is therefore important to mark the workstations so that the students are aware of the monitoring. Figure 3.1 shows the theoretical placement for a prototype of the system under the desk.

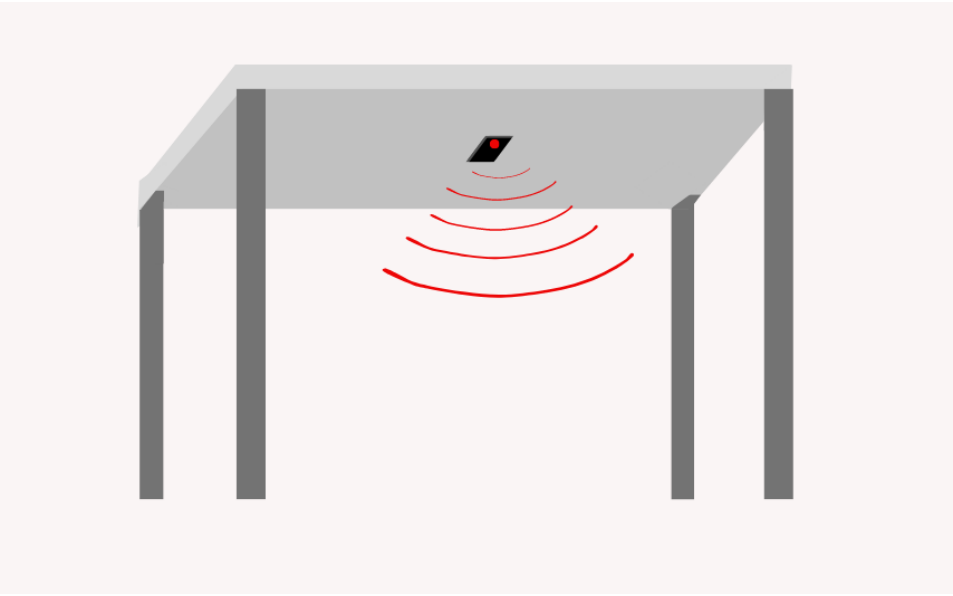


Figure 3.1: A sketched setup of the sensor system at the work-station

Cost and Scalability

For the system to be beneficial to the university, it needs to be at a level of cost that justifies its benefits. However, the level of cost for a proof of concept and theoretical prototype will not correspond with a scaled system. A system encompassing multiple sensor systems spanning multiple study halls will be cheaper per workstation because the price of components in bulk is lower than single components. In addition, taxes and transport fees will be distributed over a larger number of products. This phenomenon, known as economy of scale, will be taken into consideration when deciding on product specifications and design choices. A preliminary cost threshold of 300kr per system was deemed a reasonable limit when keeping this in mind.

3.2 Product Specifications

The product specifications in this section describe the system that meets the requirements of the market specifications and problem statement. It includes an overview of how the objectives are solved without detailing the design and implementations. Since this thesis results in a proof-of-concept, the functionality of the system will be the main focus. This means that a physical casing and a solution for a power supply will not be implemented. However, power-saving measures are still implemented for the sake of a future battery solution.

3.2.1 Overview and Communication

The initial system overview is a framework that includes the communication pathway from the sensor to the interface, as seen in Figure 3.2. A microcontroller was a natural choice for handling sensor data and further communication. The use of the I2C communication protocol between these two components was mainly a consequence of the choice of sensor, which will be further elaborated in subsection 3.2.2. The protocol ensures that the microcontroller can request and receive data from a register address on the sensor. These two components, along with the schematics, make out the physical module of the system.

The communication from the microcontroller was decided to go via WiFi as it is a practical way of machine-to-machine communication. As using WiFi does not need wiring, it would make the hardware setup easier to implement and make the physical component flexible. The communication protocol MQTT was decided to run between the microcontroller and the interface. This protocol was selected due to its established usage in the Internet of Things (IoT) domain and its potential for large-scale communication. To establish the connection, the Eduroam wireless network access service was chosen. Eduroam is a widely-used network access service across multiple universities worldwide. This enabled the possibility of deploying several systems around campus and at other universities with minimal modifications.

Three possibilities were considered for how to run the MQTT broker. First, a computer using the Mosquitto server was reviewed. This solution requires the setup and maintenance in addition to the cost of a computer to be running the server at all times. This would not be feasible in many of the study halls on campus because of a lack of equipment and space.

An alternative to this solution was to run the broker on a microcontroller, e.g. Raspberry Pi, to be able to have a solution that could be permanently switched on. However, this would also require the setup, maintenance, and cost related to the microcontroller.

Lastly, the use of Cloud.MQTT as an external MQTT broker was considered. This way one would not need to deploy and maintain the messaging infrastructure. In addition to being reliable, it is a solution that is able to handle a large number of devices.

With scalability and practicality in mind, the decision was to use Cloud.MQTT as the MQTT broker. This way the system is easy to set up and maintain. However, since the service is not free of charge, a decision also had to be made regarding the payment plan. The 5$ solution was chosen as it satisfied the need for connections to set up the proof-of-concept in this paper, and the transfer speed not having any major implications. Moreover, upgrading the subscription is easy if the need arises.

The WebSocket communication protocol is utilized for communication between the MQTT broker and the dashboard.

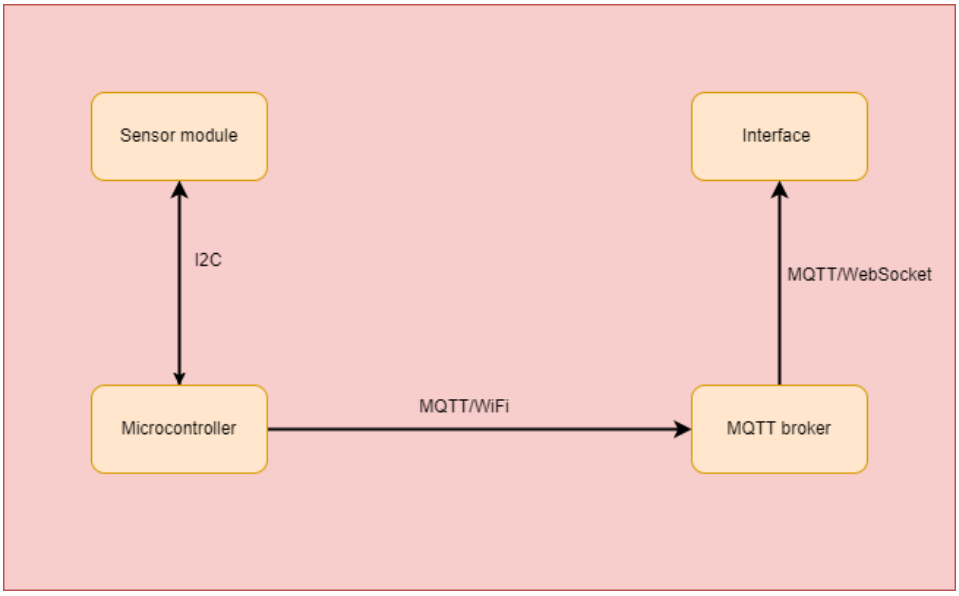


Figure 3.2: Visualization of the information flow from the sensor to the interface

3.2.2 Choice of Sensor

As the choice of sensor would have consequences for the placement, cost, and ease of use of the system as a whole it was a critical decision that required careful consideration. Time and research were therefore put into finding an optimal solution. To maintain a reasonable level of cost and complexity in the system, it was determined that only one sensor would be used per system. Although this deviates from many of the related implementations and theory presented in subsection 2.1.1, the motivations of this thesis only require the measurement of a single point where human occupancy is determined. This approach also eliminates the need for sensor fusion and reduces the additional expenses associated with incorporating multiple sensors in a single system.

Several sensors were evaluated for the detection of occupancy. Initially, an IR, ultrasonic, or camera sensor was considered. However, one of the criteria was that the system should be able to distinguish between a person and other obstacles. IR and supersonic sensors would not be able to distinguish between e.g. a bag on a chair and a student and were therefore deemed unfulfilling. Although camera sensors outperform most other sensors, they often require extensive processing of the image data to identify occupancy. In addition, good-quality cameras are often an expensive solution. A pressure sensor in the chairs was an option that would be both cheap and simple. However, since chairs at workstations on campus are not directly linked with the desks, this solution was deemed unusable.

A thermal sensor was considered the best option as it would be able to distinguish between inanimate objects and people with a normal body temperature. Although slightly more

expensive than supersonic and IR sensors, thermal sensors are within the price range for this project. In addition, the specific sensor that was chosen was the D6T-1A-01 introduced in section 2.2.1, which does not require extensive implementation or processing of the data. The fact that the sensor is only compatible with I2C is not viewed as an obstacle, as it is a commonly used communication protocol in hardware design.

3.2.3 Choice of Microcontroller

A well-suited microcontroller had to be chosen to efficiently run the necessary algorithms for processing the data and controlling the power consumption. An important aspect is the need for a wifi module to transfer data over MQTT. In addition, a cheap and accessible microcontroller is important for the cost and scalability of the system.

There were mainly two possibilities that were considered; Arduino Nano and ESP32. Both these systems are relatively affordable and less complicated to program than many of their competitors. The main difference for the sake of this project is the integrated wifi module on the ESP32. Therefore, it was decided to use the ESP32 as it does not demand the additional ordering, integration, and programming of a wifi module. It only handles 3.3V and therefore requires a level shifter component to connect to the 5V sensor. It also requires an additional 5V power source, but this was not viewed as a large obstacle. Lastly, the power-saving capabilities of the ESP32 sleep modes were seen as an advantage.

The ESP32 was also considered a practical choice as it integrates with the Arduino IDE. Using the Arduino programming language gives one access to many forums and example codes that are useful when programming an embedded system, as mentioned in section 2.2.1.

3.2.4 Embedded Functionality and Power Saving Measures

For the development of the proof-of-concept subject for this thesis, the power supply will be conducted from a computer via USB cables. However, a future prototype of the product as described in section 3.1 would require a battery as a power supply to allow for mobility. Therefore, many of the specifications and design choices are made with the idea of keeping power consumption at a minimum. There are limits to how much can be done to reduce the current consumption without impacting the functionality of the system. However, measures were implemented to optimize the software's efficiency and the time period during which the system would be operational.

It was decided as a power-saving measure to only have the system be active between the hours of 0700 in the morning and 1800 in the evening. This way, the sleep modes of the ESP32 can be utilized and let the microcontroller consume a significantly lower amount of power.

The time step between sensor measurements also impacts the total time the ESP32 is in

active mode. It was decided to be 5 minutes as a compromise between power consumption and the necessity of updated information on the interface. Furthermore, the default value of the desks on the dashboard interface is set to available, so the microcontroller only needs to transmit data from the ESP32 if the desks are occupied.

3.2.5 Interface

As the interface needed to be accessible on multiple platforms, a web-based dashboard was deemed the most practical approach. While other options, such as mobile and web applications, were considered, it was concluded that the development workload required for these platforms was not justified by their potential benefits. A web-based dashboard can be displayed on any device with a web browser. In addition, it can be expanded and further developed if the project is to be scaled e.g. to encompass multiple rooms.

Several methods and libraries exist for developing a web page. Two important aspects of the decision were the compatibility with MQTT and the scale of tools in the form of web forums, guides, and packages. The best-suited choice for this was decided to be the React.js library with the Next.js framework. As an extensive library, React includes several packages and components that will be useful when developing the interface, such as the MQTT package and the canvas environment. In addition, the adaptive programming of HTML files and readily available environment make the Next.js framework, in combination with the React.js library, a suitable tool for this project.

The functionality that was required for the front-end software was to receive the occupancy status from the MQTT broker and display this on the interface. As there is no need for power-saving measures on this side of the system, the step time for checking this is set to 10 seconds. Additionally, the dashboard includes error detection to handle situations where sensor data retrieval fails, displaying an appropriate message to inform the user about the error.

Chapter 4

Design

This chapter will introduce the choices made in regard to the design of the system. That includes the design of the circuit, the embedded software algorithm, and the interface.

4.1 Components and Schematics

The microcontroller and the sensor are the major components in the hardware setup, and thus all other components are consequential. The bidirectional level shifter converts the 3.3 voltage signals from the microcontroller to 5 voltage signals that the sensor is able to handle and the other way around. Each of the power supplies is connected to their respective signal wires through 10 kΩ pull-up resistors. The resistors are necessary for the I2C protocol, as seen in Figure 2.4, by making sure the signals are retrieved at the expected voltage level.

The sensor comes with a fastened connector that either requires a standardized harness or the soldering of wires. Although slightly more expensive, the harness was chosen as the easy implementation, and time-saving was prioritized.

In addition to the components listed in Table 4.1, a breadboard and wires were acquired for connecting the components.

Component	Product name	Manufacturer	Amount	Cost [NOK]
Thermal sensor	D6T-1A-01	Omron Electronics	1	123
Microcontroller	ESP32 Wroom 32 Devkit	Espressif Systems	1	100
Logic shifter	5V-3V IIC UART SPI 4 Channel	CFsunbird	1	25
Resistors	10K ohm resistor	*	4	*
Wire harness	2JCIE-HARNESS-01	Omron Electronics	1/2	62

Table 4.1: Components needed per desk setup. *The resistors were retrieved free of cost from a lab on campus

The wiring from the microcontroller is determined based on the pin layout of the board. It is shown in Figure 4.1 and displays how pin number 21 and 22 handles the SDA and SCL signal lines, respectively. It also features the placements of the relevant power pins.

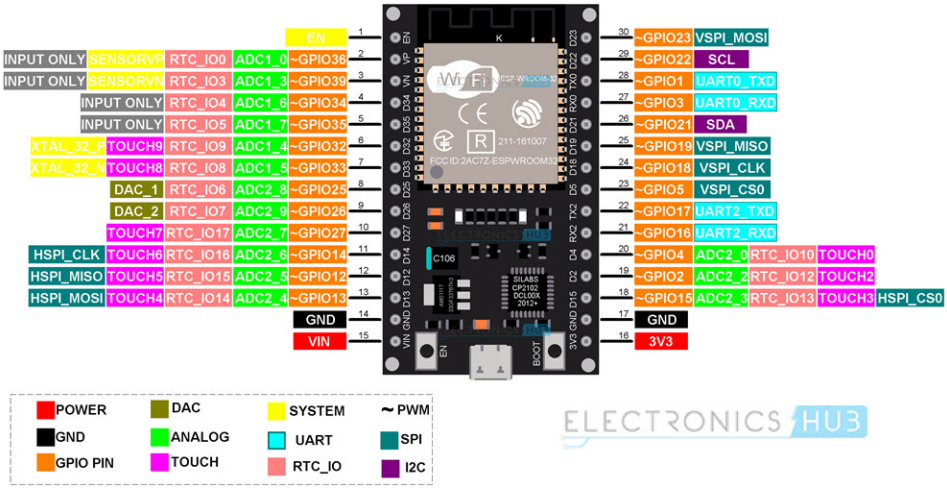


Figure 4.1: The pin layout of the ESP32 Wroom 32 Devkit. It is gathered from the electronic engineering forum Electronics Hub (Teja, 2021)

The schematics were drawn out in the electronic design environment, KiCad. It is largely based on a scenario from the D6T sensor user manual, which features a level shifter (Omron, 2018, p.6). As Figure 4.2 shows, the microcontroller receives 3.3V from a power supply that is transmitted to the level shifter. The SDA and SCL signal is also wired to this

component on the 3.3V side. The other side is connected to a 5V power supply that, again, is connected to the power input of the sensor. The signal lines on the 5V side of the level shifter are connected to the respective lines on the sensor. In addition, on both sides of the component, the power lines are connected to the two signal lines through pull-up resistors. Lastly, all components are connected to the same ground.

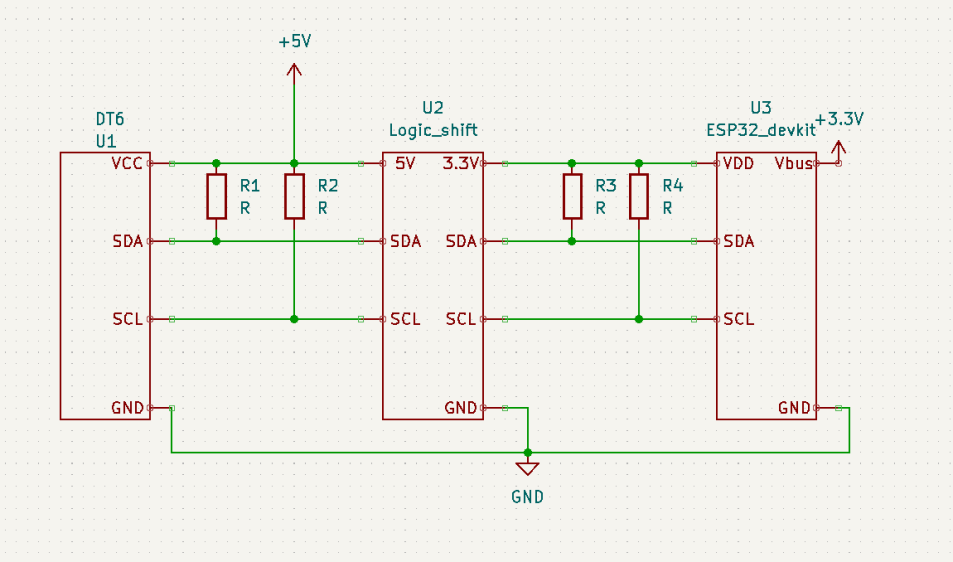


Figure 4.2: The schematics drawn out in KiCad

4.2 Software Design

4.2.1 Embedded Software

As the choice of microcontroller was the ESP32, the embedded software was decided to be written in the Arduino IDE. The main objective of the software is to retrieve, process, and efficiently transmit sensor data. It includes procedures for obtaining and handling the temperature values, establishing connections to the network and the MQTT broker, and transmitting data to the appropriate MQTT topic. Additionally, a key objective for the embedded software is to limit current consumption by utilizing the ESP32 sleep modes. Even though the hibernation mode draws the least amount of current, the fact that it disables the RTC memory makes it infeasible for the embedded software algorithm. In the instances where sleep mode is enabled, deep sleep mode is therefore utilized.

Figure 4.3 displays the flowchart of the embedded algorithm. It shows how the microcontroller configures the RTC to the current local time before it enters the algorithm's main loop. In this loop, the sensor value is read and handled. If the value of the sensor is above

the threshold indicating occupancy, the ESP32 connects to WiFi and sends a message to the MQTT broker. The microcontroller then disconnects and sleeps for 5 minutes before the next iteration. However, if the time is between 1800 in the evening and 0700 in the morning, the RTC is updated to adjust for deviations, and the sleep mode is activated until 0700 o'clock.

Lastly, error detection is implemented for cases where the ESP32 receives no sensor data. A message will then be sent to the MQTT broker that the dashboard algorithm will handle.

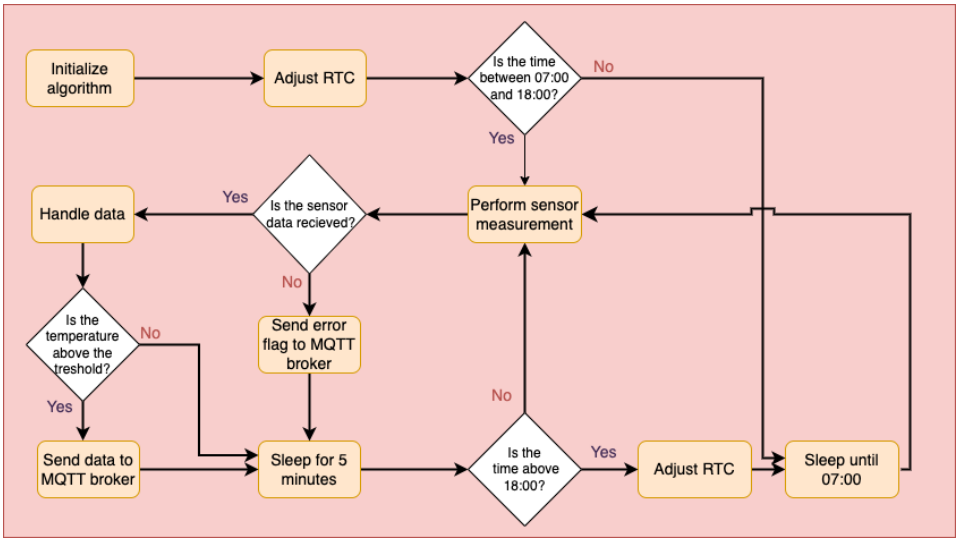


Figure 4.3: The flow chart detailing the embedded software algorithm

Two variables are stored in the RTC memory to make sure the logic of the algorithm is upheld. The first is a boolean flag to determine if the initial configuration is finished. This is needed so the action is not redone every time the ESP32 boots after sleep. The second variable is the counter that keeps track of the number of measurements throughout the day. A threshold is defined to correspond to the number of measurements a day it takes for the time to be 1800. When the counter variable is equal to this threshold, the system sleeps for the night. This means that due to fluctuations in the connection time to the WiFi and MQTT broker, the threshold will be hit at various times after 1800, depending on the day. However, this is not seen as a problem, as the predetermined time limit of 1800 is only set based on the assumption that students typically leave school around that time.

4.2.2 Interface Design

It was decided to develop a simple design showcasing a room consisting of 14 work-places. It was modeled after a room for master students in the old physics building at

Gløshaugen Campus, NTNU. A sketched layout is drawn out using the online diagramming tool, Draw.io, and is shown in Figure 4.4. It displays the workplaces enclosed by cubicles with a screen at the back of the desk, signaling the way the user faces. The colors green and red signal which workplaces are free to use and which are not, respectively. An additional cross was added to the desks that were occupied with the consideration of people with color blindness. For the cases when the sensor data retrieval fails, the desk color turns yellow and displays an error message. The color choice of the empty space was a light gray to separate the interface from the whiteness of the web browser. Lastly, a tilted line was added to show where the entrance to the room was.

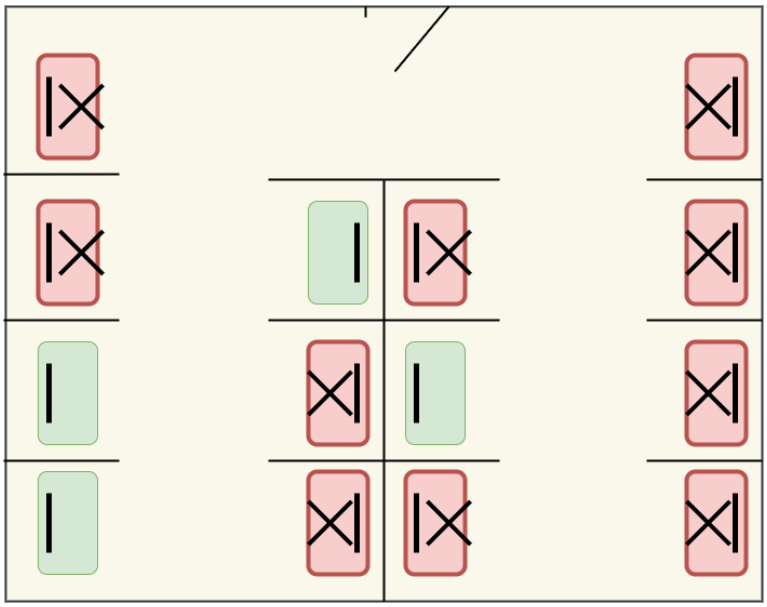


Figure 4.4: The initial sketched layout for the web interface

4.2.3 Front-end Software

The main functionality of the algorithm written in the Next.js framework is to display the designed interface depicting whether or not the workstations are occupied. This is done with the logic detailed in Figure 4.5. After the website is initialized, the algorithm checks for incoming messages on the relevant MQTT topic every tenth second. If the message signals occupancy, the desk color is set to red and marked with an X, and if no message is received, the default color of green is unchanged. Finally, if the message received is the error signal mentioned in subsection 4.2.1, a message will appear on the interface saying that there is an error with the sensor, and the desk color will turn yellow.

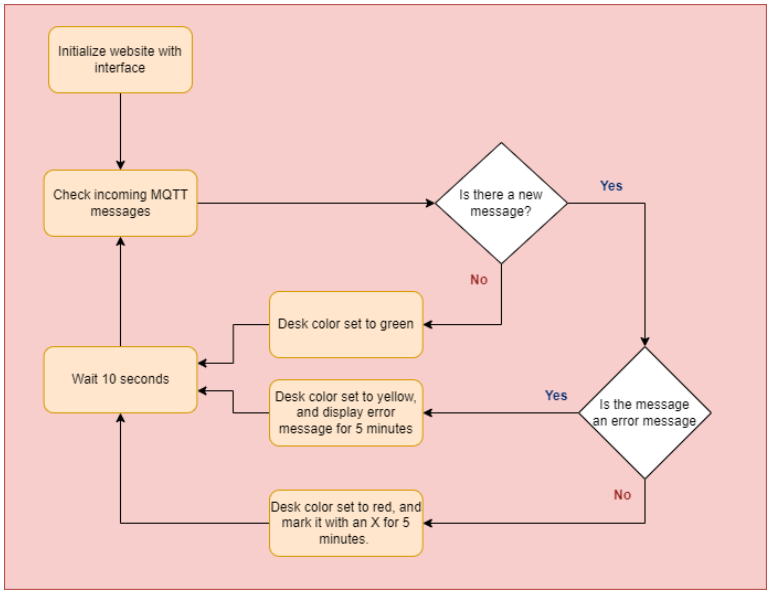


Figure 4.5: The flow chart detailing the algorithm that runs the interface

Chapter 5

Implementations

This chapter will present the implementations that were carried out based on the specifications and design choices. The implementation of the system was executed stepwise. Initially, the completion of the hardware setup was carried out and tested. The connection to Eduroam and the MQTT broker was then established before the embedded software was finished. Lastly, the algorithm running the interface was completed.

5.1 Hardware Setup and Testing

The hardware setup followed the design described in section 4.1. The wire harness was cut in two so that the cut wires could be soldered to pins and connected to the breadboard. In addition, an LED connected to ground through a 220ohm resistor was used to inspect the power supply of the connections. The level shifter was soldered onto a pin setup for it to be placed on the breadboard. An Arduino UNO board was used as a source for the sensor side 5V. The ESP32 and Arduino were both connected to a computer with a USB cable. The finished setup is shown in Figure 5.1.

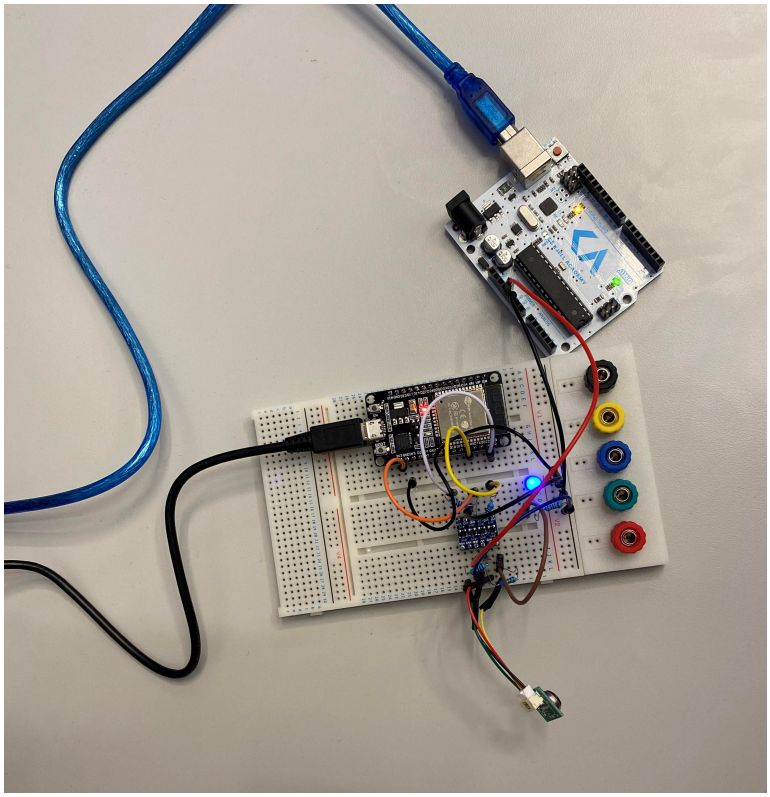


Figure 5.1: The setup for the hardware based on the schematics in figure 4.2.

The functionality for retrieving the temperature data was programmed in the Arduino IDE for testing the sensor. Listing 5.1 shows the functionality for receiving this data. The addresses that are used are gathered from the sensor's user manual, "D6T_ADDR" for the D6T default address and D6t_CMD for the data register (Omron, 2018, p.13).

```
Wire.begin();
uint8_t data[DATA_LENGTH];
Wire.beginTransmission(D6T_ADDR);
Wire.write(D6T_CMD);
Wire.endTransmission();
Wire.requestFrom(D6T_ADDR, DATA_LENGTH);
String message;
if(Wire.available() >= DATA_LENGTH) {
    for(int i=0; i<DATA_LENGTH; i++) {
      data[i] = Wire.read();
    }
    int temperature = data[0] + (data[1] << 8);
    Serial.print("Temperature data from D6T: ");
    Serial.println(temperature);
} else {
    Serial.println("Error: Temperature data not recieved ");
}
```

Listing 5.1: The algorithm for retrieving temperature data from the sensor

The testing involved placing the sensor in different environments. This included inside a hand, pointing at a cold bottle, pointing at a room temperature floor, and pointing at a human thigh. The last two placements were instrumental in setting the threshold for desk occupancy. The sensor had a slight bias toward a higher temperature. It measured $23.6\,^{\circ}\mathrm{C}$ pointing towards the floor in a room with a temperature of $20\,^{\circ}\mathrm{C}$. When the sensor was pointing towards a thigh with a distance of 10cm, it measured $26.5\,^{\circ}\mathrm{C}$. This prompted the threshold to be set at $24.8\,^{\circ}\mathrm{C}$.

The testing also revealed that the adjustment period from one temperature to another was slow and could take up to a couple of minutes, depending on the temperature difference. This was not necessarily seen as a problem as the time step for the system was 5 minutes.

5.2 Connection to WiFi and MQTT

Since Eduroam requires login data, a username and password were created for the ESP32. This was done by the NTNU IT department. With the use of the wifi.h library on the Arduino platform, the ESP32 is able to connect to the network successfully. The algorithm that connects to the Eduroam network is largely based on the GitHub repository of Chlebovec (2023).

As stated in subsection 3.2.1, the choice of MQTT broker was the external solution provided by CloudMQTT. This required the setup of a user account. The 5$ subscription that was chosen is referred to as The Humble Hedgehog Plan on the CloudMQTT website. It provides the server name, username, password, and port which is required for a device to connect to the broker. When the connection is established, a topic can be created that allows for the ESP32 to publish and receive messages from the broker. On the microcontroller, the Arduino library PubSubClient.h is used to connect, subscribe and publish to the

MQTT broker. Functionality is included in the form of while-loops to ensure that other operations do not initiate while the connection to both Eduroam and the MQTT broker is being established. This is shown in Listing 5.2 for the WiFi network connection.

```
1
2   Serial.print(F("Connecting to network"));
3   WiFi.disconnect(true);
4   WiFi.mode(WIFI_STA);
5   WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
        EAP_PASSWORD);
6   while (WiFi.status() != WL_CONNECTED) {
7     delay(500);
8     Serial.print(F("."));
9   }
```

Listing 5.2: The algorithm for connecting to WiFi

A message in the form of ”Test message from ESP32” was sent on a test topic from the microcontroller. The CloudMQTT website features a log of connections and transmissions that are useful when investigating if the communication is successful. Figure 5.2 shows the log at the CloudMQTT website after the test message is published from the ESP32, including the QoS-flag which is set to zero.

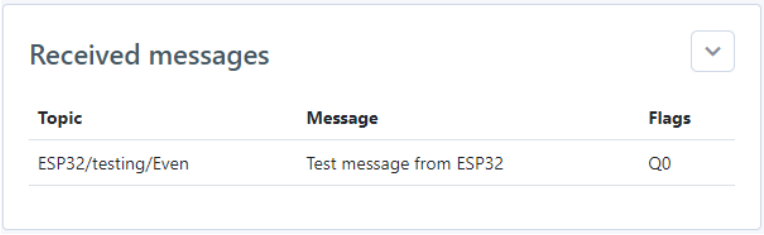


Figure 5.2: The test message published by the ESP32 displayed on the CloudMQTT website

The Serial Monitor in the Arduino IDE was used for error detection and testing during the implementation of the software. Figure 5.3 shows the output after successfully connecting to the network and MQTT broker. In this case, the ESP32 continuously sends the temperature data while also receiving messages on the topic ”Old_physics/desk2”.

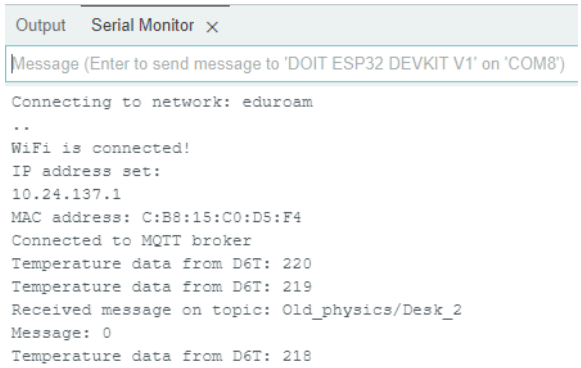


Figure 5.3: The Arduino IDE serial monitor after the ESP32 has connected to the WiFi and MQTT server, and are receiving measurements.

5.3 Embedded Software Algorithm

The embedded software is written as a sole file in the Arduino IDE. The whole algorithm with comments and error detection in the form of printed strings to the serial monitor can be seen in Appendix A. For the algorithm to run with minimal current consumption, the serial setup functionality and all printed strings need to be excluded. In addition, the functionality for subscribing to the MQTT topic is excluded from the finished code as the algorithm only needs to publish messages.

Both the initialization of the system and the main loop take place in the ”setup” function. Since the ESP32 reboots every time it goes to deep sleep and begins on the top of the setup function again, there is no need to use the integrated ”loop” function. The two variables that are utilized by the RTC memory are called ”measurementsPerformedToday” and ”first_boot”. The first one iterates through the main loop, while the second one is a boolean flag that is used to avoid invoking the configuration every time the microcontroller wakes from sleep.

```
1
2  void setup(){
3    if (first_boot){
4      configureSystem();
5      first_boot = false;
6    }
7    performMeasurement();
8    if (measurementsPerformedToday == maxNumberOfDailyMeasurements){
9      adjustTimeAndSleep();
10   }
11   esp_sleep_enable_timer_wakeup(sleepBetweenMeasurements * uS_TO_S_FACTOR)
         );
12   esp_deep_sleep_start();
13 }
```

Listing 5.3: The main loop of the algorithm running on the ESP32

In order to streamline the algorithm, the program's main tasks have been divided into three distinct functions.

The first function, "configureSystem", is responsible for adjusting the RTC to the local time in Trondheim and initiating the main loop based on the current time of day.

The second function, "performMeasurement", is responsible for requesting temperature data from the D6T sensor and assigning a value to the "message" variable that is then transmitted via the MQTT topic. If the temperature exceeds the threshold, the message value is set to 1. If no data is received, the message value is set to 2. Finally, if the temperature is below the threshold, the message value is set to 0, and no data is transmitted. Before the function finishes, it iterates one more count for the "measurementsPerformedToday" variable.

Lastly, the "adjustTimeAndSleep" function adjusts the RTC and calculates the time until 0700 the next morning. It then sets the "measurementsPerformedToday" to zero and goes to sleep for the calculated amount of time.

In addition to these main functions, the algorithm for connecting to the Eduroam WiFi network was placed in a function called "WiFiConnect".

The algorithm for retrieving data from the sensor was developed by using the standard I2C library on the Arduino platform, named "wire.h". The temperature reading came in as two 8-bit data strings and could be displayed directly in the IDE's serial monitor. It displayed a 3-digit number for each measurement representing the degrees in Celcius, whereas the last number was a decimal. As the temperature threshold was defined as $24.8\,^{\circ}\mathrm{C}$, this was set to 248 in the code.

5.4 Web Dashboard

As the decision was to use React with the Next.js framework, the necessary packages and modules has to be downloaded. To get started with React, node.js needs to be downloaded, which is a Javascript runtime environment that is used to run websites on external servers. Next.js is then downloaded, and the command ”create-next-app” needs to be run in the terminal for the necessary framework to be readily available. Lastly, the MQTT package extension requires downloading. The programming environment employed to construct the website is VS Code, chosen due to its practicality and user-friendly features.

As the initial framework is given in TypeScript, it was decided to continue to use this, instead of JavaScript. The first implementation is the design of the website. A file named Canvas.tsx was created in the Next framework for the design to be programmed directly. The necessary hooks are imported to add functionality to the component. With the use of the Canvas environment in React, the design is drawn out inside a hook, with the use of lines, rectangles, and color combinations.

A few minor adjustments are made to the design in Figure 4.4, mainly based on the limitations of the canvas component and to make a clearer design. The rounded edges of the desks are made straight, and the desks are made longer. In addition, the color tunes are changed to give a clearer contrast. All proportions within the design are relative to the width and height of the interface rectangle. This way the entire canvas is interconnected and it enables resizing without distorting the shapes within the design.

In order to obtain the required data for the interface, it is necessary to establish a connection to the MQTT server and subscribe to the appropriate topics. This is achieved within the same component by implementing an additional hook. Since the communication between the dashboard and the broker utilizes WebSockets, the WebSocket port is required on the dashboard side. This is found along with the rest of the required credentials on the cloud.MQTT server details site. Two of the desks on the dashboard are programmed to react dynamically based on the MQTT messages, serving as a test to ensure that the dashboard can manage communications across multiple topics. If the MQTT message is ’1’, these desks change color to red and exhibit an ’X’. If the message is ’2’, they shift to yellow and display an error message. In both scenarios, they reset to the original green color after 5 minutes. The hook is configured to run at a 10-second interval.

Listing 5.4 shows the algorithm for drawing the upper left desk. It features if-statements for when an occupancy or error message is received.

```
1
2         // Desk #1 with MQTT connection
3         ctx.lineJoin = "bevel";
4         ctx.lineWidth = 2;
5         ctx.strokeStyle = "black";
6         ctx.fillStyle = message1.trim() === '1' ? 'red' : (message1.trim()
              === '2' ? 'yellow' : '#A2E47A');
7         ctx.fillRect(canvas.width/60, canvas.height/24, canvas.width/12,
              canvas.height/6);
8         ctx.strokeRect(canvas.width/60, canvas.height/24, canvas.width/12,
               canvas.height/6);
9
10        if (message1.trim() === '1') {
11          //Drawing X
12          ctx.strokeStyle = "black";
13          ctx.lineWidth = 10;
14          ctx.beginPath();
15          ctx.moveTo(canvas.width/60, canvas.height/24);
16          ctx.lineTo(canvas.width/12 + canvas.width/60, canvas.height/6 +
                canvas.height/24);
17          ctx.moveTo(canvas.width/12 + canvas.width/60, canvas.height/24);
18          ctx.lineTo(canvas.width/60, canvas.height/6 + canvas.height/24);
19          ctx.stroke();
20        } else if (message1.trim() === '2') {
21          // writing "ERROR" text
22          ctx.fillStyle = "black";
23          ctx.font = "bold 16px Arial";
24          ctx.fillText("ERROR: Sensor data not recieved", canvas.width/20,
                canvas.height/8);
25        }
```

Listing 5.4: The algorithm for drawing the upper left desk in the study hall

The function containing the two hooks is rendered and exported so it can be utilized in the function running the homepage. This script can be found in Appendix B and requires the command ”npm run dev” in the terminal to display the website. Lastly, in the homepage function, a welcome message was written above the room layout. The finished dashboard interface can be seen in section 6.3.

Chapter 6

Results and Analysis

In this chapter, the result of the implemented system is presented. The resulting functionality will be subject to analysis and will be further discussed in chapter 7.

6.1 Hardware

The hardware setup is functioning as a temporary setup for the proof-of-concept. The sensor successfully captures environment measurements with reasonable accuracy, although there is a slight deviation from the actual temperature, and sends the data via the I2C protocol to the ESP32. The WiFi module is also functioning as intended, connecting to the Eduroam network and transmitting data. However, the initial appearance of the setup is somewhat disorganized due to the wiring on the breadboard and the utilization of an Arduino as a 5V power source. In addition, the fact that both microcontrollers are connected to a PC makes the system immobile in its current state.

6.2 Operation and Communication

The functionality of the system is mainly based on the embedded software and the web dashboard software. It is working according to the flowcharts depicted in subsection 4.2.3 and subsection 4.2.1. The sensor system accurately deploys its functionality according to the time of day and utilizes its sleep mode in 5-minute intervals between measurements. Despite fluctuations in the connection time to the network and MQTT broker which affect the timing of the sleep period occurring after 1800 hours, the system accurately starts up again at 0700 in the morning.

The communication works as intended from the ESP32 to the broker and again to the interface. The microcontroller manages to publish the necessary messages in the form of signaling occupancy and sensor error. The choice of an external MQTT broker for a monthly fee is an expensive solution long term. However, it turned out to be an effective decision for a proof-of-concept as it demonstrates the capabilities of the MQTT protocol without demanding too much development and maintenance.

It is worth noting that the data logged on the MQTT broker remains stored while the website displaying the log remains active. This deviates to some degree from the intended specification of not storing any data. Although the data is not being stored on a server and is cleared the moment the website is terminated, this process inadvertently creates an opportunity for potential unwanted surveillance. Figure 6.1 shows the website displaying the log of the MQTT broker with three instances of the sensor detecting occupancy.

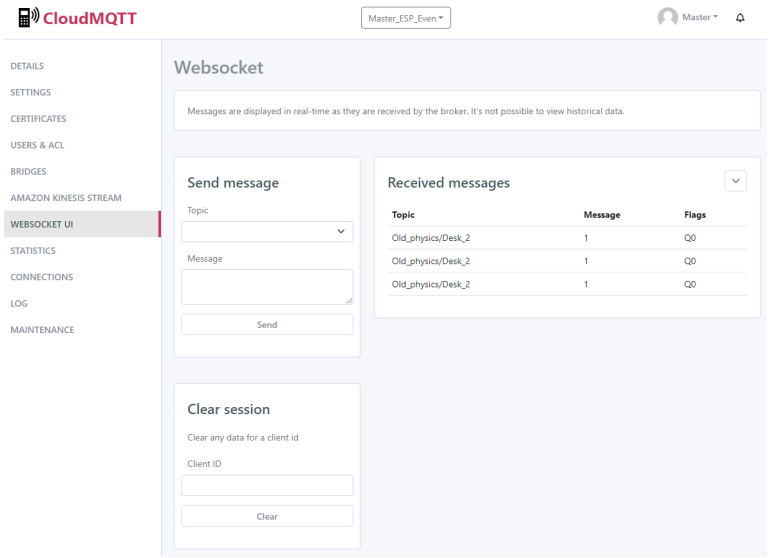


Figure 6.1: The website displaying the log of the MQTT broker

Since no measurements have been carried out on the system's current consumption, the power-saving measures' specific impact can not be determined. However, it is reasonable to assume that the utilization of the ESP32 sleep modes contributes significantly towards this, as can be seen from the properties presented in section 2.2.1.

6.3 User Interface

The finished dashboard interface was created with few changes to the original design, as explained in section 5.4. Figure 6.2 presents the website's appearance when the second

desk on the left side is occupied, while Figure 6.3 presents the same website when the sensor data is not received by the microcontroller.

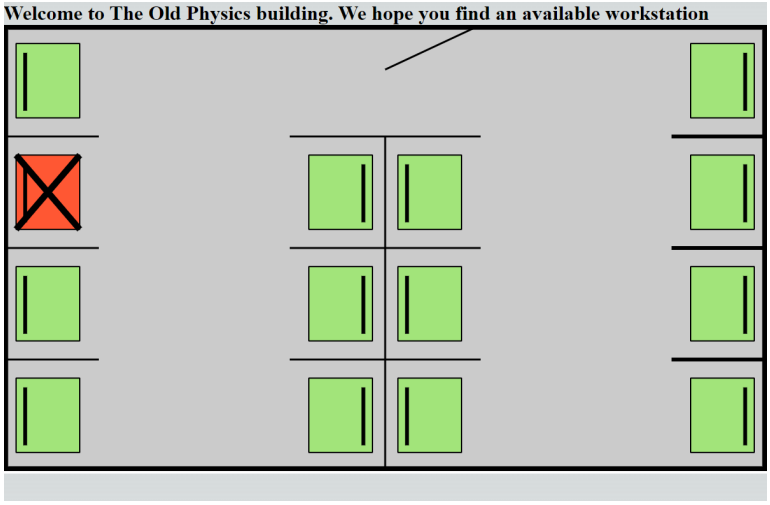


Figure 6.2: The dashboard interface, with one workstation being occupied

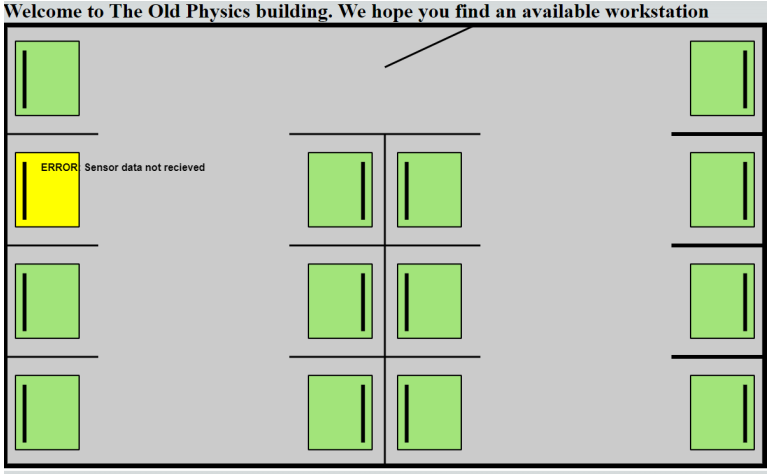


Figure 6.3: The dashboard interface, with sensor error in one workstation

The website's design and functionality are straightforward, as it was designed with the sole purpose of displaying the availability of workstations to students. Per this thesis, only the two upper left desks are interactive, responding to input from their respective MQTT topics. However, modifying the rest of the desks is easily applicable.

6.4 Properties of Cost and Scalability

The total cost of the system considers a one-month subscription to Cloud.MQTT and disregards tax and shipping costs were 332kr. This exceeds the initial cost threshold by 32kr. Although this is not extensive, it will increase even further when the MQTT subscription fee continues over several months. The implications of this can lead to a system that is too expensive to deploy, and measures, therefore, need to be considered. This is discussed in section 7.3

Every part of the system is subject to scalability per this thesis without considering the cost of the MQTT subscription. The hardware is replicable when it comes to both schematics and embedded software. The current MQTT broker subscription is capable of up to 25 clients, although this limit can be increased by upgrading the subscription. The interface features a room containing 14 workstations. However, expanding the website to include additional rooms is readily achievable by utilizing the React application.

Chapter 7

Discussion

Even though the result of the development resulted in a functioning proof-of-concept, the system is not necessarily ideal. Many of the specifications and design choices might seem flawed in hindsight. These decisions will be debated in this chapter, and the result will be compared to similar systems mentioned in chapter 2.1.1. Lastly, solutions for the future development of a prototype and final product will be presented and discussed.

7.1 Choice of Components

The type of sensor chosen was thoroughly evaluated as explained in subsection 3.2.2, and after implementation, nothing has refuted the thermopile sensor as the most practical choice. However, given the deviations measured in the specific sensor in this thesis, it is not necessarily the best choice of thermopile sensor. The D6T temperature sensor subject to this thesis is the ”1A-01”. The D6T line of sensors features other types that provide more accurate measurements. These are, however, more expensive and would exceed the cost limit suggested in section 3.1.3. The price range of these sensors is comparable to that of camera sensors, which were discarded, partially because of their price.

The ESP32 performed effectively in line with its intended functionality. However, the inclusion of a level shifter to convert 3.3V signals to 5V signals, as well as the requirement for a 5V power supply, contributed to complicating the circuit. This could be avoided by choosing a microcontroller with a 5V input. Arduino provides a number of microcontrollers that fulfill this desire, as mentioned in subsection 3.2.3. The purchase and implementation of a WiFi module need to be weighed against the complication caused by different voltage levels. Nonetheless, there exist special Arduino boards with wifi modules included, although these are more costly than the aforementioned cost threshold would allow (Arduino, 2023).

A solution for simplifying the circuit without changing the microcontroller would be to design a PCB (Printed Circuit Board). This would allow the schematics to take up significantly less space, in addition to giving the physical module a clearer design. Furthermore, for the capability of scaling, a PCB allows for the circuit to be easily produced in numbers.

7.2 Error Detection

Although some error detection is included for when the sensor data is not received, additional measures could have been implemented. If the sensor value turns out faulty, there are no procedures in place for notification. This issue can be addressed by introducing upper and lower thresholds to determine acceptable sensor values within the range of the room temperature and body temperature.

A failure in the MQTT and network connection will also be unnoticed by the dashboard interface. However, to solve this, data must be expected from the interface side even when the occupancy is set to available. A solution to this could be the ESP32 sending a message to the web dashboard in the function that adjusts time and goes to sleep once a day. If the message is not received, the dashboard will display an error notice.

7.3 MQTT Broker

Even though the choice of an external solution can display the functionality of the proof-of-concept, a local solution like Mosquitto would be a better choice with future development in mind. It would not require the cost of a monthly subscription which long term would significantly reduce the expenses of operating the system. In addition, letting a third party handle sensitive data might have some security concerns that are not fully addressed in this thesis. A local server would allow the deployer to manage the data, which could result in a more transparent data-handling process.

Although some setup is needed for a local server, more research could have been conducted to assess the time investment in relation to the potential benefits.

7.4 Current Consumption

Performing measurements for the current consumption is something that would be beneficial to do during the testing of the hardware. It would have given a pointer towards how much and which functionality should be adjusted and possibly changed. Even though the current consumption is uncertain, power-saving measures beyond what is implemented could have been relevant for this thesis.

One major issue is that the sensor performs measurements without the microcontroller requesting data. Since the sensor has an independent power supply, this can be solved by integrating a power control circuit and connecting it to the ESP32. By utilizing a switching component such as a transistor in series with the power supply, the microcontroller can be programmed to deactivate it when measurements are not required.

The choice of time step between measurements is a critical factor that impacts the overall power consumption of the system. A longer time step reduces the frequency at which the system performs measurements, connects to the network, and publishes data to the MQTT broker, resulting in potential power savings. Hence, prioritizing a longer time step is favorable from a power-saving standpoint. However, the decision to set the time step to 5 minutes was primarily based on the assumption that this delay would be acceptable to the students. It is important to note that this assumption may not necessarily hold true in practice. Therefore, conducting further research would be beneficial in order to optimize the time step between measurements, taking user preferences into account to strike a balance between power efficiency and acceptable measurement intervals.

An additional power-saving measure that would be relevant is deactivating the system on days with little to no activity on campus, specifically on Saturdays and Sundays. This can be accomplished by obtaining the current weekday information from the NTP along with the time in the ”adjustTimeAndSleep” function. By incorporating an if statement, the system can be programmed to enter a sleep mode over the weekend, starting from Friday. However, it is important to acknowledge that during examination periods, there may be a high demand for system operation, even on weekends, necessitating some consideration. Lastly, the system should also account for university breaks during the summer, Christmas, and Easter, adjusting the time management accordingly.

7.5 Comparisons to Similar Systems

Some of the products mentioned in subsection 2.1.1 feature sensor systems that have undergone comprehensive development, like the desk occupancy sensor by Disruptive Technologies (DisruptiveTechnologies, 2022). The quality of the product in terms of practicality, effectiveness, and power utilization is superior to the product developed in this thesis, and a direct comparison would be unfair. Despite the higher price range associated with these systems, it is important to consider whether the benefits they offer outweigh the associated costs. Conducting a comprehensive cost-benefit analysis of the various suggestions presented in the preceding sections is therefore strongly recommended before embarking on further development endeavors.

7.6 Future Work

7.6.1 Power Supply

Both the ESP32 and the sensor side of the level shifter require a voltage supply. To be able to have a portable system that can easily be set up at workplaces, a battery is needed to supply power to these components. Since the current consumption of the system per this thesis is uncertain, choosing an appropriate battery would require measuring the power usage of the sensor and microcontroller during operation. Although some indication of the consumption can be found by studying the datasheets of the sensor and microcontroller, the use of more precise methods, like a shunt resistor to measure the current, would deliver more accurate results.

The communication over WiFi is the operation that draws the most amount of power, and since the connection time varies from measurement to measurement, the consumption will also vary. It would therefore be useful to calculate an average to be able to estimate the current consumption in a day.

The decision for the type of battery also requires considerations towards easy maintenance. The people most likely to perform the task of changing the batteries are the general maintenance staff on campus, e.g. cleaning personnel or janitorial staff. These professions are not expected to have experience with hardware systems, and an easily obtainable and changeable battery is therefore recommended. A 9V battery would be an applicable solution. These batteries generally have a current capacity of around 1000mAh. If a higher capacity is required for the system to run for a reasonable period of time, options like Lithium-Ion or Lithium-Polymer batteries could be considered. However, again, these batteries require a higher degree of maintenance.

If the hardware solution is continued, linear voltage regulators are recommended for the battery to deliver the 5V and 3.3V to the ESP32 and sensor, respectively. This way the output voltage would be held at a constant level. Alternatively, a resistor divider could be chosen, but this is an inefficient solution that would drain the battery quickly. Figure 7.1 shows a theoretical circuit from a 9V battery to the 5V input using the LM7805 linear voltage regulator (HTC, 2022). A capacitor on both sides of the regulator makes sure the circuit delivers a constant voltage. This circuit could also be included in a potential PCB.

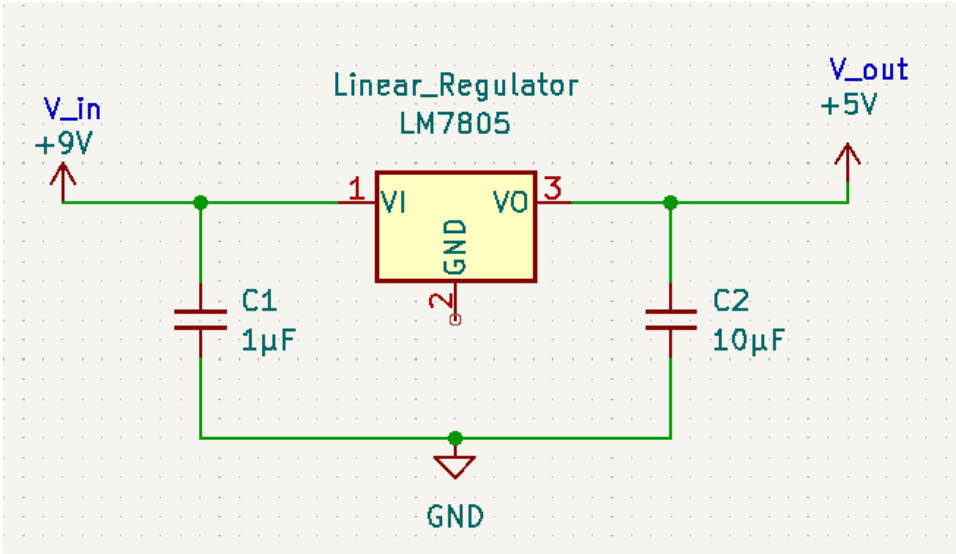


Figure 7.1: A sketched theoretical circuit transforming a 9V input into a 5V output.

For power-saving measures beyond what is suggested in the previous section, it might be necessary to look at alternative hardware. Although the sleep mode properties of the ESP32 are effective and similar in current consumption to many of their competitors, the WiFi module draws a significant amount of current. The datasheet estimates the consumption to be between 180-240mA for WiFi transmission (EspressifSystems, 2023c, p.48). A power-saving alternative to utilizing WiFi is Bluetooth. Although still a significant current consumer, Bluetooth utilizes an estimated 130mA on the ESP32. However, other microcontrollers have the potential to consume even lower amounts. For instance, the Nordic Semiconductor's nRF52840 dongle consumes between 3.83mA and 16.40mA when transmitting over Bluetooth (NordicSemiconducter, 2021, p.61).

Nevertheless, utilizing microcontrollers with only Bluetooth capabilities would render it impossible to establish communication via the MQTT protocol directly. A solution to this could involve setting up a local Bluetooth server in each room to manage transmissions from individual desks and forward them to the WiFi network using the MQTT protocol. However, implementing this solution would entail significant setup and logistical efforts compared to the solution proposed in this thesis.

7.6.2 Casing Design

The design of the casing encompassing the electronic components plays a crucial role in ensuring the system's accessibility and straightforward implementation at workstations. Additionally, as presented in subsection 3.1.3, an essential aspect of the setup is to notify users about the monitoring process. This can be accomplished simply by attaching an

informational sticker on the desks.

The physical module's design should consider not obstructing users' activities in the work environment and should occupy minimal space. Compactness is important since the system incorporates two relatively large components; a microcontroller and a battery. Developing a PCB to replace the existing schematics would aid in achieving a compact setup.

Furthermore, the casing should facilitate an unobstructed line of sight for the sensor and provide easy access for battery replacement. Figure 7.2 presents a theoretical sketch of a casing for a system utilizing a 9V battery. It features platforms on each side to accommodate the sensor and battery, while the central housing contains all the necessary electronics for the operation, including the ESP32, wires, and other essential components.

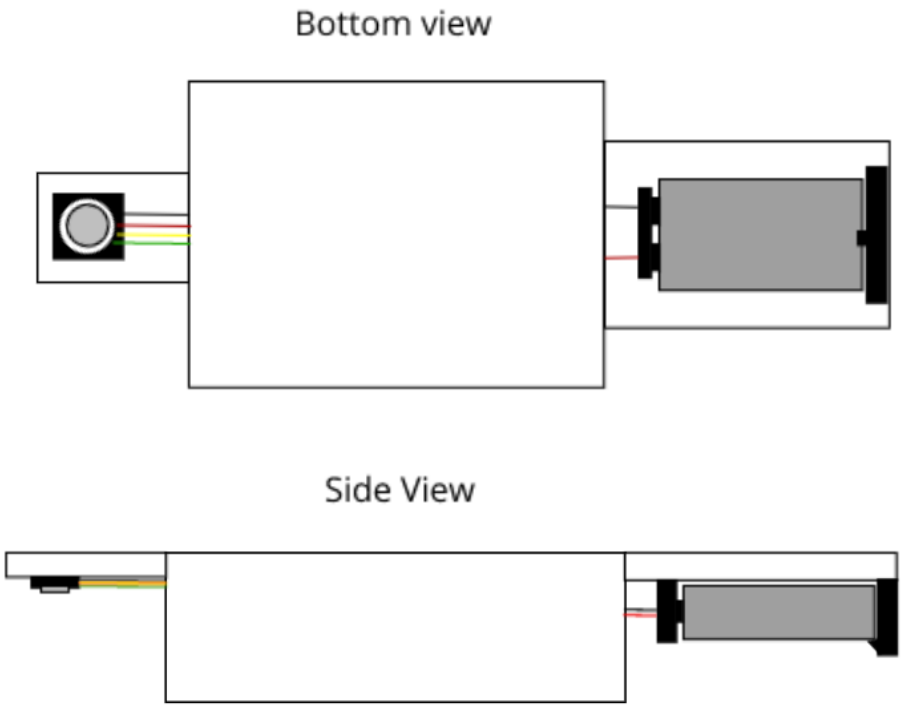


Figure 7.2: An example sketch of the casing for the physical module

For the development of the casing, several methods can be considered. However, based on the flexibility in design and cost-effectiveness, 3D printing is recommended. By utilizing Computer-Aided Design (CAD) programs, such as Fusion 360, which is available for all students and employees at NTNU, an STL file can be generated for the design. The file can then be used by a 3D printer to generate the physical component in plastic. Although

several types of plastics are available for 3D printing, a recommendation would be to use PLA (Polylactic Acid) as it is a cheap and biodegradable alternative.

7.6.3 Scaling and Further Development of the Web Dashboard

The development of the system for large-scale implementation depends on the scalability of the components and structure. The schematics per this thesis are replicable and the development of a PCB, as suggested, makes it easy to order the circuit component in bulk along with sensors and microcontrollers. The physical casing described in the previous subsection can be manufactured locally by 3D printers. However, for a large number of products, it might be beneficial to use the STL file to place an order from a large-scale manufacturer.

For the solution with a local MQTT broker, a large number of clients must be accommodated. This could be done by setting up a server where the system is utilized in each room. By developing the MQTT server system on a microcontroller, like a raspberry pi (RaspberryPi, 2023), the system can be easily replicated.

A website domain is needed for hosting the web dashboard, which requires a name for the final website. Also, further development is required for the web dashboard interface to enable the representation of multiple rooms. However, this is readily achievable, as mentioned in section 6.4. One potential approach is to design the homepage of the website to display a list of study halls on campus that utilize the system. Upon clicking on a specific study hall, users would be directed to a dedicated page displaying the corresponding room layout and individual desks' occupancy status.

An alternative approach that focuses on visual representation involves creating a campus map highlighting the rooms where the system is operational. Users would be able to click on these highlighted rooms to access the corresponding room layout. This is, nonetheless, a solution that requires more comprehensive front-end development. MazeMap has already developed campus maps for orientation purposes (MazeMap, 2023). Exploring the possibility of partnering with or integrating into their existing systems could offer users a comprehensive and seamless experience combining navigation and real-time occupancy status in study halls.

Chapter 8

Conclusion

The possibility of low-cost, real-time desk monitoring systems aimed at university campuses is proved with the proof-of-concept developed in this thesis. The system accurately detects human presence at workstations and displays the information on an intuitive interface through a web dashboard. The practice of anonymization and non-storage of data, along with a transparent design sketch, addresses the privacy concerns typically related to monitoring systems.

However, the proof-of-concept should be viewed as an early step in the system's development, with several opportunities for improvement in cost and practicality. The development of a prototype, and potential finalized product, will need to address these limitations, especially the development of a battery solution, a physical casing, and a cost-effective communication setup.

Relevant theory on desk monitoring systems suggests that most current solutions are more advanced when it comes to functionality and overall design. However, the price range and target group deviate from the system subject to this thesis. Therefore, the objective of the system might fill a gap in the market for student-centric logistic solutions. Hence, the further development of the proof-of-concept is encouraged, although it requires a cost-benefit analysis and a review of the suggested adjustments in chapter 7.

Bibliography

Arduino, 2023. *ARDUINO UNO WiFi REV2*.
https://store.arduino.cc/products/arduino-uno-wifi-rev2

Ball, K., 2010. *Workplace surveillance: an overview*.
https://www.tandfonline.com/doi/full/10.1080/00236561003654776

Bonta, R., 2023. *California Consumer Privacy Act (CCPA)*.
https://oag.ca.gov/privacy/ccpa

Boothman, P., 2022. *Introducing The Wireless Desk Occupancy Sensor*.
https://www.disruptive-technologies.com/blog/introducing-the-wireless-desk-occupancy-sensor

Campbell, S., 2016. *Basics of the I2C communication protocol*.
https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol

Chlebovec, M., 2023. *ESP32-eduroam*.
https://github.com/martinius96/ESP32-eduroam

Circuito, 2018. *Everything you need to know about Arduino code*.
https://www.circuito.io/blog/arduino-code/

CloudMQTT, 2023. *FAQ*.
https://www.cloudmqtt.com/docs/faq.html

DevicePlus, 2020. *Arduino: The Popular Microcontroller Board! Its History and to How to Use It*.
https://www.deviceplus.com/arduino/arduino-the-popular-microcontroller

DisruptiveTechnologies, 2022. *Wireless Desk Occupancy Sensor - Product Datasheet*.
https://www.disruptive-technologies.com/hubfs/Wireless%20Desk%20Occupancy%20Sensor%20-%C2%A0Product%20Datasheet%201.0.%20(1).pdf

DisruptiveTechnologies, 2023. *Sensor Starter Kit*.
https://store-no.disruptive-technologies.com/products/sensor-starter-kit

Eclipse, 2023. *Eclipse Mosquitto™ An open source MQTT broker*.
https://mosquitto.org/

EPRS, 2020. *The impact of the General Data Protection Regulation(GDPR) on artificial intelligence*.
https://www.europarl.europa.eu/RegData/etudes/STUD/2020/641530/EPRS_STU(2020)641530_EN.pdf

EspressifSystems, 2022. *ESP32 Technical Reference Manual*.
https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

EspressifSystems, 2023a. *About Espressif*.
https://www.espressif.com/en/company/about-espressif

EspressifSystems, 2023b. *API Guides - Memory Types*.
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/memory-types.html

EspressifSystems, 2023c. *ESP32 Series datasheet*.
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

HTC, 2022. *LM7805 Datasheet*.
http://www.htckorea.co.kr/Datasheet/CMOS%20ULDO/LM8805.pdf

IETF, 2011. *The WebSocket Protocol*.
https://datatracker.ietf.org/doc/html/rfc6455

IOTSpot, 2023. *The benefits of desk monitoring*.
https://www.iotspot.co/services/desk-monitoring

Lazar, V., Siegel, R., König, C. J., 2022. *The impact of electronic monitoring on employees' job satisfaction, stress, performance, and counterproductive work behavior: A meta-analysis*.
https://www.sciencedirect.com/science/article/pii/S2451958822000616?via%3Dihub

Maiti, A., Ye, A., Schmidt, M., Pederson, S., 2022. *A Privacy-Preserving Desk Sensor for Monitoring Healthy Movement Breaks in Smart Office Environments with the Internet of Things*.
https://www.mdpi.com/1424-8220/23/4/2229

MazeMap, 2023. *NTNU - Gløshaugen*.
https://use.mazemap.com/#v=1&zlevel=1&center=10.403905,63.416555&zoom=15.1&campusid=1

Meta, 2023. *React, The library for web and native user interfaces*.
https://react.dev/

Microsoft, 2023. *What is TypeScript?*
https://www.typescriptlang.org/

NordicSemiconducter, 2021. *nRF52840 Dongle Product Specification, V1.7*.
https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.7.pdf

Oasis, 2019. *MQTT Version 5.0, OASIS Standard*.
https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901000

Omron, 2018. *D6T User Manual*.
https://components.omron.com/us-en/ds_related_pdf/A284-E1.pdf

Omron, 2022. *D6T data sheet*.
https://components.omron.com/eu-en/datasheet_pdf/A274-E1.pdf

Ongweso Jr, E., 2022. *'NO': Grad Students Analyze, Hack, and Remove Under-Desk Surveillance Devices Designed to Track Them*.
https://www.vice.com/en/article/m7gwy3/no-grad-students-analyze-hack-and-remove-under-desk-surveillance-devi

Quinn, B., Jackson, J., 2016. *Daily Telegraph to withdraw devices monitoring time at desk after criticism*.
https://www.theguardian.com/media/2016/jan/11/daily-telegraph-to-withdraw-devices-monitoring-time-at-desk-after-cri

RaspberryPi, 2023. *Raspberry Pi computers and microcontrollers*.
https://www.overleaf.com/project/63c015e1b6a34da03e7f59cf/detacher

Sikkandhar, M., Lim, R., 2021. *Pressure Sensor Substrate for Prolonged Sitting and Posture Monitoring*.
https://ieeexplore.ieee.org/abstract/document/9663903

Teixera, T., Dublon, G., Savvides, A., 2010. *A Survey of Human-Sensing: Methods for Detecting Presence, Count, Location, Track, and Identity*.
http://thiagot.com/papers/teixeira_techrep10_survey_of_human_sensing.pdf

Teja, R., 2021. *ESP32 Pinout — ESP-WROOM-32 Pinout*.
https://www.electronicshub.org/esp32-pinout/

Thakkar, M., 2020a. *Introducing React.js*.
https://link.springer.com/chapter/10.1007/978-1-4842-5869-9_2

Thakkar, M., 2020b. *Next.js*.
https://link.springer.com/chapter/10.1007/978-1-4842-5869-9_3

Vercel, 2023. *Getting started with Next.js*.
https://nextjs.org/docs

Appendices

Appendix A

Embedded Algorithm

Listing A.1: Embedded software algorithm written in the Arduino programming language

```
1  //libraries
2  #include <Wire.h> // I2C library
3  #include <WiFi.h> // WiFi library
4  #include "esp_wpa2.h" // wpa2 library for connections to Enterprise
       networks
5  #include <PubSubClient.h> // MQTT library
6  #include <esp_sleep.h> // ESP32 sleep modes library
7  #include <NTPClient.h> // NTP server library
8  #include <WiFiUdp.h> // UDP over WiFi Library
9  #include <time.h> // time library
10
11 //Eduroam credentials
12 #define EAP_ANONYMOUS_IDENTITY "********" // Username eduroam
13 #define EAP_IDENTITY "*******" // nickname@example.com, not always
       neccessary
14 #define EAP_PASSWORD "*******" //password for eduroam account
15 const char* ssid = "eduroam"; // eduroam SSID
16 const char* host = "arduino.clanweb.eu"; // webserver
17 String url = "/eduroam/data.php"; // URL to target PHP file
18 byte mac[6]; // ESP32 mac address
19
20 //D6T addresses and data length
21 #define D6T_ADDR 0x0A // D6T address
22 #define D6T_CMD 0x4C // D6T command to read data
23 #define DATA_LENGTH 16 // Length of data to be received
24
25 //Temperature treshhold
26 #define TRESHHOLD 245
27
28 //ESP32 sleep values
29 #define uS_TO_S_FACTOR 1000000ULL  // Conversion factor for micro seconds
       to seconds
30 #define sleepBetweenMeasurements  300  // 5 minutes in seconds
```

```
31
32 //Number of measurements from 0700 to 1800
33 #define maxNumberOfDailyMeasurements 122
34
35 //MQTT credentials
36 const char* mqttUser = "*****";  //MQTT user
37 const char* mqttPassword = "*******"; //MQTT password
38 const char* mqtt_server = "hairdresser.cloudmqtt.com"; //MQTT server
39 const int mqttPort = *****; //MQTT port
40 const char* mqtt_topic = "Old_physics/Desk_2"; //MQTT topic
41
42 //Defining the ESP client
43 WiFiClient espClient;
44 PubSubClient client(espClient);
45
46 //Configuring the ntp server based on the time-zone of Trondheim
47 WiFiUDP ntpUDP;
48 const char* ntpServer = "europe.pool.ntp.org"; // NTP europe server
49 const long  gmtOffset_sec = 3600;  //gmt+1
50 const int   daylightOffset_sec = 3600; //gmt+2(summer)
51
52 //Initialize the wakeup timer
53 int timeUntilWakeup = 0;
54
55 //The RTC memory
56 RTC_DATA_ATTR int measurementsPerformedToday = 0;
57 RTC_DATA_ATTR volatile bool first_boot = true;
58
59 //Setup function works as a loop since the sleep functions reboot the
       microcontroller
60 void setup(){
61   Serial.begin(115200);
62   Serial.println(" ");
63   Serial.println("firstboot flag is: " + String(first_boot));
64   //If statement for running the configuration only once
65   if (first_boot){
66     configureSystem();
67     first_boot = false;
68     Serial.println("finished booting");
69   }
70   performMeasurement();
71   Serial.println("measurements: " + String(measurementsPerformedToday));
72   // If the daily measurements indicate that the time is above 18:00,
         adjust time and sleep
73   if (measurementsPerformedToday == maxNumberOfDailyMeasurements){
74     Serial.println("sleep LONG");
75     adjustTimeAndSleep();
76   }
77   //Sleep for 5 minutes
78   Serial.println("sleep SHORT");
79   Serial.println(" ");
80   Serial.println("Hardcoded sleep mode boot data:"); //Can be disabled by
         wiring D15 to ground
81   esp_sleep_enable_timer_wakeup(sleepBetweenMeasurements * uS_TO_S_FACTOR)
         ;
82   esp_deep_sleep_start();
83 }
```

```
84
85 void performMeasurement(){
86   Serial.println("perform measurement begin");
87   //initialize I2C
88   Wire.begin();
89   // Buffer to hold received data
90   uint8_t data[DATA_LENGTH];
91   //Beginning I2C communication and requesting the D6T sensor for data
92   Wire.beginTransmission(D6T_ADDR);
93   Wire.write(D6T_CMD); //
94   Wire.endTransmission();
95   Wire.requestFrom(D6T_ADDR, DATA_LENGTH);
96
97   String message;
98   if(Wire.available() >= DATA_LENGTH) {
99       // Read data received frmo the sensor
100      for(int i=0; i<DATA_LENGTH; i++) {
101        data[i] = Wire.read();
102      }
103      int temperature = data[0] + (data[1] << 8);
104      Serial.print("Temperature data from D6T: ");
105      Serial.println(temperature);
106      //check if temperature is above treshhold and set message value
             accordingly
107      if (temperature <= TRESHHOLD){
108        message = "0";
109      }
110      else{
111        message = "1";
112      }
113      //Set error message
114    } else {
115      message = "2";
116      Serial.println("Error: Data not received");
117  }
118  // Publish error message or desk availability to MQTT server
119  if (message == "1"  message == "2"){
120    WiFiConnect();
121    client.setServer(mqtt_server, mqttPort);
122    while (!client.connected()) {
123      if (client.connect("ESPclient", mqttUser, mqttPassword )) {
124      client.publish(mqtt_topic, message.c_str());
125      } else {
126      Serial.print(client.state());
127      delay(5000);
128      }
129    }
130    //disconnect from MQTT server and wifi
131    client.disconnect();
132    WiFi.disconnect(true);
133  }
134  //Wait for any ongoing operations to completeand and increment the daily
         measurement counter
135  delay(100);
136  ++measurementsPerformedToday;
137 }
138
```

```
139 void loop(){
140   //Setup function works as a loop. The void loop is therefore unused
141 }
142
143 void adjustTimeAndSleep(){
144   Serial.println("Adjust time and sleep begin");
145   //Adjust RTC after current clock from NTP server
146   WiFiConnect();
147   configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
148   delay(5000);
149   WiFi.disconnect(true);
150   //Set timeUntilWakeup to time until 07:00
151   struct tm timeinfo;
152   int hour = timeinfo.tm_hour;
153   int minute = timeinfo.tm_min;
154   int second = timeinfo.tm_sec;
155   int totalSeconds = (hour * 3600) + (minute * 60) + second;
156   timeUntilWakeup = (13 * 3600) - (totalSeconds - (18 * 3600));
157   //Reset the measurementsPerformedToday and sleep until 07:00
158   measurementsPerformedToday = 0;
159   Serial.println(" ");
160   Serial.println("Hardcoded sleep mode boot data:");
161   esp_sleep_enable_timer_wakeup(timeUntilWakeup * uS_TO_S_FACTOR); //(
          timeUntilWakeup*uS_TO_S_FACTOR);
162   esp_deep_sleep_start();
163 }
164
165 void configureSystem(){
166   //Adjust the RTC to the NTP server time
167   WiFiConnect();
168   configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
169   delay(5000);
170   WiFi.disconnect(true);
171
172   //Set totalseconds to seconds since 24:00
173   time_t now = time(NULL);
174   struct tm *timeinfo = localtime(&now);
175   int hour = timeinfo->tm_hour;
176   int minute = timeinfo->tm_min;
177   int second = timeinfo->tm_sec;
178   int totalSeconds = (hour * 3600) + (minute * 60) + second;
179   Serial.print("This is the hour  ");
180   Serial.println(hour);
181   //Sleep until 07:00 if the time is above 18:00
182   if(hour <= 07){
183       timeUntilWakeup = (7 * 3600) - totalSeconds;
184       Serial.println("IF state 1 enabled");
185       Serial.println(timeUntilWakeup);
186       esp_sleep_enable_timer_wakeup(timeUntilWakeup * uS_TO_S_FACTOR);
187       esp_deep_sleep_start();
188     }
189     else if(hour >= 20){
190       timeUntilWakeup = (7 * 3600) + (24 * 3600) - totalSeconds;
191       Serial.println("IF state 2 enabled");
192       Serial.println(timeUntilWakeup);
193       esp_sleep_enable_timer_wakeup(timeUntilWakeup * uS_TO_S_FACTOR);
194       esp_deep_sleep_start();
```

```
195       }
196       //Calculate expected measurement count for the time of day
197       else{
198         measurementsPerformedToday = round((hour - 7.0) * 11);
199         Serial.print("IF state 3 enabled, Measurements: ");
200         Serial.println(measurementsPerformedToday);
201     }
202   }
203   void WiFiConnect(){
204     //Connect to network through WiFi
205     delay(10);
206     Serial.print(F("Connecting to network"));
207     WiFi.disconnect(true);
208     WiFi.mode(WIFI_STA);
209     WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_ANONYMOUS_IDENTITY, EAP_IDENTITY,
            EAP_PASSWORD);
210     while (WiFi.status() != WL_CONNECTED) {
211       delay(500);
212       Serial.print(F("."));
213     }
214   }
```

Appendix B

Dashboard Algorithm

Listing B.1: JavaScript code for the dashboard interface

```
//Importing necessary packages
const React = require('react');
const { useRef, useEffect, useState } = require('react');
const mqtt = require('mqtt');

//MQTT credentials
const options = {
  clientId: 'Even_windows',
  username: 'jqkljioe',
  password: 'N0UsHuLzxFOS'
};

function CanvasComponent() {

  //Defining two different messages for the two topics
 const canvasRef = useRef<HTMLCanvasElement>(null);
  const [message1, setMessage1] = useState('');
  const [message2, setMessage2] = useState('');


  useEffect(() => {
    const client = mqtt.connect('wss://hairdresser.cloudmqtt.com
    :36722', options); //MQTT server and websocket port

    client.on('connect', function() {
      console.log('Connected to MQTT broker');
      client.subscribe('Old_physics/Desk_1'); //Topic 1
      client.subscribe('Old_physics/Desk_2'); //Topic 2
```

```
    });

    client.on('message', function(topic, message) {
      console.log('Received message on topic', topic, ':', message
    .toString());
      if (topic === 'Old_physics/Desk_1') {
        setMessage1(message.toString());
      } else if (topic === 'Old_physics/Desk_2') {
        setMessage2(message.toString());
      }
    });

    const intervalId = setInterval(() => {
      setMessage1('');
      setMessage2('');
    }, 5*60*1000); // Time interval of 5 minutes to reset the
    messages

    return () => {
      clearInterval(intervalId); // Clear interval
    };
  }, []);




  useEffect(() => {
    const canvas = canvasRef.current;

    if (canvas) {
      const ctx = canvas.getContext('2d');
      if (ctx) {
        //Dimensions of the interface rectangle
        canvas.width = 1200;
        canvas.height = 700;

        // Drawing the rectangle
        ctx.lineJoin = "bevel";
        ctx.lineWidth = 15;
        ctx.strokeStyle = "black";
        ctx.fillStyle = "#CBCBCB";
        ctx.fillRect(0, 0, canvas.width, canvas.height);
        ctx.strokeRect(0, 0, canvas.width, canvas.height);

        // Drawing the lines
        // Middle line
        ctx.strokeStyle = "black";
        ctx.lineWidth = 3;
        ctx.beginPath();
        ctx.moveTo(canvas.width/2, canvas.height/4);
```

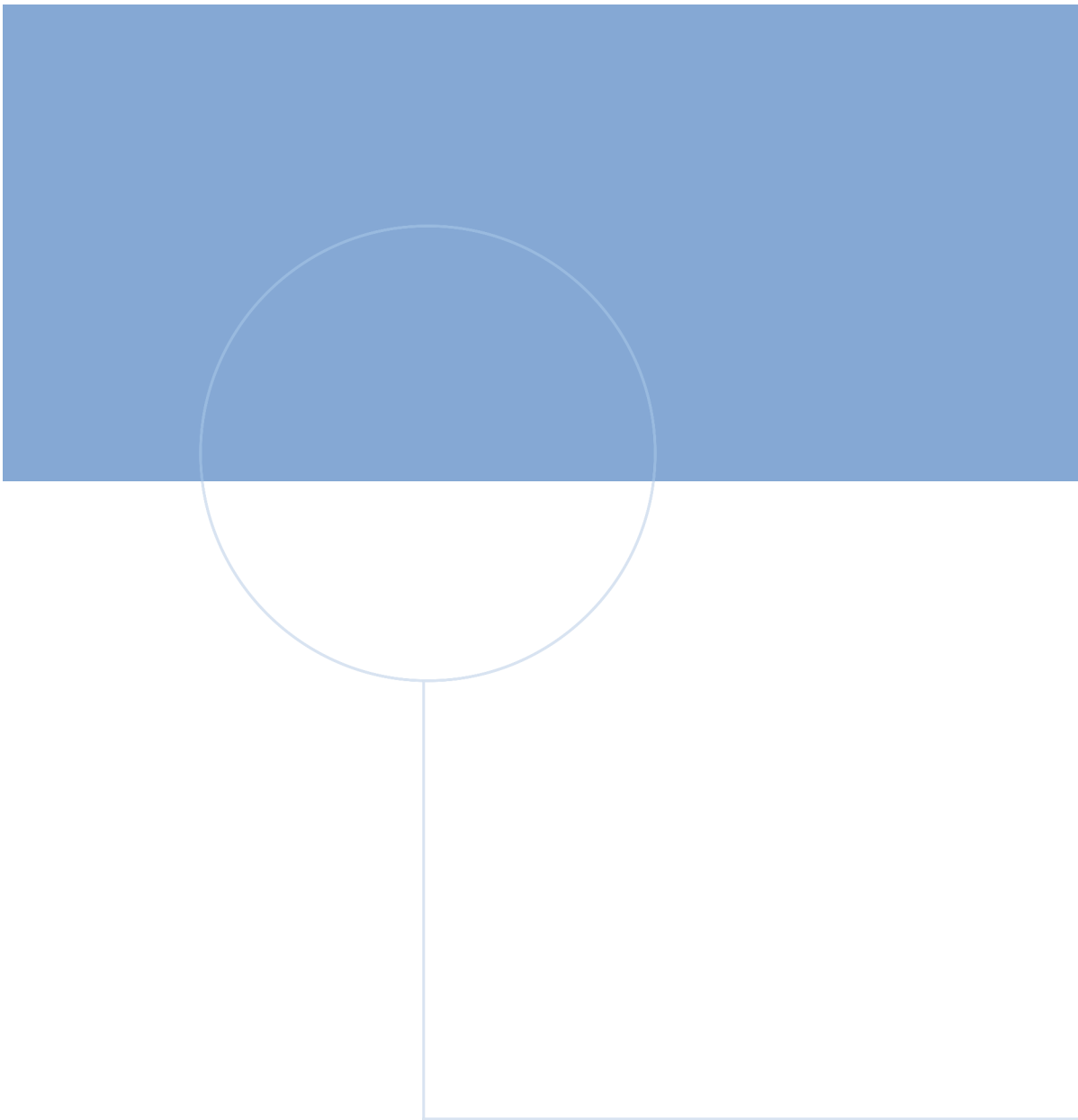
```
76          ctx.lineTo(canvas.width/2, canvas.height);
77          ctx.stroke();
78
79          ctx.strokeStyle = "black";
80          ctx.lineWidth = 3;
81          ctx.beginPath();
82          ctx.moveTo(canvas.width/2 + canvas.width/8, 0);
83          ctx.lineTo(canvas.width/2, canvas.height/10);
84          ctx.stroke();
85
86
87
88
89          // Desk #1 with MQTT connection
90          ctx.lineJoin = "bevel";
91          ctx.lineWidth = 2;
92          ctx.strokeStyle = "black";
93          ctx.fillStyle = message1.trim() === '1' ? 'red' : (
    message1.trim() === '2' ? 'yellow' : '#A2E47A');
94          ctx.fillRect(canvas.width/60, canvas.height/24, canvas.
    width/12, canvas.height/6);
95          ctx.strokeRect(canvas.width/60, canvas.height/24, canvas.
    width/12, canvas.height/6);
96
97          if (message1.trim() === '1') {
98            //Drawing X
99            ctx.strokeStyle = "black";
100           ctx.lineWidth = 10;
101           ctx.beginPath();
102           ctx.moveTo(canvas.width/60, canvas.height/24);
103           ctx.lineTo(canvas.width/12 + canvas.width/60, canvas.
    height/6 + canvas.height/24);
104           ctx.moveTo(canvas.width/12 + canvas.width/60, canvas.
    height/24);
105           ctx.lineTo(canvas.width/60, canvas.height/6 + canvas.
    height/24);
106           ctx.stroke();
107         } else if (message1.trim() === '2') {
108           // writing "ERROR" txet
109           ctx.fillStyle = "black";
110           ctx.font = "bold 16px Arial";
111           ctx.fillText("ERROR: Sensor data not recieved", canvas.
    width/20, canvas.height/8);
112         }
113
114
115
116
117
118         //Desk #2 with MQTT connection
```

```
119         ctx.lineJoin = "bevel";
120         ctx.lineWidth = 2;
121         ctx.strokeStyle = "black";
122         ctx.fillStyle = message2.trim() === '1' ? 'red' : (
    message2.trim() === '2' ? 'yellow' : '#A2E47A');
123         ctx.fillRect(canvas.width/60, canvas.height/24 + canvas.
    height/4,  canvas.width/12, canvas.height/6);
124         ctx.strokeRect(canvas.width/60, canvas.height/24 + canvas.
    height/4, canvas.width/12, canvas.height/6);
125         if (message2.trim() === '1') {
126           // Drawing X
127           ctx.strokeStyle = "black";
128           ctx.lineWidth = 10;
129           ctx.beginPath();
130           ctx.moveTo(canvas.width/60, canvas.height/24 + canvas.
    height/4);
131           ctx.lineTo(canvas.width/12 + canvas.width/60, canvas.
    height/6 + canvas.height/24 + canvas.height/4);
132           ctx.moveTo(canvas.width/12 + canvas.width/60, canvas.
    height/24 + canvas.height/4);
133           ctx.lineTo(canvas.width/60, canvas.height/6 + canvas.
    height/24 + canvas.height/4);
134           ctx.stroke();
135         } else if (message2.trim() === '2') {
136           //writing "ERROR" text
137           ctx.fillStyle = "black";
138           ctx.font = "bold 16px Arial";
139           ctx.fillText("ERROR: Sensor data not recieved", canvas.
    width/20, canvas.height/3);
140         }
141
142
143         //Draw the rest of the cubcles and desks
144         //Left desks
145         for (let i = 0; i < 4; i++) {
146
147           //Cubicles
148           ctx.lineWidth = 3;
149           ctx.beginPath();
150           ctx.moveTo(0, i*canvas.height/4);
151           ctx.lineTo(canvas.width/8, i*canvas.height/4);
152           ctx.stroke();
153
154           //Desks
155           ctx.lineJoin = "bevel";
156           ctx.lineWidth = 2;
157           ctx.strokeStyle = "black";
158           ctx.fillStyle = "#A2E47A";
159           ctx.fillRect(canvas.width/60, canvas.height/24 + i*
    canvas.height/4 + canvas.height/2, canvas.width/12, canvas.
```

```
    height/6);
160         ctx.strokeRect(canvas.width/60, canvas.height/24 + i*
    canvas.height/4 + canvas.height/2, canvas.width/12, canvas.
    height/6);
161
162         //Computers
163         ctx.lineWidth = 6;
164         ctx.beginPath();
165         ctx.moveTo(canvas.width/35, canvas.height/16 + i*canvas.
    height/4);
166         ctx.lineTo(canvas.width/35, canvas.height/5.2 + i*canvas
    .height/4);
167         ctx.stroke();
168       }
169
170       //Middle left desks
171       for (let i = 0; i < 4; i++) {
172
173         //Cubicles
174         ctx.lineWidth = 3;
175         ctx.beginPath();
176         ctx.moveTo(canvas.width/2, i*canvas.height/4);
177         ctx.lineTo(3*canvas.width/8, i*canvas.height/4);
178         ctx.stroke();
179
180         //Desks
181         ctx.lineJoin = "bevel";
182         ctx.lineWidth = 2;
183         ctx.strokeStyle = "black";
184         ctx.fillStyle = "#A2E47A";
185         ctx.fillRect(canvas.width - (canvas.width/60 + canvas.
    width/2 + canvas.width/12), 7*canvas.height/24 + i*canvas.
    height/4, canvas.width/12, canvas.height/6);
186         ctx.strokeRect(canvas.width - (canvas.width/60 + canvas.
    width/2 + canvas.width/12), 7*canvas.height/24 + i*canvas.
    height/4, canvas.width/12, canvas.height/6);
187
188         //Computers
189         ctx.lineWidth = 6;
190         ctx.beginPath();
191         ctx.moveTo(canvas.width/2 - canvas.width/35, canvas.
    height/16 + canvas.height/4 + i*canvas.height/4);
192         ctx.lineTo(canvas.width/2 - canvas.width/35, canvas.
    height/5.2 + canvas.height/4 + i*canvas.height/4);
193         ctx.stroke();
194       }
195
196       //Middle right desks
197       for (let i = 0; i < 4; i++) {
198
```

```
        //Cubicles
        ctx.lineWidth = 3;
        ctx.beginPath();
        ctx.moveTo(canvas.width/2, i*canvas.height/4);
        ctx.lineTo(5*canvas.width/8, i*canvas.height/4);
        ctx.stroke();

        //Desks
        ctx.lineJoin = "bevel";
        ctx.lineWidth = 2;
        ctx.strokeStyle = "black";
        ctx.fillStyle = "#A2E47A";
        ctx.fillRect(canvas.width/60 + canvas.width/2, 7*canvas.height/24 + i*canvas.height/4, canvas.width/12, canvas.height/6);
        ctx.strokeRect(canvas.width/60 + canvas.width/2, 7*canvas.height/24 + i*canvas.height/4, canvas.width/12, canvas.height/6);

        //Computers
        ctx.lineWidth = 6;
        ctx.beginPath();
        ctx.moveTo(canvas.width/2 + canvas.width/35, canvas.height/16 + canvas.height/4 + i*canvas.height/4);
        ctx.lineTo(canvas.width/2 + canvas.width/35, canvas.height/5.2 + canvas.height/4 + i*canvas.height/4);
        ctx.stroke();

    }



    //Right cubicles
    for (let i = 0; i < 4; i++) {

        //Cubicles
        ctx.beginPath();
        ctx.moveTo(7*canvas.width/8, i*canvas.height/4);
        ctx.lineTo(canvas.width, i*canvas.height/4);
        ctx.stroke();

        //Desks
        ctx.lineJoin = "bevel";
        ctx.lineWidth = 2;
        ctx.strokeStyle = "black";
        ctx.fillStyle = "#A2E47A";
        ctx.fillRect(canvas.width - (canvas.width/60 + canvas.width/12), canvas.height/24 + i*canvas.height/4, canvas.width/12, canvas.height/6);
```

```
240             ctx.strokeRect(canvas.width - (canvas.width/60 + canvas.
    width/12), canvas.height/24 + i*canvas.height/4, canvas.width
    /12, canvas.height/6);
241
242             //Computers
243             ctx.lineWidth = 6;
244             ctx.beginPath();
245             ctx.moveTo(canvas.width - canvas.width/35, canvas.height
    /16 + i*canvas.height/4);
246             ctx.lineTo(canvas.width - canvas.width/35, canvas.height
    /5.2 + i*canvas.height/4);
247             ctx.stroke();
248           }
249
250         }
251       }
252
253   }, [message1,message2]);
254
255   return <canvas ref={canvasRef} />;
256 }
257
258 export default CanvasComponent;
```



NTNU

Norwegian University of
Science and Technology