Development of a new expandable Andon light system into lean manufacturing production using MQTT and smart bulbs

Master of Science (Tech) Thesis
University of Turku
Department of Computing, Faculty of Technology
Robotics and Autonomous systems
2023
Kalle Hautamäki

Supervisors:
Prof. Tomi Westerlund
MSc. Kristian Väisänen

UNIVERSITY OF TURKU
Department of Computing, Faculty of Technology

Kalle Hautamäki: Development of a new expandable Andon light system into lean manufacturing production using MQTT and smart bulbs

Master of Science (Tech) Thesis, 52 p.
Robotics and Autonomous systems
September 2023

In an era of constant technological advancement, manufacturing industries are witnessing transformative changes. The traditional Andon system, once a cornerstone of lean manufacturing through visual and auditory cues, now faces the need for modernization in today's interconnected production facilities. This thesis explores the development and implementation of an advanced Andon light system in the context of in-house logistics and manufacturing. It addresses the need for an updated system to replace outdated hardware and software. Using MQTT as a lightweight communication protocol and smart bulbs with Zigbee technology, the research focuses on improving system efficiency and versatility. The study presents the development of a solution which offers the traditional Andon light system features with support for diverse applications.

This thesis relies primarily on experimental research to address the development of the Andon system. The research encompasses critical decisions regarding the choice of communication protocol, specifically among MQTT, HTTP, and WebSockets. Additionally, the selection of system hardware is examined, with a comparative analysis between conventional Andon light posts and smart bulbs. Each choice is meticulously explained based on its features and implications. Further on the development process and the architecture of the system is explained in-depth.

The system presented in this thesis prioritizes cost-efficiency, resulting in the utilization of two smart bulbs per manufacturing cell. This decision posed a significant challenge due to the complexities in managing colors within these two bulbs. While not directly applicable as traditional Andon lights, this thesis explores alternative applications for the system, emphasizing its potential for expansion within logistics and observability.

Keywords: Andon, Lean Manufacturing, Smart bulbs, MQTT, Zigbee, Zigbee2MQTT, Visual Guidance, IIoT

Contents

List of Figures

List of Tables

1 Introduction

New technologies such as artificial intelligence, machine networking, robotics, virtual reality, and additive manufacturing are changing the operational environment of production and logistics. These kind of technologies have not only accelerated production processes but have also entered the production industry into an era of connectivity and data-driven decision-making.

Amidst the rapid advancements in technology, one critical aspect that demands continous improvement is the Andon system - a tool that empowers production floor operators to make decisions, and identify issues using visual and auditory assistance. The traditional Andon was developed over 50 years ago, and in today's interconnected world, a more intelligent and ductile approach is required to meet the demands of modern manufacturing plants.

In response to these evolving needs, this thesis presents a solution to universalize the visual aspect of Andon using MQTT and smart bulbs. By incorporating these technologies, the proposed systems aims to make the Andon lights more versatile in different environments and situations, facilitating immediate communication even in difficult networks, and enabling real-time monitoring.

While the utilization and advantages of smart lighting systems in various settings are extensively discussed and researched, there remains a dearth of exploration from an industrial standpoint [1]. This gap in research highlights unexplored potential within industries, such as manufacturing and logistics, which could enhance

efficiency.

The research delves into the design, development, and implementation of this IoT-driven Andon light system, exploring different options for the communication protocols, options for the light platfroms, and the opportunities the system can provide. This thesis' objective is also to explore how to upgrade the Andon light system to deliver even greater value, reduce lead times, and create new opportunities for manufacturing plants.

1.1 Significance and Motivation

This thesis is a collaboration with Teleste Oyj, a network-related device manufacturer based in Finland. The aim of the thesis is to rethink the light controlling system within the company's internal logistics.

The Andon lights used at Teleste differ from the normal Andon lights in that they have five different colors instead of the original three. Blue and white colors have been added to the red, yellow, and green colors that were originally designed for the Andon lights, creating divergence in the visual guidance for the operators. This implies that there is potential for even more colors to guide production logistics and information. The introduction of more colors in the production facility opens up the possibility for conveying more information within the same visual framework. This has the potential to simplify in-house logistics and enhance production efficiency.

MQTT, being lightweight and using a publish-subscribe model for communication, provides a solid foundation in the production facility for data-driven decision-making due to its low-latency and high-predictability. MQTT is not only suitable for communicating with the lights but can be applied throughout the entire manufacturing process.

Smart bulbs that use Zigbee for communication are commonly found in homes and home automation systems. However, due to their use of Zigbee, which is a mesh

network, smart bulbs have numerous opportunities in production facilities that have yet to be fully researched. Employing smart bulbs for guidance can bring value to production operators through various scenarios and use cases. For example, they can provide immediate visibility into the capacity utilization rate of a warehouse location or the status of a machine or process. As the automation rate in production becomes mature enough, lights and colors might lose their value, but the data provided can still be beneficial for the automated system.

Research questions for this thesis are as follows:

- How can the Andon light system be developed to support other functions in lean manufacturing?
- How does the new Andon light system using MQTT and ZigBee compare to existing systems in terms of functionality, efficiency, and cost?
- Is it possible to create a well-functioning Andon light system with smart bulbs?

1.2 Related works

The topic of improving Andon lights is a niche and thus has not been discussed much. The research on benefits of the usage of smart bulbs is on the raise, and there are constantly more publications on the topic. Most commonly the publications analyze smart bulbs on energy savings, and the benefits of their usage in an office environment through the variability of the temperature of the illumination. Slowly the topic is further also stretched into the industrial scene of publications. The state of MQTT has been static for a time, but the amount of research on potential utilization targets for MQTT is still high. The nature of publish-subscribe model, and lightweightness of the protocol makes it a notable choice when it comes to choosing a communication protocol.

Lei, Lu, and Sang [2] designed a wireless Andon system using Zigbee. The system consisted of Andon communication, production monitoring, and statistics reports. The system was found to be scalable and very reliable in the production environment. The architecture of the designed system consisted of the main computer, Zigbee module, and Zigbee nodes, which work as the Andon calling devices. The paper focuses on the design of an entire Andon system, whereas this thesis focuses solely on Andon lights and light systems. Therefore the suggested design could benefit from what this thesis offers.

Yeh, Chen, and Li [3] designed and implemented an MQTT based communication architecture for a smart factory facility. The research contained setting up the MQTT network, and defining different layers of functionality. The functional layers consisted of motion control layer, edge computing layer, fog computing layer, and application layer. The system was designed to communicate with machines, have a webserver as an interface to humans, and do data processing and analysis in edge machines, to name a couple of examples. The paper suggests a common communication protocol into a smart factory where clients are mostly examples on how the system can be used, while this thesis is more focused on the client sided actions and how the communication protocol functions with the application. These two systems can benefit from each other bringing the end result closer to the ideal scenario.

1.3 Structure

The purpose of this document is to detail the design and implementation of an expandable Andon light system to manufacturing production using MQTT and smart bulbs, and the design and implementation of an application that controls smart bulbs with python programming language.

- Chapter 2 introduces the technologies and methodologies related to this thesis,

such as MQTT, ZigBee, and Lean manufacturing.

- In chapter 3, we explain the design process of the system step by step describing and evaluating each choice.
- Chapter 4 describes the software component, which is responsible in handling the features of the system.
- In chapter 5, we review the implementation of the system in a production environment, and report the results and findings from the experiment.
- Finally, chapter 6 concludes this work and outlines future work directions.

2 Background

This chapter provides information on the background needed to understand this thesis. The sections include information about the technologies used in this thesis and the state of the factory where the new expandable Andon light system is designed to.

2.1 Lean Manufacturing

Lean manufacturing is a production philosophy and set of principles that seeks to maximize value and minimize waste in the manufacturing process. It originated in the automotive industry in Japan in the 1950s and 1960s, and has since been adopted by many other industries around the world [4].

At its core, lean manufacturing is based on the idea that waste, or anything that does not add value from the customer's perspective, should be eliminated. This is achieved by identifying and reducing or eliminating non-value-added activities, such as overproduction, waiting, excess motion, excess inventory, defects, and unnecessary processing.

To implement lean manufacturing, companies often use a set of tools and techniques, such as value stream mapping, Kanban, 5S, and standardized work, to identify and eliminate waste, improve flow, and increase efficiency in their production processes. Lean manufacturing also emphasizes the involvement of all employees in continuous improvement efforts, and the use of visual management tools, such as

Andon lights, to support this effort.

Lean manufacturing has been shown to improve quality, reduce lead times, increase flexibility, and reduce costs in a variety of industries. It is often used in conjunction with other production management methodologies, such as Six Sigma and the Toyota Production System, to achieve even greater improvements in manufacturing performance [5]. In addition to its application in manufacturing, the principles of lean can also be applied to a wide range of other business contexts [4].

In general, lean is a philosophy and set of principles that focuses on maximizing value and minimizing waste in any type of business or organization. This can be achieved by identifying and eliminating non-value-added activities, improving flow, and involving all employees in continuous improvement efforts.

For example, lean principles can be applied to the design and delivery of services, such as healthcare, education, or government services, to improve the quality and efficiency of these services. Lean principles can also be applied to administrative or support functions, such as finance, human resources, or information technology, to reduce waste and improve the flow of information and processes within these areas.

In short, lean is a versatile and powerful approach that can be applied to a wide range of business contexts to improve performance, reduce waste, and increase value for customers.

2.2 Andon

Andon is a Japanese word for "light" or "lamp". It is also originally a part of the Toyota Production System where the status of the production line can be expressed through different colored lights [6]. Andon light systems are used to improve the efficiency and quality of manufacturing operations by providing real-time visual feedback on the status of the production process. They can help operators and supervisors identify and address problems quickly, reducing downtime and improving

overall productivity [4].

In addition to just status lights, Andon can be used with boards or cords. Andon boards are a measure of performace in production, where the target value is the amount of work that is supposed to be done during a shift, and the current value how much has been done so far.

Andon cords are a tool used in manufacturing settings to alert supervisors when immediate attention is needed. A worker can pull the cord, which turns on a light or plays sounds very near the working spot of a supervisor to inform them of required assistance.

Traditionally Andon lights work through switches or the manufacturing machine, however there are cases where Andon systems are made wireless with ZigBee [7].

Visual management can also help to foster a culture of continuous improvement by making it easy for workers to see opportunities for improvement and take action to address them. By making information and processes visible, visual management can help to engage and empower workers, encouraging them to take ownership of their work and identify ways to improve it [8][9].

Overall, visual management is valuable in manufacturing and other industries because it helps to improve communication, efficiency, and continuous improvement, which can lead to improved performance and increased value for customers.

2.3 MQTT

MQ Telemetry Transport (MQTT) is a low power, low profile communication protocol developed in 1999, and designed for embedded system devices in difficult and unreliable networks [10]. MQTT was originally developed for connecting oil pipelines to satellites by Andy Stanford-Clark and Arlen Nipper with the following requirements:

1. Simple implementation
2. Quality of Service data delivery
3. Lightweight and bandwidth efficient
4. Data agnostic
5. Continuous session awareness

MQTT slowly found its place in the IoT area and eventually became royalty-free for everyone to use in 2010 [11].

MQTT functions with a publish/subscribe principle, where clients either publish messages or subscribe to topics. Topics in turn are subject areas or message channels for clients to publish or subscribe to. MQTT does this in a way where the clients do not require the knowledge of other clients existence [12]. The way this works is that a publisher client sends a message into a topic of its own desire. After that, if a subscriber client has subscribed to that particular topic, it will receive the message. The amount of topics is not restricted by the architecture and both publishers and subscribers can use as many topics as they require.

In MQTT, topics are used to specify the destination for a published message. Topics have a hierarchical structure, with each level separated by a '/' character. For example, a topic might be structured as "sensor/temperature/office", with "sensor" being the top-level topic and "temperature" and "office" being subtopics. This hierarchical structure allows for a flexible and scalable organization of topics into trees [13].

MQTT also supports the use of wildcards in topic subscriptions. The '+' wildcard can be used to match a single level of a topic hierarchy, while the '#' wildcard can be used to match multiple levels. This allows subscribers to selectively receive messages based on the topics they are interested in.

MQTT is a lightweight protocol designed for efficient transfer of messages. One of the key features that contributes to its lightweight nature is the small size of the messages it transfers, with the header being just two bytes. MQTT is also a binary protocol, meaning that it encodes data in binary format rather than in plain text, which can make it more efficient and more compact than text-based protocols. In addition to its small message size and binary encoding, MQTT is also data-centric, meaning that data is transferred as a byte array and the content of the message is not interpreted or analyzed by the MQTT system. This allows MQTT to be flexible and easily adaptable to a wide range of use cases and applications [14].

In an MQTT network, clients and a broker form the core components. The broker serves as the intermediary between clients and facilitates the exchange of messages. It acts as the central hub for all communication, and its presence is imperative for the proper functioning of the network. Without a broker, the network would be inoperable, highlighting the criticality of its role in the overall system. An example of an MQTT network architecture can be seen in Fig. 2.1.

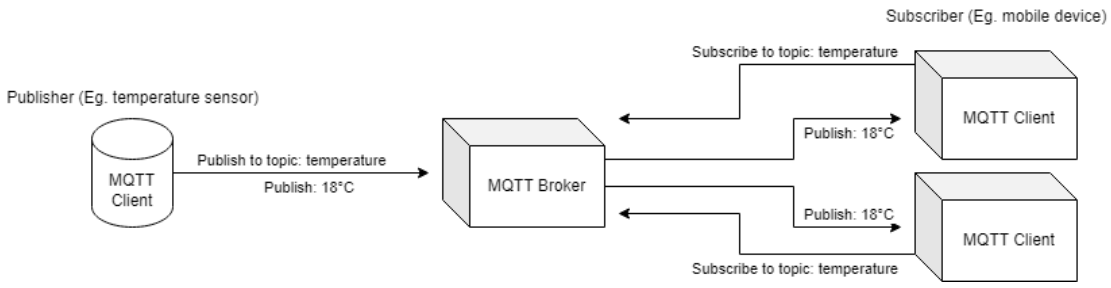

Figure 2.1: An example of an MQTT publish/subscribe architecture

MQTT offers several key features that enable efficient and effective communication within the network [13]. These features include:

- Quality of Service (QoS): There are three levels of Quality of Service in MQTT. Levels from 0 to 2 defines the message delivery insurance, level 0 being no ac-

knowledgement from the broker (fire-and-forget), level 1 being a confirmation of message reception, and level 2 being guaranteed delivery with a four-way mechanism. With each level the network bandwidth consumption is plausible to increase. Each client and message can be configured to have its own QoS based on their specific requirements and the nature of the data being transmitted.

- Message retention: MQTT includes support for retained messages, which means that a device can publish a message with a flag indicating that it should be stored by the broker and sent to any new subscribers that connect. This allows devices to receive the latest available data when they first connect to the broker, without missing any updates that may have occurred while they were offline.

  Retained messages are not perfect. Even if they deliver the last published message all of the rest are lost, and there can be lots of data that could have been useful. There are suggestions for improvement on this case where the broker counts and saves all messages into a queue for a client that has been disconnected. This queue can then be put to use having no data lost. [15]

- Wills: Wills are messages that clients publish whenever they connect or disconnect from the network. They can be beneficial for systems that use devices to function, for example a device can tell the system whenever they come online or go offline by updating the status through a specific topic [13].

Overall, the features MQTT provides enable lightweight, reliable communication for a wide range of use cases, making it an ideal choice for IoT and M2M applications [13].

Mosquitto

Eclipse foundation has a project called Mosquitto, which is a lightweight message broker server, available for any user under the Eclipse Public license. Mosquitto is a very common choice for an MQTT server because its ease of use, and its features like Transport Layer Security (TLS).

Mosquitto is not only a broker server, but it can also be installed to be used on a command line interface. This is an easy way to test and use MQTT servers as clients. In addition to this, Eclipse has provided a Python package called paho-mqtt which can be used to create clients. The paho-mqtt package provides all the necessary elements, even for more complicated use-scenarios.

Mosquitto can be run on a variety of platforms like Windows, Linux, and macOS, and it also has a docker image ready to use. [16]

2.4 Zigbee

Zigbee is a wireless low-range mesh network, that is used, among other things, for providing communication between sensor technology devices [17]. In ZigBee networks, each device can behave as a node to extend the network, so that even if a commanding device is further away than the connection would allow, the packet can be transmitted through other devices, what creates the mesh-network.

The name Zigbee comes from zigzagging honey bees that would fly around from one flower to another and to the hive, which represents the mesh network of the system.

There can be three types of nodes in a Zigbee network: a coordinator, a router and an end-device. The coordinator is the only device that can create a network, meaning that it is the one that can add other devices to the network mesh. There exists only one coordinator in a Zigbee network. Routers are nodes that can transmit

messages to other devices. It can also behave so that other devices connect to it when coming into the network. Routers are most commonly connected to mains, for example light bulbs or plugs, which allows the devices to consume enough power to handle the network traffic going through them. End-devices are the ones that only connect either to a coordinator or a router. They are the most passive of the three, and only work independently on one connection. These three types of nodes complete the Zigbee network, and an example can be seen in Fig. 2.2.

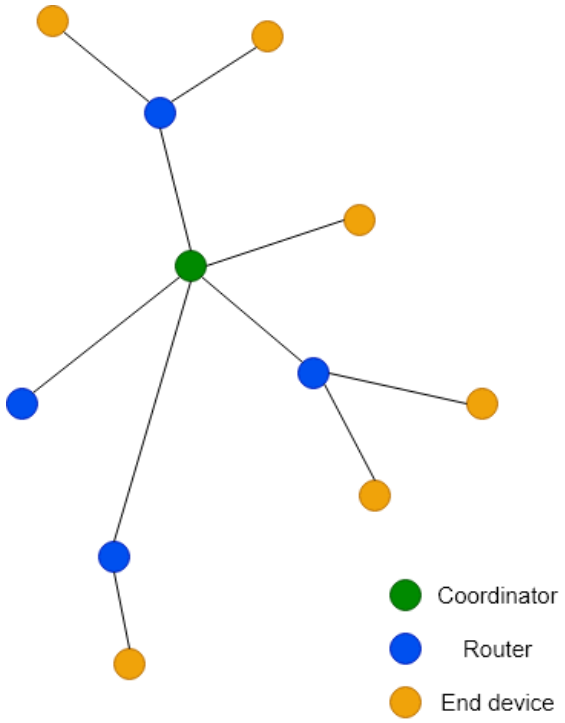

Figure 2.2: An example of a ZigBee network

Zigbee is relatively low-power, only requiring as low as 1 mW to function. Battery powered devices can be used for months allowing them to sleep while not communicating with any other device [18]. The device goes to low duty cycle and communicate only when there is a change of state, for example in a light switch [19].

The bandwidth used in Zigbee technology is either 868 MHz (European), 915 MHz (North American), or 2.4 GHz (worldwide) [20]. In areas with heavy WiFi traffic, the use of the 2.4 GHz bandwidth can result in significant interference. Zigbee

supports different channels for different bandwidths, and this interference can be mitigated by configuring the network to use a channel that falls within the channel range but outside the frequency range of a single WiFi band.

2.5 Software components

The development and implementation of the solution in this thesis utilize the following software technologies.

2.5.1 Python

Python is an interpreted programming language that was originally created in the 1990s, with the latest major revision released in 2008. It is a dynamically typed, high-level abstraction language, making it easy to learn and program due to its readable syntax. Python's standard library encompasses a wide range of features, making it a structured, object-oriented, and functional programming language. Additionally, its support for modules and packages renders Python a versatile choice for various programming tasks [21]. Python programming language was chosen due to its popularity in general, and a lot of software already use Python in Teleste's systems [22]. It is also easy to find packages to Python for example to handle the MQTT and REST communication protocols.

2.5.2 Docker

Containerization involves packaging and isolating software code, libraries, dependencies, and the operating system into a single, lightweight, executable unit that can be executed on any infrastructure. These packaged systems are referred to as containers. Compared to virtual machines, containerization offers greater portability and resource efficiency. Docker, introduced in 2013, has become the preferred

option for deploying applications [23].

Docker comprises four main internal components: Docker Client and Server, Docker Images, Docker Registries, and Docker Containers [24]. The Docker Client and Server are separate parts, with the server/daemon responsible for building, running, and distributing containers, while the client serves as an interface to interact with Docker. Docker images can be read-only templates or created from scratch. Typically, containers contain some form of read-only template images, such as the operating system. Docker registry functions as a repository, housing a library of images that can be pulled or pushed. Docker containers are self-sufficient units that can be deployed and executed as applications. These Docker containers are built from images.

2.6 The need for a new system / Current state of the production

Teleste Andon light system is becoming outdated. While efforts have been made to keep the system current, the hardware and software do not meet the current standards required for the direction in which the manufacturing process is heading.

The old andon light system utilizes in-house-made multitools, which consist of simple circuit boards and feature five different lights of varying colors in a light post:

- green - the current work batch has been completed and is waiting for to be dispatched
- blue - there is issues or need to address potential issues with the material, for example if a worker notices faults in a component
- white - there is need for a new batch, meaning that material is needed

- red - supervisor attention is required. Can mean that the worker is asking for holidays or there is serious problems with the production
- yellow - insufficient materials for the current batch

Each color have two states: blinking if the request is sent and no response has been received, or constantly on if the request is received and there are actions for the request. A light post can be seen in Fig. 2.3.

Figure 2.3: An existing light post of a cell in the production

As can be seen, the colors vary from the original three andon light colors, that were introduced by Toyota, and has been modified to suit better the flow of Teleste's production. One challenge posed by this irregularity is the learning time for new operators. A key point in the Andon system is to seem similar to a traffic light, with the colors trancending language and culture to be understandable immediately, which this system bypasses.

These lights are placed in the manufacturing production cells, where most often a single operator is placed. The operator can themselves push a button connected to an Arduino Nano to send an andon request and put a light on, and the light

multitool listens to IP-traffic to switch states if an outside source deems so. This multitool is programmed with the C++ programming language to control the light bulbs. Additionally the multitools are no longer manufactured, which is one reason the system needs replacing.

The light posts themselves are 12 volt bulbs that can be found in road cars, cased in different colored plastic housings.

2.6.1 LabView and Andon software

The Andon software that is in use in Teleste's factory was developed in 2013 as a part of Harri Laaksonen's thesis. The software was created with LabView [25], which is a graphical programming software developed to automate test systems, and in a very broad use in Teleste [22][26]. There is a trend where the production is starting to replace outdated technologies like LabView and change the programming frameworks to something more popular.

The Andon software consists of three parts:

- A server that is responsible for managing and maintaining the status of each cell. Also offers an external interface for other software to access and make updates to the cell statuses.

- A cell (client) interface to serve the worker in the cell. Polls their own status every 3-5 seconds from the server, and is also responsible for the Andon light itself, hence the polling.

- A manager view for supervisors and other staff to respond to the Andon request. In a case where the client has written down something specific into the request, the other parties can prepare themselves before going to the particular cell to solve the issue.

2.6.2 Current software that can use new system

There are multiple software applications in use in Teleste production that could benefit from the new Andon light system, besides the Andon software itself.

Teleste Warehouse Management System (TWMS) is a software made to simplify everyday deeds of employees and to gather data from different parts of the production. TWMS contains every item, location, transfer of items and much more of the in-house logistics of the production, and has been in development for nearly three years at the date of writing this thesis. Every set of items is stored in a handling unit uniquely divided with Universally Unique Identifiers (UUIDs). TWMS can benefit from the Andon software by for example lighting up a color at a specific location for displaying the capacity of the location.

Teleste Smart Collector (Collector) is a software made specifically for delivering hardware for the assembly cells. The assembly of a single product is made up of multiple items in the warehouse, and this software's purpose is to make the gathering of these items as easy as possible. Collector already benefits from smart bulbs and Electronic Shelf Labels (ESLs) for the ease of finding each item, by blinking a particular colored light depending on the employee.

There is also a constant flow of new software and innovation for the in-house logistics and state maintenance of the production where this new light system is able to help. Currently the trend of the production is to create digital twins, which should be taken into account in the design.

3 Design overview

The purpose of this chapter is to explain the design process of the expandable andon system. Each section of this chapter goes through each design step of the system describing and evaluating the choices. The overall design idea is to have different colored lights turn on whenever a command to do so is sent.

3.1 Choosing a communication protocol

HTTP (Hypertext transfer protocol) is a well-known communication protocol used commonly in IoT [27]. It uses a request-response model, where a client sends a request for another device, and the other device then sends a response. For this purpose when only the light bulbs must be commanded, polling is required. Polling means that the client must send a request in certain intervals to receive a response in order to update its state.

For HTTP in IoT there can be two types of polling: regular polling and long-polling. Regular polling involves sending requests constantly at a certain interval, with each request eliciting a response from the other device. Long-polling means that the end device sends a request once, and the other device would only respond when there would be a change of state for that requesting device. This type of polling would create a lot of traffic into the network if there would be multiple devices sending requests.

The WebSocket is a protocol created for web-applications to connect clients to the

server. WebSockets perform similarly to MQTT in terms of publish-subscribe model. In IoT WebSockets can be a very powerful option when the devices gather data and it needs only to be presented [28]. In this thesis it is important for messages to reach from one client to another in a mannerly time, so it would require unnecessary lot of configuring for the WebSocket server in comparison to MQTT.

The reason why MQTT was chosen over HTTP or any other protocol is that MQTT does not require requesting, and it is easy to implement. MQTT topics can be easily setup, and for this system it was decided that the topics should contain the location of the light and the color. For example, if a red light is wanted to blink in the manufacturing cell C5, the topic would be "c5/red" and the message would be in json {"state": "blink"}.

An MQTT broker was implemented using the Mosquitto software and deployed on a virtual server within the Teleste network. The goal of this implementation was to create a centralized messaging system that could be accessed by any device connected to the network.

The Mosquitto MQTT broker was set up using an existing Docker image, a containerization platform which allows for easy deployment of software applications. This was done in order to minimize the time and effort required for setup and configuration. The virtual server on which the broker was launched was chosen for its ability to provide the necessary resources and connectivity to support the broker's operation.

Throughout the entire design process, an application called MQTT Explorer was used to assist in the utilization of MQTT. It provides a user-friendly interface, allowing users to access each individual topic and message on the broker, which provides simple and easy-to-read insights about the communication. The application also offers information about the bridge itself, such as the number of retained messages and the bridge's uptime.

3.2 Choosing a platform for Andon lights

When the design needed a platform for the actual lights, two different approaches were initially studied: a similar approach as the production has currently with each location having a separate PCB (Printed Circuit Board), or switching to smart bulbs which are controlled through a software that is run on a server.

Because of the specified design done with MQTT, both options exists since there are no restrictions from the communication side of the system.

3.2.1 Microcontroller approach

The option of going with a microcontroller design approach requires firstly a development board, and secondly the lights themselves. The purpose is to create an embedded system that is connected to the local area network to receive MQTT messages, and illuminate the colors with lights connected to the IO-pins of the board. This section explains evaluations of possible choices for the embedded system.

Development board

There are different options to consider when choosing a development board, such as Arduino or Raspberry Pi, to name a couple of examples. The board should have I/O pins and an Ethernet connection if the old lamp posts are to be used with this system. Additionally, the Ethernet connection provides an alternative to a wireless system, which reduces the burden on the wireless channels.

The ESP32 board was selected for this evaluation because it is an affordable microcontroller board that offers sufficient performance. The ESP32 also has extensive documentation, making it easy to implement. There are numerous existing examples of MQTT implementation [29][30]. Furthermore, it meets the requirements of having I/O pins and an Ethernet connection.

One of the negatives of using PCBs is the difficulty of replication, and if there

is no automated system, it requires time and effort to duplicate them. One option can also be to create an interface to control the embedded system with, for example a display with buttons.

Three WT32-ETH01 development boards were ordered for test development. This type of board supports Ethernet connection without customization, features sufficient I/O pins, and is small in size suitable in challenging environments. The board can be seen in Fig. 3.1. It was studied on how difficult it is to program a light controlling application on the board, including creating a connection to the MQTT broker. The study also included insight on how the board has been used with different LED lights.

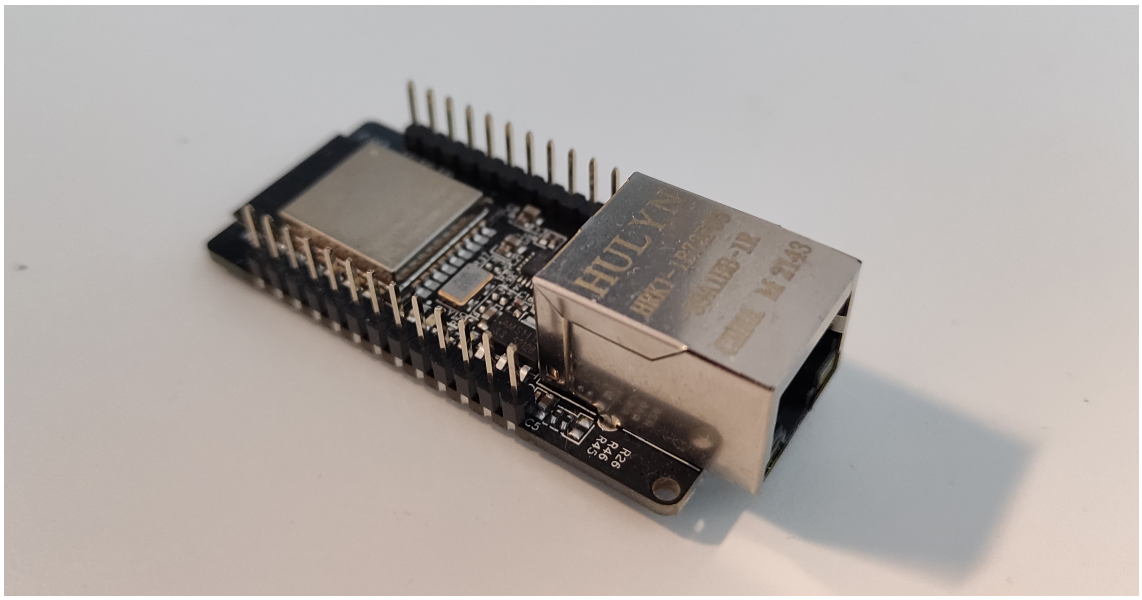

Figure 3.1: The WT32-ETH01 board

Physical lights

Two options were thought of when choosing lights for the circuit board: Utilization of the existing light posts, or creating new light posts with WS2812 LED-strips.

The existing light posts are equipped with lights of five different colors and

perform effectively in the current environment. These posts utilize 12 Volt bulbs commonly found in passenger vehicles. Consequently, the selected ESP32 alone cannot handle the bulbs due to its limited voltage compatibility, which ranges from 3.3V to 5.0V. Additional hardware is therefore required to address this limitation.

Using the existing posts still presents a restriction: they do not support any colors other than those already available. If the system is ever extended to include a sixth color, the old posts will not be usable anymore. The positive aspect of the existing posts is that they have painted plastic housings, clearly indicating the available colors in the current location. This helps users observe all the colors even when they are not illuminated. Observing lights that are not on can also provide valuable information to users, indicating potential scenarios and confirming that there is no need for attention.

The other option of using LED-strips is not without negatives either. Although the board is able to handle the lights, the looks of an LED-strip would not seem professional. The LED-strips can be covered in plastic housing, which fixes the outlook, but this still does not terminate the problem where the users could not see colors which are not on. One option would be to paint the plastic housing. This option would then make the LED strips similar to the light posts and introducing new colors to the system would be difficult.

Programmable LED-lights are easy to use, and there are a lot of examples where WS2812 LEDs are handled using an ESP32 board. They use 5V power so there is no need for an external power source besides the one that powers also the board. The WS2812 LEDs are simple in design, and can be piped to a large scale depending on the power input. An image of a singe LED module can be seen in Fig 3.2. Each module takes in 5 volts power, a digital input, and a ground, and features outputs for 5 volts, ground, and digital output. Most commonly LED strips contain 60 modules per strip, but can be easily extended when necessary.

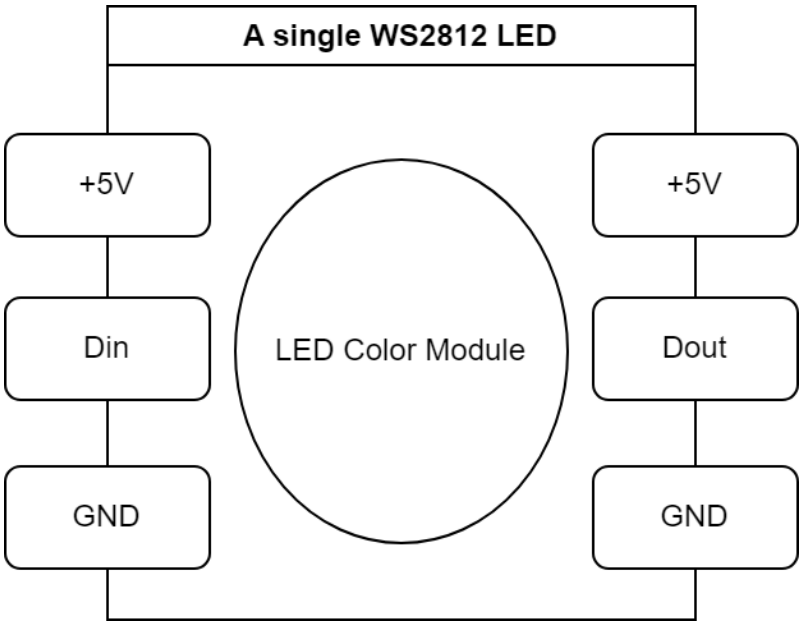

Figure 3.2: A simplified view of a WS2812 LED module

WS2812 LED strips of 60 modules were ordered to evaluate the difficulty of programming them, and determine if the looks of the strips are sufficient in the production environment. The testing and evaluation brought insight on how the LED strips would be contained in plastic housing and one strip would be split in sections of modules, which would be reserved for each color in the Andon system. Since a single LED strip contains 60 modules, and the Andon system in its current state features five colors, each section of the strip would have 12 modules reserved for each color of the system.

3.2.2 Philips hue

Philips hue (hue) is a light system produced by the Philips company, and as a lighting brand sells lighting products and services to consumers and professionals [31].

Hue uses technologies ZigBee, Matter, and Bluetooth, depending on the user and the application. ZigBee and Matter require a bridge to work, both being products of

the Connectivity Standards Alliance, and the bridge hue provides support for both technologies. Bluetooth does not require a bridge but it would require a separate connection to each of the bulbs creating too many connections, which makes this option not very user friendly and is not viable for this thesis.

Matter is a relatively new technology developed for the Internet of Things (IoT) with the help of multiple international companies [32]. It was released in October 2022 and, as a result, has not yet gained widespread adoption compared to more established technologies like ZigBee. Therefore, it was not used in this thesis, which began development prior to the release of Matter.

Hue provides an open HTTP REST Application Programming Interface (API) to the bridge documented on their website [33]. The API provides end-points to different sorts of home devices in the Hue product family, but this thesis focuses on the smart bulb aspect only.

The API provides interfaces for getting and setting lights states, searching for new lights, renaming lights, deleting lights, grouping lights, and renaming these light groups. To access these APIs, a username is required so anyone that can access the bridge cannot tinker with the system. Table 3.1 explains the most important APIs in this thesis.

Method	URL	Explanation
GET	/api/<username>/lights	Get all lights connected to the bridge
PUT	/api/<username>/lights/<id>	Set attributes of a single light
GET	/api/<username>/groups	Get all groups from the bridge
POST	/api/<username>/groups	Create a new group

Table 3.1: HUE API Examples

Here is an example payload in the PUT method to put a single light on with the Hue API:

```
{
    "hue": 50000,
    "on": true,
    "bri": 200,
    "alert": "lselect"
}
```

The hue color of the bulb is defined to be 50000 with the "hue" key in this payload, which would be slightly purplish blue due to the uint16 color spectrum being defined as ranging from 0 to 65535. The "on" key defines whether the light is wanted on or off. "bri" key defines the brightness from 1 to 254. The "alert" key is reserved for a blinking effect, "none"-value being not blinking at all, "select" blinking once, and "lselect" blinking for 15 seconds. The hue bulb itself cannot blink more than 15 seconds without sending another request restarting the time, creating one issue that needs to be dealt with later on.

The hue developers documentation also tells that the maximum amount of lights connected to one bridge is 63, and after the amount rises over 50, the system would become less responsive in general. This also creates one restriction which will be focused on later.

Another issue with using Hue is the price. At the time of writing this thesis the price of a single Hue smart bulb is around 50 euros.

Zigbee2MQTT - an alternative

Zigbee2MQTT is a platform for any zigbee device to connect to, and is a popular choice in home automation to eliminate the need of vendors' bridges such as the hue bridge [34]. It is developed by Koen Kanters, and is an open source project. As the name suggests, Zigbee2MQTT is used to convert MQTT messages into Zigbee, making it easy to control devices with the publish-subscribe method.

Zigbee2MQTT requires a Zigbee adapter to work, and provides a list of adapters to choose from. For this thesis a Texas Instruments LAUNCHXL-CC26X2R1 development kit was used to avoid uncertainties of the other choices. Most of the other adapters are based on the same hardware architechture as the Texas Instruments development kit.

The software architecture in Zigbee2MQTT consists of three parts:

- A zigbee-herdsman to handle the core communication from the zigbee adapter, and makes an API available for other parts of the software to use.
- Zigbee-herdsman-converters which convert and communicate from the herdsman to the device level zigbee clusters.
- Zigbee clusters define how devices on the zigbee network communicate each other.

Finally the software itself converts the zibee messages into coherent MQTT messages [35]. Zigbee2MQTT saves the states and capabilities of each connected device, and publishes them into corresponding mqtt topics [34]. Once a device is connected to the zigbee network, it is given a friendly name that refers to a human-readable name that users can modify by their needs and defaults to a general id.

The Zigbee adapter will not work by itself and requires a computer to plug into. A Raspberry PI 3 was chosen for this thesis, but any other device that can run Docker will work. Additionally the device must be accessible to a local area network (LAN) in order to communicate with MQTT. Additionally the Zigbee adapter was required to flash with the Zigbee coordinator and herdsman firmware according to Zigbee2MQTT instructions. After this the Zigbee2MQTT bridge could be started with a docker-compose file according to the guidelines provided on the Zigbee2MQTT website.

Zigbee2MQTT has a built-in user interface for controlling devices on the bridge.

It can be accessed through any web browser, and provides tools for example to add, remove, update, and control devices to the network, create and modify groups, and show logs of the Zigbee2MQTT system. This makes it very easy for maintainers to create new location instances where Andon lights are needed. The UI can be seen in fig. 3.3. As devices are connected to the network, Zigbee2MQTT subscribes to topics provided by the devices. In the case of a color smart bulb, the topics Zigbee2MQTT subscribes to are in the form "bridge name/friendly name/set", for example "zigbeeBridge/hue-bulb-1/set". The messages it is designed to understand are in JSON format, from which key-value pairs are parsed. For example, publishing the message

```
{"state": "ON"}
```

will turn on the light bulb. Table 3.2 Shows the most important topics needed in this thesis.

Pub/Sub	Topic	Explanation
Subscribe	zigbeeBrigde/bridge/devices	Get all devices connected to the adapter
Subscribe	zigbeeBridge/bridge/groups	Get all groups created on the network
Publish	zigbeeBridge/<device name>/set	Change the properties and status of a device

Table 3.2: Zigbee2MQTT message examples

Zigbee2MQTT supports different Zigbee devices from many different manufacturers, creating choices other than just Hue. Many of the light bulbs Zigbee2MQTT support, work generally the same. The features most bulbs have are state, brightness, color temperature, color, and effect.

When comparing the choices between the microcontroller and Philips Hue ap-

proach, both have their advantages and disadvantages. In summary they are compared in Table 3.3 with the existing system inlcuded.

	Development	Duplicability	Estimated costs	Communication
Philips Hue	Programmable in any language	Configurable and extendable through development	Approximately 100 euros per cell - bulb lifespan up to 25 years	Zigbee via a bridge - controllable via MQTT
Microcontroller	Limited to specific language	Must be programmed and requires extra hardware	Approximately 25 euros per cell - lifespan unknown	MQTT
Existing Lights	End of life - programmed with LabView	Established system	Replaceable bulbs 3 euros per piece - lifespan 1 year	IP controlled

Table 3.3: Comparison table of different design options

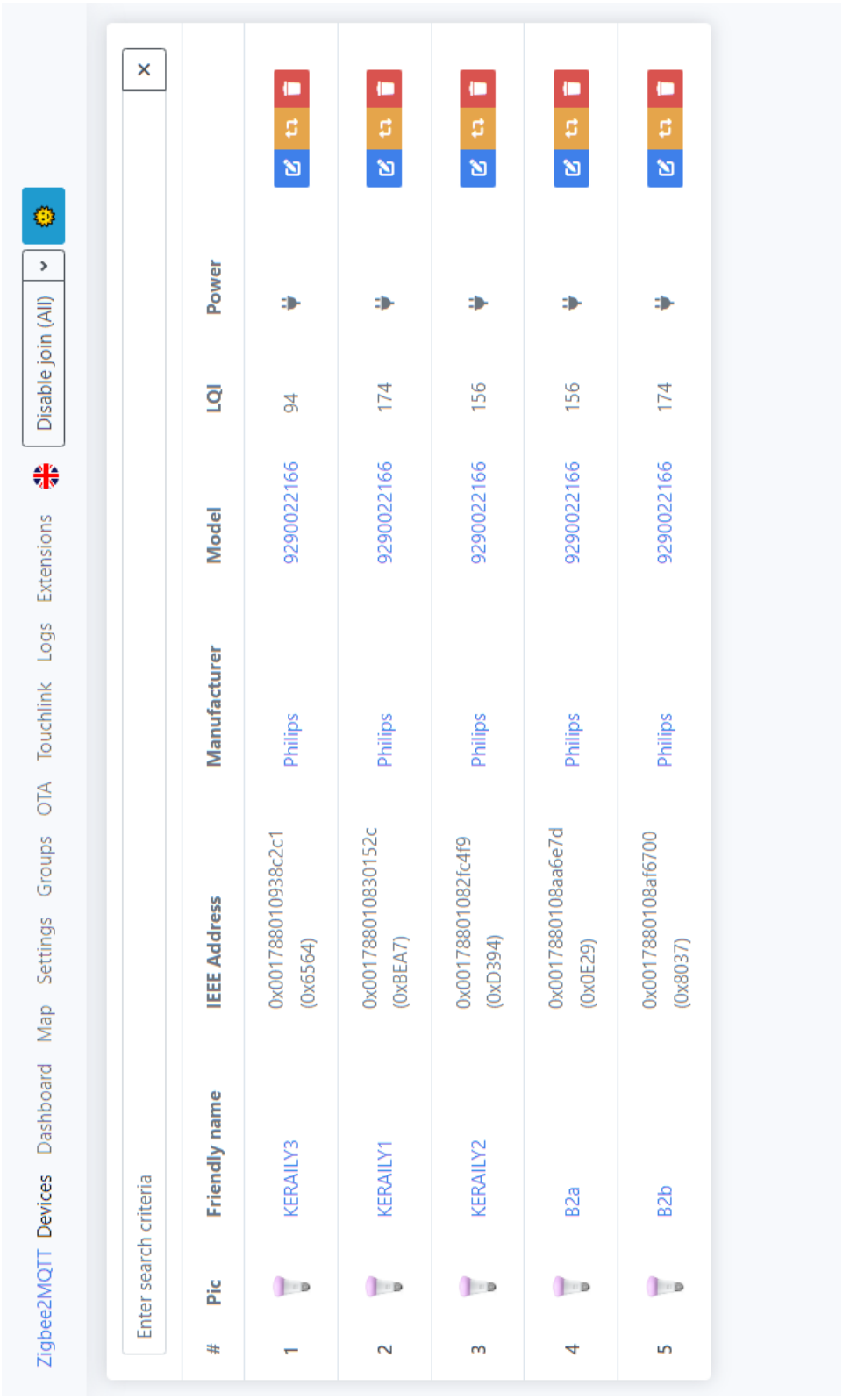

Figure 3.3: A view of the Zigbee2MQTT frontend UI

4 Development overview

In this chapter the software component is described in the development process. We describe the flow of the program which controls the lights, and touch on the testing of the system on how it performs. Lastly the whole structure and architecture of the system is reviewed.

In the second chapter it was explained how the current Andon software and its lights work. There is currently no need to rethink the Andon lights features, so the main purpose is to create lights that can replace the current light posts.

The hue approach was chosen due to the lack of research on the use of smart bulbs in manufacturing factories. It was deemed to be the most interesting approach since some smart bulbs were already in use in Teleste's manufacturing. Additionally using the PCBs would require a broader view on the issue before further actions might be made.

In order to keep the number of bulbs to a minimum, a proposal was made to have each cell contain only two smart bulbs: one for blinking (Andon request) and one for displaying constant colors (Andon response). Each of the two bulbs would cycle through the array of colors it is supposed to show in two second intervals. For example if there were two colors blinking, the bulb would blink in red for two seconds and transition to the next color after this. This allows users to easily understand the state of each Andon color, and it would be easy to add new features if necessary. These features could be new colors or different states for existing colors, to name a

few examples.

A software was developed to receive MQTT messages and send requests to the Hue bridge to control the constantly changing colors in the bulbs and manage the blinking.

4.1 Development of the software

It was decided to use the hue's groups as an ease-of-use to store which bulbs belong to which location or manufacturing cell. The groups are named correspondingly to the name of the cell.

For the software to function properly, it is required that all necessary bulbs are connected to the hue bridge, and belong to a group before the software is launched. Adding either manually or through the API of the bridge, new bulbs can be connected, and after placing the bulbs where they are wanted, create location based groups on the bridge.

The first step in the software is to connect to the Hue bridge to receive the current state of all connected bulbs and determine their location. This is achieved using the Hue group API, and a python package called phue, which contains all necessary functions to communicate with the bridge. The phue package is an open-source library and has an MIT licence which will work perfectly for this application. Since each cell contains two bulbs, a group is created to contain these two bulbs. The group is then named after the location of the two bulbs. For example, if two bulbs with IDs 4 and 12 are located in cell 'C8', the group would be named 'C8'. This allows the requesting client to receive the name of the group as well as the IDs of the light bulbs.

The software creates objects for each group, location, or cell and for each light. The group objects have attributes such as their name and the light objects they contain, as well as potentially their light states. The light objects have attributes

including their state, ID on the bridge, available colors, and whether or not they are supposed to blink.

The next step for the software is to create subscribable topics for the MQTT broker based on the groups. Communication with MQTT was done using the paho-mqtt package for python. The top-level topic for the lights is the group's name, and the subtopic is the color that the state is being changed to, for example, "C8/blue." Later on an upper level topic was added to separate the factories, since the location could be in a different municipality, and the same MQTT broker could be used in all of the factories. In this case the example topic would be "littoinen/C8/blue". The topics are automatically created from the group objects' names and the available colors of the light objects. The messages this software handles are simple JSON with a single key "state" whose value can be "on," "off," or "blink."

Once all of the topics have been subscribed to, the software can start to handle messages. Paho provides an "on message" function which will be run every time a message is received from the subscribed topics. The topic and the message itself are split to determine the correct location and the correct light, and the state it is supposed to be set to. First, the color and state are sent to the corresponding location, which then handles sending the state to the correct light, and if necessary, changes another light's state.

When a light object receives the message to set a color, it adds the color into an array which holds all the colors the light currently has on. The light object handles changing colors with a timed function that is launched with a predetermined interval. The timed function selects the first color in the color array and afterwards, it is put last in the array. If a color needs to be turned off, it is simply taken from the array. Every time the timed function is run, it uses a custom function to send a message to the hue bridge using the phue package. The function takes the light's ID, color, and state as arguments and sends a corresponding message to the bridge. When the

last light is turned off, a message is sent to turn it off.

The script was designed to allow a single light object instance to exist in multiple location instances. This means that if a command is issued to activate a color in two different locations that share a light, the light object itself recognizes that duplicate colors should not be added or removed from the color array.

Furthermore, it was determined that a location can have either one or two light bulbs. It was specifically implemented so that locations with only one bulb display blinking lights instead of constant colors. This feature serves to support warehouse logistics, including systems like the Collector system. By incorporating this feature, the light system can be applied in various scenarios, providing assistance in simpler ways. For example, it can help locate specific items or determine the occupancy status of a warehouse location.

This script is lastly dockerized using docker compose. Using docker makes this script easy to run from a server or a machine that is already used in running software. Dockerizing also makes it easy for a maintainer to restart the system if needed, and for DevOps to make quick changes in the system if required. A dockerfile, a docker-compose file, and a script file was created for this use. The script contains pulling new software from the source control service, building and restarting the container.

Testing the software

To test the application, a test environment was created. A Philips Hue bridge was setup to a corner in an office, and 12 Philips Hue E27 bulbs connected to it. To the bridge, seven separate locations were created, five of which had two bulbs, and two with one bulb. The light controlling script was run in a Docker container on a development computer, a Thinkpad P14s. Additionally a test script was created, which connects to the MQTT broker, and publishes commands to the seven locations with changing intervals from one to five seconds. The test script was run from the

same machine as the light controlling application. Finally the test script commands all of the bulbs to turn off.

At this point it was evident that even a low amount of bulbs connected to the Hue bridge will cause it to lag. Even with only a couple of lights could cause the blinking to look off, when the bridge is supposed to receive a new message every two seconds per light. It was chosen to take the Zigbee2MQTT approach, getting rid of the HTTP protocol at the same time.

To switch the python application to use Zigbee2MQTT instead of the phue package, it required just small modifications. The class that previously handled the phue package was replaced with a similar class alongside with identical methods. For example instead of requesting groups from the bridge, a subscription was made to gather groups from the Zigbee2MQTT system. Fig. 4.1 shows a UML diagram of the light controller script with the updated Zigbee connection and fig. 4.2 shows the flowchart of the software.

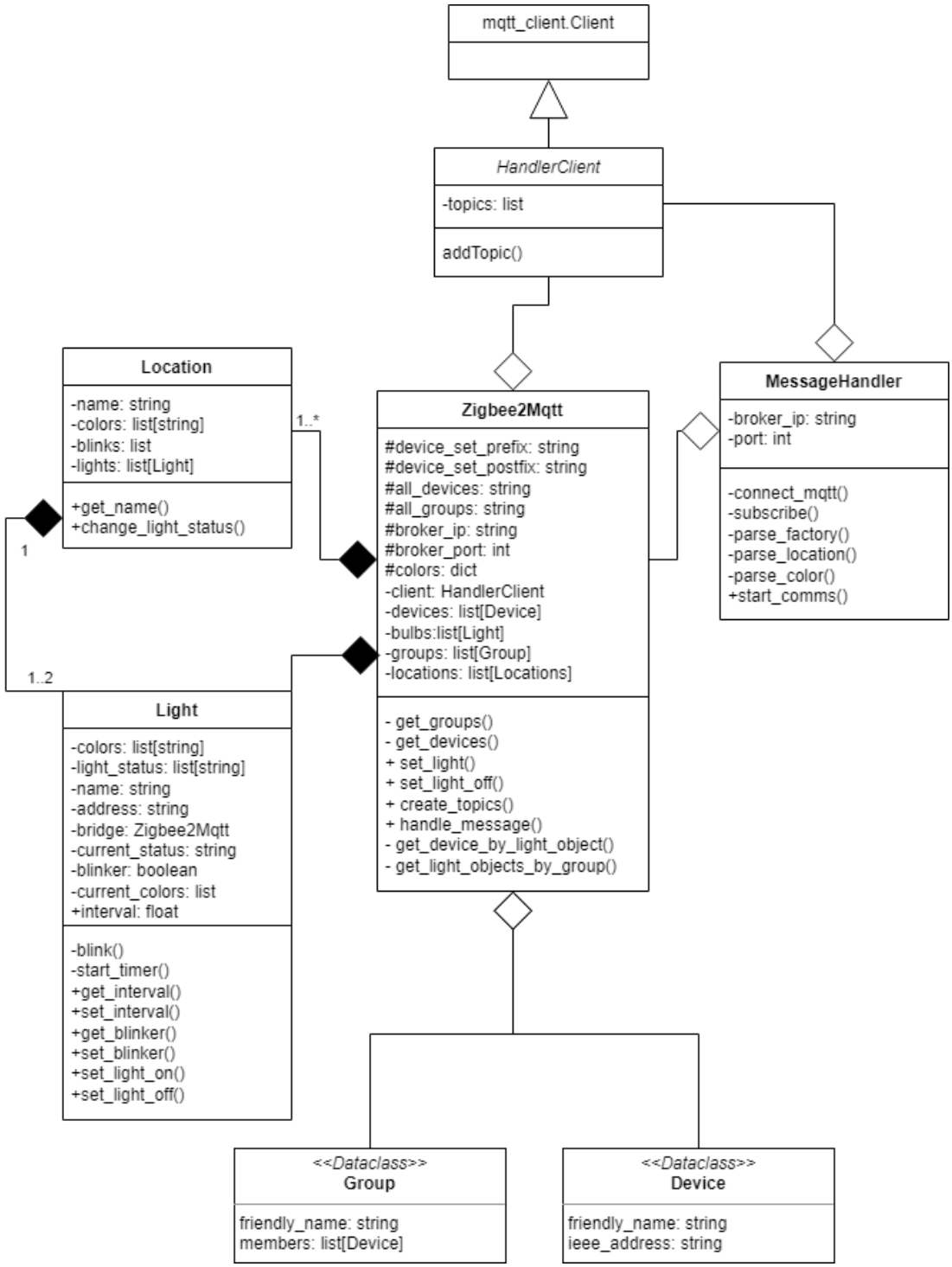

Figure 4.1: An UML class diagram showing the software's architecture

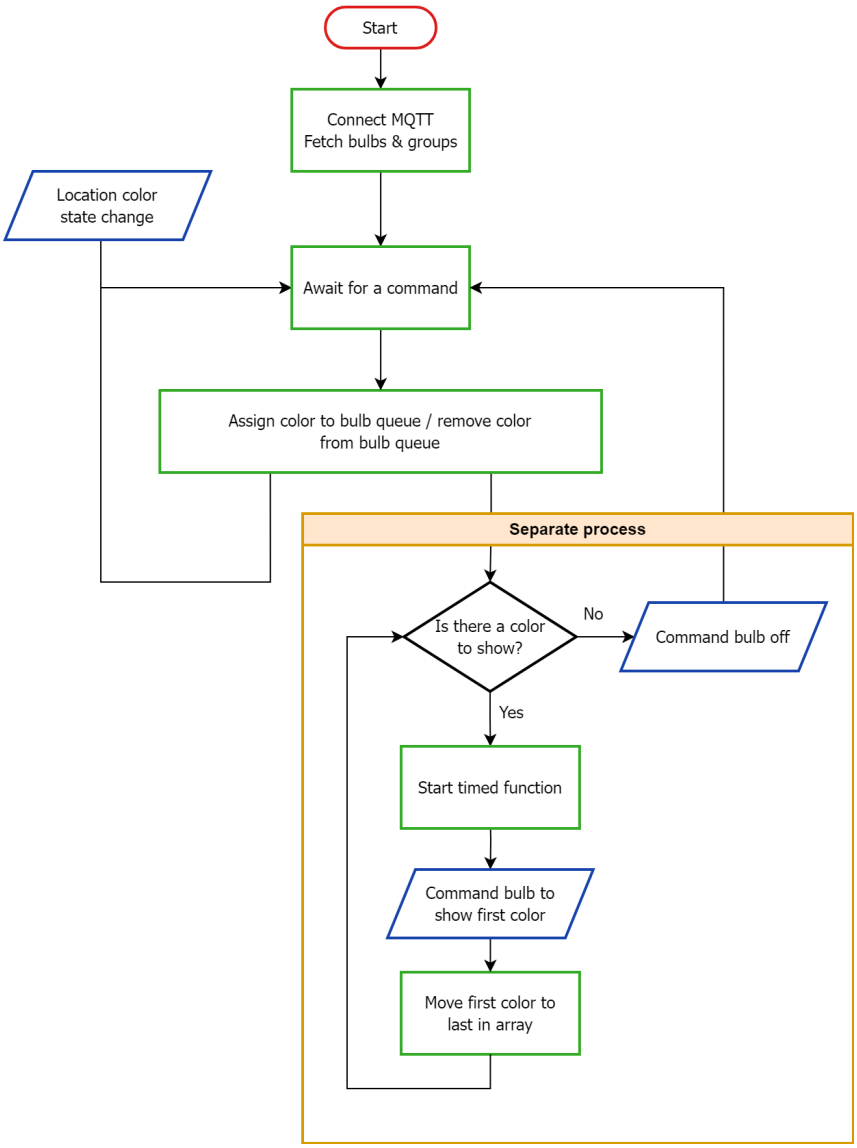

Figure 4.2: A flowchart view of the light controlling software

Other manufacturers' smart bulbs

After the swap from hue-bridge to Zigbee2MQTT was done, other manufacturers' Zigbee smart-bulbs were tested if there would be cheaper alternatives to hue-bulbs. Bulbs from Immax, ELGO, Ikea, and Osram were ordered for testing. All of the bulbs were similar to Philips Hue, with E27 sockets. One of the major differences in all smart-bulbs is the blinking functionality. Each bulbs' features were tested as

follows:

- Ikea Trådfri: Does not support a blinking feature and thus cannot be used in this system. Additionally the colors in the bulbs would have required fine adjusting. For example the color cyan appeared more white than preferred.
- OSRAM Ledvance: Supports blinking, but blinks in an atypical way. When there is a color and the blinking feature on, the bulb goes from the color to white and back to the color. If the bulb was to blink with the color white, it would appear not to blink at all. Therefore this option is not usable.
- Eglo Connect: Blinking looks desirable, but colors are off, and would require fine tuning. On the contrary, this bulb behaves irregularly with the blinking feature. When the bulb is supposed to go off, the bulb keeps blinking. Additionally the bulb might flash by itself occasionally.
- Immax NEO: With this bulb was achieved the best result of the other manufacturers' bulbs. The colors look desirable, and the blinking feature works as desired. This bulb blinks less often than the Philips Hue, but can still be used.

Overall, Philips Hue was found to be most technically advanced among all of the tested smart bulbs, which can explain the difference in price. Most of the alternative manufacturers' bulbs cost less than 20 euros, which would have significantly reduced the overall system cost. The Osram Ledvance, priced at around 40 euros, was comparatively more expensive. Presently, the market for advanced Zigbee bulbs remains limited, with only a few notable manufacturers offering products for sale.

4.2 General architecture

With the design of the light controlling script being completed, and the light communication system finalized, a coherent view of the complete architecture can be reviewed. Fig 4.3 shows an image of the system architecture.

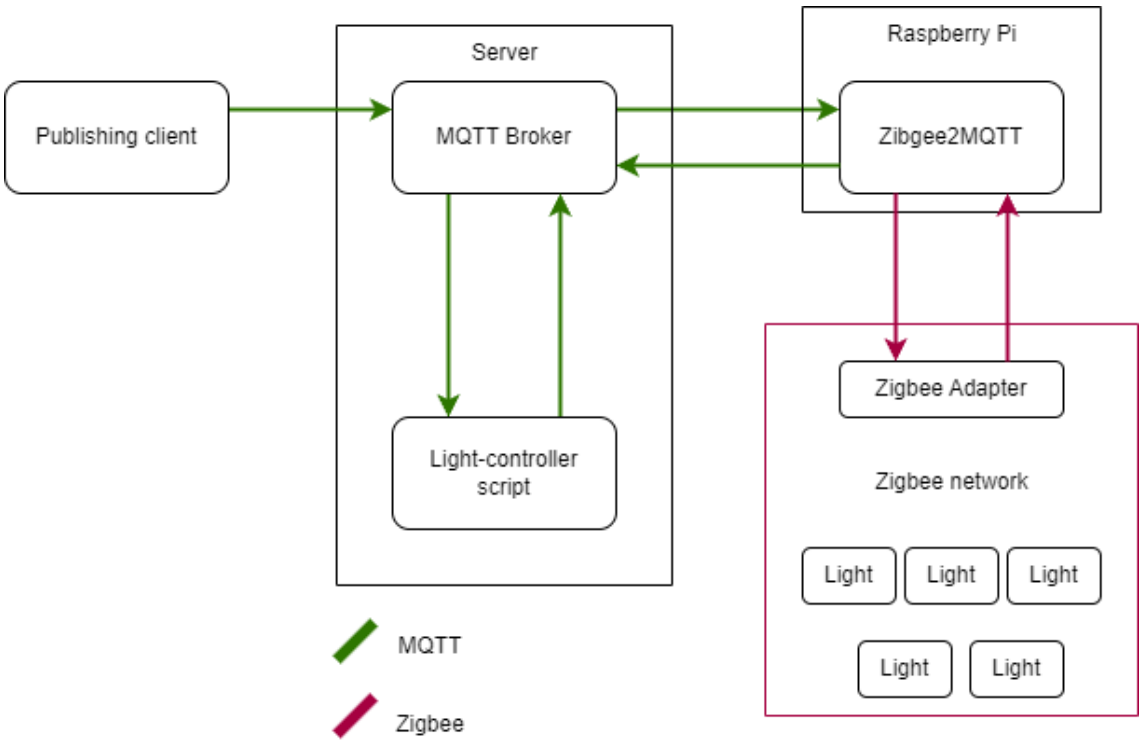

Figure 4.3: The complete architecture of the system

The MQTT broker mostly controls all of the communication, except for the communication in the Zigbee network, in which the light bulbs exist. The broker is responsible for transmitting messages from the publishing clients, ie. clients that need to control the lights, to the light controlling script, and from there to the Zigbee2MQTT service. The broker can reside in a server running in a docker container using an existing image.

The Zigbee2MQTT service is responsible for controlling all of the lights, and the Zigbee network itself. The Zigbee2MQTT itself runs in a docker container in

a device, in this case a Raspberry PI but any other will do, which can connect to the MQTT broker, and have the Zigbee adapter attached to. Zigbee2MQTT provides the necessary functions to transfer MQTT messages to Zigbee commands, and provides an interface for attaching new devices to the Zigbee network. The interface provided by Zigbee2MQTT can be accessed from any web browser and eases the use for maintainers and plausible debugging of the system.

The Zigbee adapter is the coordinator in the Zigbee network, meaning that the network would stop working without it. It is the necessary component on the Zigbee2MQTT service, which creates the bridge between the Zigbee communication and the service's interfaces.

The lights in the architecture are responsible for delivering the visual message to the user. They need to be Zigbee compatible to be able to connect to the network and the system. Most commonly used bulbs in this thesis were the Philips Hue bulbs, but some other manufacturers' bulbs would work as well.

The light controller script is the beating heart of the system. It receives simple MQTT messages and from that point on controls all of the bulbs on the system. The script's responsibility is to maintain the bulbs blinking and changing the colors within those bulbs in the given interval. The script is written in python programming language and is run in a docker container.

5 Implementation and Results

In this chapter, we delve into an in-depth analysis of the prototype that was designed in the previous chapter. Firstly, we scrutinize the design's characteristics and evaluate the ease of setting it up in a new environment. Additionally, we assess the overall platform's performance in a factory setting and identify any potential limitations that may arise during implementation.

Furthermore, this chapter includes a comprehensive analysis of the system's ability to meet the requirements of a smart factory, exploring the possibilities it can create.

5.1 Platform Characteristics

To setup the system, there are three parts that need to be configured for the system to start:

- An MQTT broker to handle the communication.
- A server or a machine where the docker container that handles the lights can be run in.
- The Zigbee2MQTT machine, which works as the coordinator in the Zigbee network.

The MQTT broker is easy to setup, as there are multiple different options available in docker images, for example Mosquitto. It is possible to setup the broker to

require username and password or even TLS certificate measures to avoid security breaches in the system. In this thesis' case the broker was chosen to be Mosquitto and it was setup in a server in the Teleste's service centre, which uses Ubuntu as the operating system. After the broker has been set up, the two clients can be run. The Zigbee2MQTT machine is started and connected to the web. The machine requires the Zigbee adapter, a configuration file, and a docker-compose file to run the Zigbee2MQTT service. In the configuration file there needs to be details about the MQTT broker; which IP address is it in, username and password for connection, or the TLS certificate details. In this thesis the Zigbee2MQTT is run in a Raspberry PI 3, but almost any other computer with similar computational power will work. The light-controller script docker can be run from the same system where the broker is. It requires all the same details as the Zigbee2MQTT about the broker, and can be started with the script.

Next step for the system is to connect lights to it. The most easy way to connect the bulbs is to access the frontend of Zigbee2MQTT, and clicking the "allow joining" button. If there are any reachable lights, they will connect to the network and all information can be seen from the frontend. The user can change the friendly name of the device if they wish so, as it can ease the process and maintenance of the system. When the user is satisfied with the amount of connected lights, the groups need to be created. The user can create as many groups as they want, and add one or two lights to a group. The group should be named by the location or the cell it is located in. One bulb can be used by multiple groups if wanted so. After any changes on the Zigbee network and groups is done, the light-controller script will need a restart, as the information about the lights and the groups is gathered only in startup.

The last step to use the system is a publishing client. Generally the system requires an MQTT message to a topic that is <factory>/<location or cell

name>/<color> with a state "on", "off", or "blink" in the message. This can be done on any other software, for example an Andon queue system.

5.1.1 Publishing client

To test the setup designed in the previous chapter, a publishing client was needed to determine the performance of the design. The Andon software used in Teleste was modified to send MQTT messages as it logs events into a database. The server is run on a windows platform which allows to install a Mosquitto package to publish messages through command line commands.

As the LabView software logs each event, it forms the topic from the name of the location and the color that corresponds the Andon task. The state the software interprets from the state of the task, eg. start, stop, hold. The server then publishes the message to the topic with the command *cmd mosquitto_pub -h *broker address* -p *broker port* -t *topic* -m {"state":*state*} -r*. The command first defines the broker's address and port, and with the -t and -m options determine the topic and the message respectively. -r option retains the message.

Additionally, since the Andon server is used in two different factories, an upper topic was added to determine the factory. For example, a topic would look like 'littoinen/A1/red'. The upper topic can be easily added to the software configuring it to the environmental variables.

The server starts to publish the state of each location it handles, which allows to follow all of the light traffic. For test purposes only one high traffic location was set up with the Andon light design in question.

Using MQTT as the communication protocol quickly proved to be the correct choice as there was almost immediately a need to get the ready status of each manufacturing cell, ie. the green status. Through this system it is also possible to create a digital twin of the state in the manufacturing floor, as the data is easily

accessible from the detained messages in the MQTT broker.

5.1.2 Single bulb locations

Single bulb configurations were tested with the Collector system. Three bulbs were set at the end of aisles in the picking warehouse of Teleste factory in Littoinen. These three bulbs were responsible to show the current picker's color whenever they needed items from the bulbs' locations. Two of the bulbs were setup so that each of them were in two separate locations, two aisles facing each other. The Collector software was modified to publish MQTT messages for blinking and shutting off the bulbs. Here is an example with python on how this is can be done:

```
import paho.mqtt.publish as publish


def publish_andon_messge(location, color, state):
    topic = f"littoinen/{location}/{color}"
    payload = {"state": state}
    auth = {"username": "user", "password": "pass"}
    publish.single(
        topic=topic,
        payload=payload,
        hostname="127.0.0.1",
        retain=True,
        auth=auth,
    )
```

In this example the publish.simple method is wrapped in a function, which can be used anywhere in the code. The settings like hostname, username, or password need to be configured correctly. Also the factory topic is hard coded and could be transferred into an environment variable.

In this setup the system functioned as expected without any major reported issues. The publishing client in this case was the Collector system, which published a message each time the user entered a new phase in the collection process. The Collector published a message which the light-controller read. The bulbs blinked for the duration of the picking until the user entered the next phase in the collection. If there were two users in the same location picking, the software handled the case by switching the colors in two second intervals, similarly to the cell locations. In this use case the two second delay is not that important, since the users are able to continue the collection even without the lights.

5.2 Experimental Results

The entire system was designed with the capability to handle more colors than originally intended. However, adding colors to the system in its current state requires modifying the code. The code includes a dictionary that defines all the colors in the system, and by modifying this dictionary, additional colors can be introduced. After making the change, the light controller subscribes to all the colors listed in the dictionary.

During testing, the system successfully incorporated two additional colors, magenta and cyan, alongside the existing five colors. It performed flawlessly without encountering any difficulties, just as expected. To enhance flexibility, the dictionary could be replaced with an external database, a file, or even environment variables. By adopting this approach, changing the colors would no longer necessitate rebuilding the Docker container.

The test setup for the lights in the production environment were made with two regular E27 light bulb sockets, as can be seen from Fig. 5.1. The sockets were attached to a metal bracket with clips to attach it to any part of the cell's structures. This is just an example of how to make easily duplicatable structures for the lights.

Figure 5.1: The bracket for the light bulbs

The lights and brackets require little space, making them easily adaptable in various locations. The primary constraint, however, lies in the cords that require a 230V power source. In terms of appearance, the setup is successful in the sense that it does not look out of place and is easily noticeable.

5.2.1 Visual representation

The visual representation of the setup is not effective. The lights cannot be seen from a 360-degree angle due to one bulb obstructing the view of the other. This becomes evident when a user walks down the corridor and can detect that a color is illuminated on the hindmost bulb but cannot determine which one until the cell is directly in front of them. This unintended behavior hampers quick and intuitive comprehension, as the user should be able to perceive and understand the state of a cell with a single glance.

A discussion was held with a long-term expert of logistics and systems of the company. The main focus of the discussion was the visual representation aspect. The logistics expert emphasized that anyone visiting the factory should be able to quickly grasp the meaning of the colors and understand the status of the cell at a glance. They compared the visual representation of the Andon lights to traffic lights, which are universally comprehensible. Two smart bulbs placed next to each other do not immediately convey a clear message. Although users can eventually understand the system's behavior after learning it, this should not be the desired premise.

One solution to tackle the visual representation problems could be to stack the light bulbs on top of each other so they would not block the visibility of each other, and might remind more of a traffic light setup. Designing the bracket for this kind of a setup then creates a challenge of its own, since the bulb sockets require wiring and solid mounting points which should not obstruct the visibility any further.

Another solution for the visual representation would be to increase the amount of bulbs in a single location. For example with four bulb there could be three traditional colors of bulbs: red, yellow, and green. The fourth bulb could then handle the more abnormal colors. This way the major colors would be individually represented, and would create a shorter learning time for the users to assimilate the colors. Increasing the amount of bulbs from two to four would double the price of the setup.

5.2.2 Delay of the lights

Another significant drawback of the system is the delay in perceiving all the colors. The two-second wait until the next color becomes visible poses similar problems to the visibility issue mentioned earlier. The delay, which exist due to changing the color in a bulb, causes the system to be slow in a way that a single glance cannot produce enough information to a user.

During the discussion with the logistics expert, the importance of immediate comprehension of the production status became apparent, highlighting the significance of the delay issue. For instance, if three different colored lights were illuminated in a cell, it would take at least four seconds for all of them to become visible. This time period is excessively long for users to detect the colors effectively.

Potential solutions to address this issue include modifying the interval between color changes. However, this could adversely affect the blinking quality of the smart bulbs, causing them to behave in an unconventional manner. Another approach, combining the colors, would only further confuse users, making it difficult to discern which colors are active. Therefore, the simplest solution would be to increase the number of bulbs, resolving the problem more easily.

5.3 Design Possibilities

The previously highlighted issues are the reason this system cannot be used in its present state. Because MQTT was used as the communication protocol, the framework for improved light posts exists. MQTT also offers the advantage of merging data into a unified entity. For instance, a digital twin, which gathers all of the states of each manufacturing cell into a singe interface, can be easily created. The view could visualize the manufacturing plant's layout, which can be further improved by showing the operator in each manufacturing cell, or tracking the actions taken by operators in response to Andon calls.

With the system's inherent flexibility to accommodate either one or two bulbs in a single location, as well as the capacity for a single group of lights to be allocated across multiple locations, it presents opportunities for broader application within the manufacturing domain. Already, it is being harnessed to aid manufacturing operators in their item-picking processes. Another opportunity is to use lights to transmit information about a location's status. For instance, consider a scenario where a batch of assembled items is consolidated at a single location, and the light would relay if the batch is still being assembled, waiting to be shipped, or temporarily designated as unsuitable due to a maintenance requirement.

An additional potential application for this system involves visualizing production progress. Given that the Andon status cannot be conveniently conveyed by this system, the bulbs could serve to indicate whether the total estimated production target has been met, or if the current batch is maintaining its scheduled pace. It is not uncommon for the output of a production cell to be influenced by various factors unrelated to the Andon system. This proposition aims to demonstrate whether production is in sync with the estimated timeline. For instance, consider a situation where a cell is projected to produce 100 units in a shift, but halfway through, only 20 units are completed. In such cases, the bulb could emit a yellow or red

light. On the other hand, if production is on schedule or surpassing the estimate, the bulb could emit a green light. This way every person on the production floor is able to conceive the production pace, and take action or analyze the reasons behind potential irregular pace.

6 Conclusion

The purpose of this thesis was to research and develop an Andon light system to replace the existing implementation for the production floor in Teleste Littoinen. While there exists an Andon system with functioning lights, it was required to upgrade it since the hardware and software are outdated. The system consists an irregular amount of five different colors deviating from the original Andon system. This five color system was originally created to assist the production further by explaining more closely what kind of attention the production cell requires.

Multiple options and choices were covered for choosing a communication protocol, and the hardware for the light system. HTTP and Websockets were found to be too difficult to create and maintain, while MQTT was found to be easy to implement and set up. This way the communication protocol stays lightweight, and quick to transfer messages.

The choice of hardware was made between a microcontroller approach and smart lights. The microcontroller was found to be similar to the current state of the system, but requires configuring light posts with it. An ESP32 microcontroller chip was found to be an easy and fitting option with an ethernet connection, which is a viable alternative to wireless configuration. With a microcontroller, two choices were considered to handle the lighting: WS2812 LED strips or reuse the existing light post. The other approach of using smart bulbs, more closely Philips Hue, was found to be more fitting choice due to the lack of research on it, and additionally the

technology was already in small use in Teleste. Philips Hue uses Zigbee wireless technology to communicate with the bulbs, and they were paired with Zigbee2MQTT, a framework for more universal Zigbee connections not limited to the Philips brand hardware.

Next topic in this thesis was to develop a way to use smart bulbs as Andon lights while keeping the cost as low as possible. The high cost of a singe smart bulb encouraged the research to try and make a cell Andon lights with only two bulbs. The idea was to have one bulb to do the blinking and the other one to show solid colors, while looping them if there would be more than one color present. Additionally one bulb groups were also taken into account for different usages. The developed script first subscribes to topics determined for the group, awaits for commands to show different colors, and publishes messages to the Zigbee2MQTT topics which then control the lights.

The developed system was found to be easy to setup and maintain with the user interface which the Zigbee2MQTT provides. The MQTT broker and the light controlling script were the other components which required to be setup. The system did not perform desirably, as looping the colors would require the operators to wait for each color to be seen when the desired outcome would have been to understand the state with a single glance. The system can be used in other scenarios, for example showing the state of a warehouse inventory location, whether it is full, half-full, or empty, or assisting operators to find items they are seeking in the warehouse. Additionally, because of the MQTT, a unified view can be created to see the state of the whole production floor.

References

[1] M. Füchtenhans, E. H. Grosse, and C. H. Glock, "Smart lighting systems: State-of-the-art and potential applications in warehouse order picking", *International Journal of Production Research*, vol. 59, no. 12, pp. 3817–3839, 2021.

[2] G. Lei, G. Lu, and Y. Sang, "Design of wireless andon system based on zigbee", in *2015 8th International Conference on Biomedical Engineering and Informatics (BMEI)*, IEEE, 2015, pp. 821–825.

[3] C.-S. Yeh, S.-L. Chen, and I.-C. Li, "Implementation of mqtt protocol based network architecture for smart factory", *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 235, no. 13, pp. 2132–2142, 2021.

[4] J. R. Bradley, *Improving business performance with Lean*. Business Expert Press, 2015.

[5] L. Wilson, *How to implement lean manufacturing*. McGraw-Hill Education, 2010.

[6] C. C. Pegels, "The toyota production system—lessons for american management", *International Journal of Operations & Production Management*, vol. 4, no. 1, pp. 3–11, 1984.

[7] J. Hirvonen, "Design and implementation of andon system for lean manufacturing", 2018.

[8] Y. Eaidgah, A. A. Maki, K. Kurczewski, and A. Abdekhodaee, "Visual management, performance management and continuous improvement: A lean manufacturing approach", *International Journal of Lean Six Sigma*, 2016.

[9] L. Koskela, A. Tezel, and P. Tzortzopoulos, "Why visual management?", 2018.

[10] M. V. Masdani and D. Darlis, "A comprehensive study on MQTT as a low power protocol for internet of things application", *IOP Conference Series: Materials Science and Engineering*, vol. 434, p. 012 274, Dec. 2018. DOI: 10.1088/1757-899x/434/1/012274. [Online]. Available: https://doi.org/10.1088/1757-899x/434/1/012274.

[11] —-, *Introducing the mqtt protocol - mqtt essentials: Part 1*, 2015. [Online]. Available: https://web.archive.org/web/20221124162235/https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/ (visited on 11/04/2022).

[12] V. Lampkin, W. T. Leong, L. Olivera, *et al.*, *Building smarter planet solutions with mqtt and ibm websphere mq telemetry*. IBM Redbooks, 2012.

[13] G. C. Hillar, *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017.

[14] "Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1", International Organization for Standardization/International Electrotechnical Commission, Geneva, CH, Standard, Jun. 2016.

[15] X. Wu and N. Li, "Improvements of mqtt retain message storage mechanism", in *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, 2018, pp. 957–961. DOI: 10.1109/IMCEC.2018.8469192.

[16] R. A. Light, "Mosquitto: Server and client implementation of the mqtt protocol", *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017. DOI: 10.21105/joss.00265. [Online]. Available: https://doi.org/10.21105/joss.00265.

[17] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, "Study on zigbee technology", in *2011 3rd International Conference on Electronics Computer Technology*, vol. 6, 2011, pp. 297–301. DOI: 10.1109/ICECTECH.2011.5942102.

[18] Z. Alliance, *Zigbee specification faq*, 2013. [Online]. Available: https://web.archive.org/web/20130627172453/http://www.zigbee.org/Specifications/ZigBee/FAQ.aspx (visited on 11/24/2022).

[19] D. Gislason, *Zigbee wireless networking*. Newnes, 2008.

[20] S. C. Ergen, "Zigbee/ieee 802.15. 4 summary", *UC Berkeley, September*, vol. 10, no. 17, p. 11, 2004.

[21] ——. "General python frequently asked questions". (), [Online]. Available: https://docs.python.org/3/faq/general.html (visited on 01/08/2023).

[22] K. Väisänen, "Kestävään ohjelmistokehitykseen siirtyminen telesten testausohjelmistotuotannossa", M.S. thesis, University of Turku, 2020.

[23] ——. "What is containerization?" (), [Online]. Available: https://www.ibm.com/topics/containerization (visited on 01/08/2023).

[24] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance", *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 17, no. 3, p. 228, 2017.

[25] *What is labview?*, 2023. [Online]. Available: https://www.ni.com/fi-fi/shop/labview.html (visited on 01/14/2023).

[26] H. Laaksonen, "Andon-työnohjausjärjsetelmän suunnittelu ja toteutuus", Turku university of applied sciences, 2013.

[27] W. Bziuk, C. V. Phung, J. Dizdarević, and A. Jukan, "On http performance in iot applications: An analysis of latency and throughput", in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, 2018, pp. 0350–0355.

[28] N. Mitrović, M. Đorđević, S. Veljković, and D. Danković, "Implementation of websockets in esp32 based iot systems", in *2021 15th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*, IEEE, 2021, pp. 261–264.

[29] Randomnerdtutorials, *Esp32 mqtt – publish and subscribe with arduino ide*, 2018. [Online]. Available: `https://web.archive.org/web/20221230111137/https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/` (visited on 01/07/2023).

[30] D. Tao, *Esp32 connects to the free public mqtt broker*, 2021. [Online]. Available: `https://web.archive.org/web/20230104084020/https://www.emqx.com/en/blog/esp32-connects-to-the-free-public-mqtt-broker` (visited on 01/07/2023).

[31] *Philips hue website*, 2023. [Online]. Available: `https://www.philips-hue.com/en-us` (visited on 01/07/2023).

[32] A. Moscaritolo. "What is matter? the new smart home standard, explained". (2022), [Online]. Available: `https://web.archive.org/web/20230102132447/https://www.pcmag.com/how-to/matter-explained` (visited on 01/07/2023).

[33] *Philips hue api documentation*, 2023. [Online]. Available: `https://developers.meethue.com/develop/hue-api/` (visited on 01/18/2023).

[34] *Zigbee2mqtt website*, 2023. [Online]. Available: `https://www.zigbee2mqtt.io/` (visited on 02/05/2023).

[35] *Zigbee2mqtt github*, 2023. [Online]. Available: `https://github.com/Koenkk/zigbee2mqtt` (visited on 02/05/2023).