Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Circuit Theory

Design and Realization of a Control System for Controlling Independent Lighting Sources

Petr Douda

Supervisor: Ing. Jan Havlík, Ph.D.
Field of study: Medical Electronics and Bioinformatics - Medical Instrumentation
May 2024

Acknowledgements

Thanks.

Declaration

Abstract

This Master thesis deals with the design and implementation of a control system suitable for controlling independent light sources using the Digital Addressable Lighting Interface (DALI). The system employs gateways for connecting the DALI bus to Wi-Fi, which can accommodate most DALI devices. These devices are subsequently managed through the gateways from a central web application, utilizing a custom communication protocol based on the Transfer Control Protocol (TCP). This facilitates significant system flexibility and remote control from various devices, including smartphones. The thesis further outlines a potential application of this control system. In this particular application, the system was effectively utilized for managing individual light sources in architectural lighting.

Keywords: Control, Lighting, Python, DALI, Wi-Fi, Gateway, HTTP/REST, TCP

Supervisor: Ing. Jan Havlík, Ph.D.
Department of Circuit Theory
Faculty of Electrical Engineering
Czech Technical University in Prague

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací řídicího systému vhodného pro řízení nezávislých světelných zdrojů pomocí rozhraní DALI (Digital Addressable Lighting Interface). Systém využívá brány (Gateway) pro propojení sběrnice DALI s Wi-Fi, díky tomu umožňuje integraci většiny DALI zařízení. Tato zařízení jsou spravována prostřednictvím těchto bran z centrální webové aplikace s využitím nového komunikačního protokolu založeného na protokolu TCP (Transfer Control Protocol). To propůjčuje systému flexibilitu a umožňuje vzdálené ovládání osvětlení z různých zařízení, včetně chytrých telefonů. V práci je dále nastíněna potenciální aplikace tohoto řídicího systému. V této konkrétní aplikaci byl systém efektivně využit pro řízení nezávislých světelných zdrojů v architektonickém osvětlení.

Klíčová slova: Řízení, Osvětlení, Python, DALI, Wi-Fi, Gateway, HTTP/REST, TCP

Contents

Figures

Tables

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Douda Petr**	Personal ID number:	**483486**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Circuit Theory**		
Study program:	**Medical Electronics and Bioinformatics**		
Specialisation:	**Medical Instrumentation**		

II. Master’s thesis details

Master’s thesis title in English:

Design and Realization of a Control System for Controlling Independent Lighting Sources

Master’s thesis title in Czech:

Návrh a realizace řídicího systému pro ovládání nezávislých zdrojů osvětlení

Guidelines:

1. Study the perception of light by humans and living nature. See also the issues of controlling light sources via DALI bus and wireless communication technologies allowing remote control of independent light sources.
2. Design and realize an electronic system for controlling independent lighting sources controlled via DALI bus.
3. Perform experimental measurements to demonstrate the functionality of the realized system.

Bibliography / sources:

[1] Cajochen, C., Freyburger, M., Basishvili, T., Garbazza, C., Rudzik, F., Renz, C., Kobayashi, K., Shirakawa, Y., Stefani, O., & Weibel, J. (2019). Effect of daylight LED on visual comfort, melatonin, mood, waking performance and sleep. Lighting Research and Technology, 51(7), 1044–1062.
[2] BOYCE, Peter. Human factors in lighting, CRC Press, Taylor & Francis 2014.
[3] https://cs.wikipedia.org/wiki/DALI_(rozhraní)

Name and workplace of master’s thesis supervisor:

Ing. Jan Havlík, Ph.D. Department of Circuit Theory, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **02.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Jan Havlík, Ph.D.	doc. Ing. Radoslav Bortel, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt Student’s signature

Chapter 1

Introduction

Light is among the most important factors that influence almost all living organisms on our planet. In addition to facilitating sight, it also governs circadian rhythms. “Circadian systems exist in a wide range of life, from unicellular organisms to humans, in insects, plants, fish, birds, and mammals. Furthermore, there are cyclic patterns that occur over the seasons, called circannual rhythms, such as the seasonal breeding of mammals and seed germination in plants. These are believed to be controlled by the gradual change in the light / dark ratio that occurs over the seasons, as signaled by the circadian system.” [1].

Lighting influences a plethora of aspects of our daily lives. Lighting conditions, for example, influence our depth perception [16], and therefore our performance in tasks where judging distances is important. Our sleep cycles are governed by changes in lighting; therefore, inappropriate lighting can lead to reduced alertness and other issues [15]. Changes in the spectrum of ambient lighting also have an impact on human perception and well-being. As documented in [2]: “We have evidence that a daylight-LED solution has beneficial effects on visual comfort, daytime alertness, mood, and sleep intensity in healthy volunteers.” All these factors make lighting in the context of living organisms and especially humans a topic of great interest.

Given the importance of lighting, it is desirable to control it as best as possible. In both biomedical engineering and general engineering practices, there is frequently a requirement to independently manage multiple light sources or to individually regulate the spectral channels of multispectral light sources. Common uses include managing light sources for phototherapy and manipulating their spectral distribution or controlling lighting technology. Furthermore, remote control of light sources is beneficial in certain scenarios.

A common method to control light sources is through the use of the DALI bus. Digital Addressable Lighting Interface (DALI) is an open standard for communication with lighting technology. DALI is a two-wire multimaster bus. DALI bus is specified by a series of technical standards under IEC 62386. A DALI network consists of one or more controllers, input devices, control gear with DALI interface (e.g. LED drivers and dimmers) and a bus power supply with current limitation [9].

Although the DALI bus can extend up to 300 meters according to the standard, there are numerous benefits to transmitting DALI communications through a computer network, including the potential for wireless connectivity. Such wireless communication might be implemented using a WiFi network, a LoRa network [17], which is a radio network designed for long-distance data transmission, or a Zigbee network [14]. Controlling light sources wirelessly can be critical in situations where the position of the luminaires must be truly independent. One example such situation can be an application in exterior lighting where light sources must be controlled from a single point; however, they are separated by a road. Although it would be arguably possible to use wired communication even in this case, it would limit the traffic on the road, and thus a wireless approach is favorable.

There are some devices available on the market that allow remote control of devices on the DALI bus. These are, for example, the R091 - Modbus TCP/DALI converter device, which serves as a Modbus TCP server (accepts Modbus TCP commands) and web server, and controls a DALI bus [4]. However this has several drawbacks:

- The module does not support wireless communication and would require a second device connected via Ethernet to allow it.
- Due to the different nature of Modbus [18] to DALI it adds additional complexity to the communication protocol.
- It does not allow for the control of multiple DALI networks at the same time.

Probably the most advanced and versatile solution on the market is provided by Zencontrol [25]. Their solution meets all technical requirements, supports wireless communication across multiple DALI networks, and has a central web interface. However, it is aimed towards large-scale projects and may be too complicated/costly to set up and maintain for smaller applications. Such as research projects that do not require a solution with a full cloud support complete with analytics and power monitoring.

Another available solution comes from Foxtron in the form of DALIweb and DALInet devices [7]. These devices also allow wireless communication with DALI and have a central web interface. However, only the central web application is wirelessly available (DALIweb device), and individual DALI networks (DALInet devices) must be connected via wires. Although this could be overcome by using multiple DALIweb control units, it would be needlessly expensive.

The goal of this thesis is thus to design and implement a control system for multiple independent light sources. A system that would overcome the shortcomings of the solutions listed above and that would meet all technical requirements. The system must meet the following requirements:

- It has to allow for independent positioning and control of the light sources which utilize DALI drivers.
- It must facilitate positioning of the luminaires in the range of tens of meters from each other.
- The designed control system must be suitable for various applications where flexibility and expandability are key features allowing the system to be adapted for each individual project.
- It must also be lightweight, and thus cost effective for smaller projects.

1.1 DALI protocol

In this thesis a basic version of the DALI communication protocol is used. The protocol uses Manchester encoding with logic levels and timing of encoded bits that can be seen at the bottom of Figure 1.1. The protocol uses a master-slave architecture and can support multi-master configuration. Basic version of the DALI protocol uses two types of frames (see Figure 1.1):

- A 16-bit forward frame that is sent from master device (application controller) to a controlled device also called control gear (e.g. a light source). The first 8 bits of this frame contain an address which allows the frame to be designated as a broadcast for all devices, a broadcast for a specific group address, or a message for a single device. The other 8 bits store the data/command that are to be sent to the controlled devices.
- An 8-bit backward frame that is sent the other way as a response to query-type commands. This frame contains the queried data such as the current light level the light source is set to.

All frames use one start bit and two stop bits. Expanded versions of this protocol also use other longer forward frames, such as 24-bit or 32-bit. In addition, input devices such as switches or light sensors that use these longer frame types are supported by the protocol. However, these longer frames are not used in this thesis, and therefore neither are any input devices.

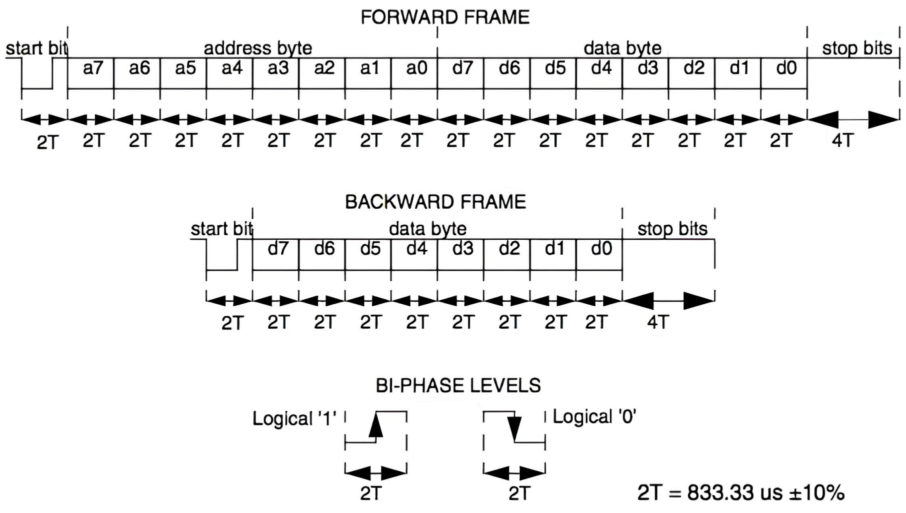


Figure 1.1: DALI protocol - Forward frame sent from controller to a controlled device (light source), Backward frame sent from controlled device (light source) to controller as a response to query type commands, Manchester encoded bits used in the protocol and timing can be seen in the bottom. Image sourced from [3].

Chapter 2

Design

In this chapter the theoretical design of the control system and it's overall structure are discussed. The control system is to be designed to function as described in Figure 2.1. It controls multiple DALI light sources over multiple DALI networks. It also supports inputs from multiple users at the same time. The light sources may be placed arbitrarily. The best suited communication protocols for this application and types of hardware are also discussed in this chapter.

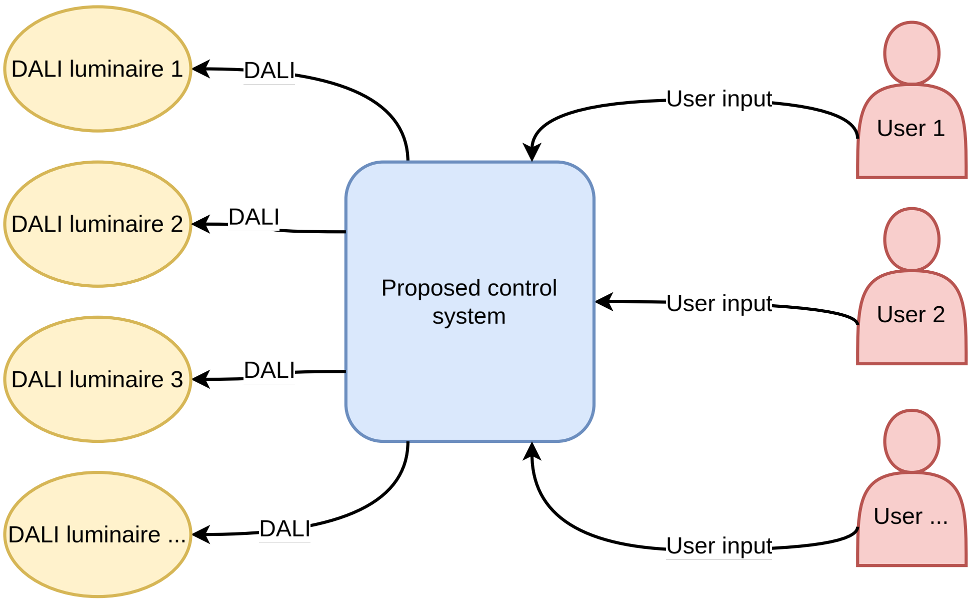


Figure 2.1: Control system interaction diagram

2.1 Interfaces

As specified in the assignment for this thesis, the light sources would be controlled via the DALI bus. The light sources can all be connected to the same DALI bus, or each light source can be connected to its own DALI bus or any combination of these. Given that the control system will need to be able to connect multiple DALI busses which do not have to be physically positioned next to each other.

To facilitate independent positioning of the light sources/DALI busses, it is necessary to use wireless communication. It would be impractical and sometimes even impossible to use wired communication with luminaires positioned further apart or, for example, over a road from each other. I have decided to use Wi-Fi [11] for all wireless communication throughout the control system. The main reasons for this decision are: Wi-Fi is readily available in many places where the system could be used or can be easily set up using a relatively cheap commercially available Wi-Fi Access Point (AP). It also allows for the creation of mesh networks or the use of commercially available repeaters for applications that require positioning of the lights over larger distances. Using Wi-Fi also allows the system to be connected to the Internet without any modifications, which makes it preferable to other technologies such as Bluetooth [13] or Z-wave [24]. One drawback of Wi-Fi is its higher energy consumption compared to technologies such as Bluetooth or Zigbee. However, each light source is expected to have a comparatively much larger power consumption than the control system itself. Thus, the power consumption of the wireless technology used will not have a significant impact on overall energy consumption. In addition, the new Wi-Fi HaLow technology described in the standard [12], which provides a superior range and lower power consumption, makes a compelling argument for the use of Wi-Fi. With the use of Wi-Fi from the start, the system can be easily upgraded to use Wi-Fi HaLow in the future.

To provide a way for the user to communicate with the system and control the light sources, I have decided to use a web application. Using a web interface provides the benefit of being able to control the luminaires from any device such as a phone or a computer available to the user of the control system. It also does not require any installation process on the user device as opposed to a conventional computer or phone application. This also makes it inherently platform-independent and usable across all of the different operating systems. For convenience of the user, this web application should also be central and aggregate all the light source controls in one page as opposed to having a web page/application for each luminaire individually.

2.2 System structure

As there can be multiple DALI busses controlled by the system, each one of the busses will have to be connected to a device that will facilitate connection to the Wi-Fi network. Also, the web user interface has to be hosted somewhere. Now, two different structures are possible:

- Distributed control - Each DALI bus will be connected to a small computer which will provide the Wi-Fi capabilities and one/multiple of them will host the web application.
- Centralized control - Each DALI bus will be connected to a small computer with Wi-Fi capabilities and the web application will be hosted on central computer separately.

The first approach has the advantage of redundancy in the sense that the control can be theoretically executed through any of the computers. However, distributed systems are generally harder to implement and debug. The centralized control system would be easier to implement. It could also use a more powerful central computer to run a complex web interface and use cheap microcontrollers to connect the individual DALI busses. However, in the distributed system all the computers must be able to run the web application, thus leading to an increase in the cost of the solution or simplifying the web application for cheaper microcontrollers. As being cost-effective is one of the requirements for the control system, I have decided on centralized control structure. Therefore, the resulting control system would be constructed as follows [5]:

- DALI controlled lights would be connected to microcontrollers with Wi-Fi capabilities to facilitate connection to a Wi-Fi network. This microcontroller should be simple and cheap as there could be a lot of individual lights thus needing a lot of microcontrollers.
- There would be one central computer. A personal computer or a single-board computer which would host a web application providing the control interface for all the individual luminaires connected to the network.
- A commercially available Wi-Fi AP would be used to create the network. To this network, the microcontrollers, the central computer, and any devices accessing the web application would be connected.

The complete design of the system can be seen in Figures 2.2 and 2.3.

I also needed to decide how to handle communication between the DALI busses and the control application through the microcontrollers. The first option was to implement some kind of custom protocol. I would define a set of higher-level messages (possibly HTTP) representing the different DALI commands along with their parameters. These commands would then be transmitted from the control application to the microcontrollers which would translate them to the corresponding low-level data bytes according to the DALI protocol and send them to the DALI bus. However, this would necessitate defining each DALI command individually and would mean a lot of work and code both on the side of the control application and on the microcontrollers. In light of that, I decided for a second option, which was implementing the microcontrollers as gateways. In this way, the control application would send messages already containing the exact DALI data bytes needed, and the microcontroller would only forward them to the DALI bus. In this way, the microcontrollers are agnostic to the data being sent and inherently support all the DALI commands available and do not need to be configured for each individual DALI bus. The commands have to be implemented only on the side of the control application, which is easier and also means that the gateways can be potentially used with a different control application in the future without the need for any modifications making the entire solution more flexible and customizable.

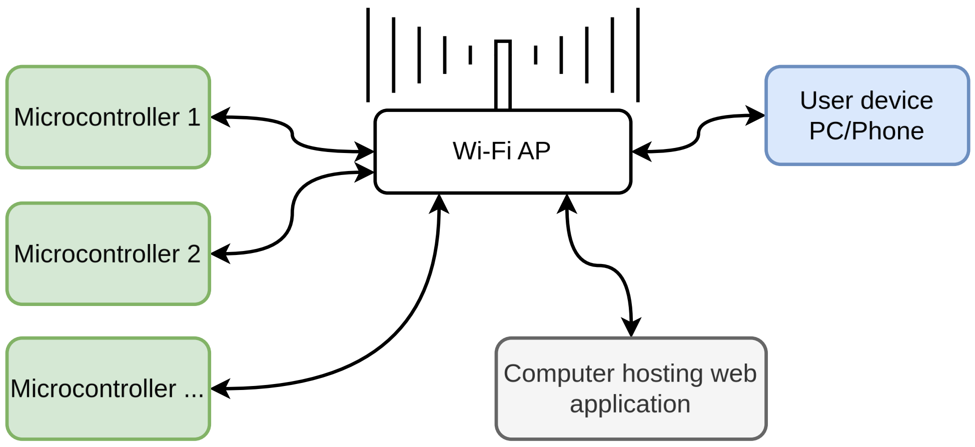


Figure 2.2: Proposed Wi-Fi network [5]

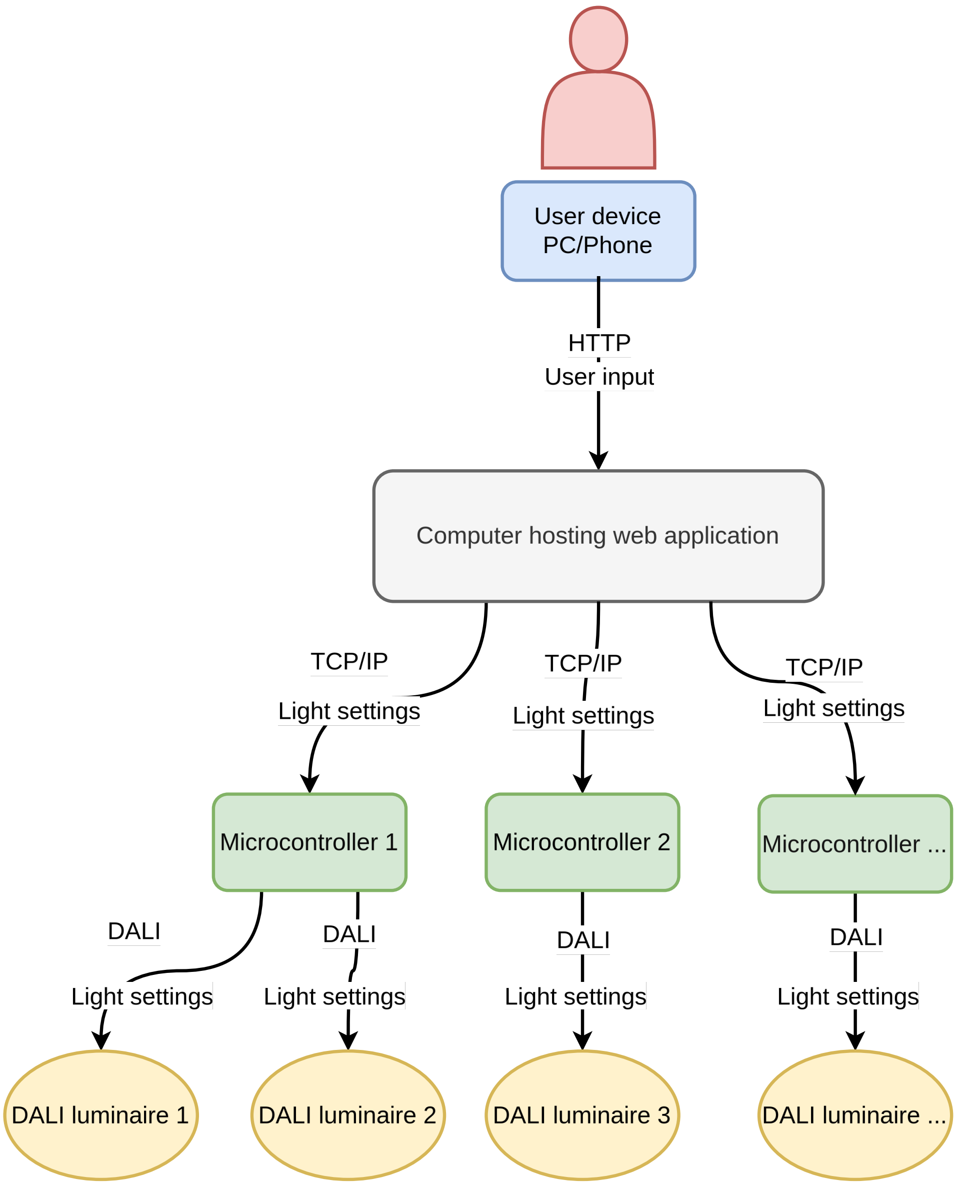


Figure 2.3: Control system architecture

Chapter 3

Implementation

In this chapter, the concrete hardware and software used to implement the control system design outlined in the previous chapter 2 are described. My own protocol used to transmit DALI data wirelessly is specified (see Figures 3.1 and 3.2). This chapter also contains a description of the DALI gateway hardware, firmware, and algorithm for receiving and forwarding messages. Lastly, the central control computer and application are discussed in detail. The user interface can be seen in Figures 3.7 and 3.8.

3.1 DALI gateway

3.1.1 Communication protocol

The communication protocol is the central piece of the implementation of the gateway between Wi-Fi and the DALI bus. As such, it was important to choose a suitable existing protocol on top of which the sending of DALI messages could be implemented so it would be simple and efficient. The first choice would be to follow the DALI technical standard [10] and use one of the approaches described there. Such an approach would be to use communication over the User Datagram Protocol (UDP). The UDP is connectionless and supports unicast, multicast, or broadcast addressing. As such it is similar to how the DALI protocol works, it is fast and low level, which are great advantages. In spite of this, UDP communication has some significant disadvantages. It has no mechanism to ensure that all data packets are delivered and does not guarantee the order of delivery of the individual packets sent. I would have to implement this functionality myself or account for these shortcomings in the control application. Also, if I wanted to send data over the internet its main advantage of similar addressing to DALI would be lost as broadcast and multicast cannot really be used over the Internet.

Another option was to use ModbusTCP; however, as discussed in the Introduction 1, I did not find this protocol advantageous for transmitting DALI data frames, as the resulting Modbus register structure would be very complicated.

When I started developing this control system and the DALI gateway, I have actually used the Hypertext Transfer Protocol (HTTP) [19] to transmit data to the gateway. This had the huge advantage, that I did not need the control application to be functional and I could easily send data from my browser. This was a great benefit for debugging the gateway implementation and DALI communication in general. However, in the end, I decided not to use HTTP in favor of switching to the Transaction Control Protocol (TCP) for the communication.

TCP is used to communicate over the computer/IP network in a reliable, ordered, and error-checked way. [21] This is done using a client-server connection over which a data stream is sent. These properties make it preferable for this application, as the order of frames is important for certain DALI commands. It is also lower level than HTTP and, therefore, has lower overhead. The only disadvantage of using TCP is that it does not support boadcasting to multiple devices/gateways at once. However, this can be easily resolved in the control application.

I had to implement some extensions on top of TCP to overcome some of the issues that come with wireless communication, such as higher latency. Simply transmitting DALI data frames one by one over TCP was not an option for several reasons. Firstly, in certain situations several DALI data frames need to be transmitted after each other within a specified time window. However, this could not be guaranteed given the nature of wireless communication over Wi-Fi. To solve this, I allowed for sending multiple DALI data frames in a batch that would be forwarded to the DALI bus all at once. Secondly, I wanted to prevent the gateway from misinterpreting other messages possibly received by chance for DALI commands sent from my control application. An example of this could be an HTTP request made to the gateway. I have solved this issue by using a unique identifier of the gateway in the messages, so only messages containing this identifier would be considered and other "random" messages would be dropped. I have also included the number of bytes that should be received at the beginning of the message to determine when the TCP connection/socket should be closed.

The resulting communication protocol then works as follows: When a socket connection is established, one forward message / command of a specified format can be received (see Figure 3.1). After that, a corresponding backward message/response (see Figure 3.2) is sent back, and the socket connection is closed. The messages consist of three segments [5]:

- 2 bytes - containing the number of bytes that constitute the next two parts (m+2n). This segment is mandatory.
- m bytes - containing a unique identifier of the gateway. In my implementation, it is 6 bytes of the gateways MAC address. If this identifier is unknown by the control application, it can be obtained from the gateway by sending an empty message consisting only of the first segment with a value of 0. This will trigger a response containing the identifier in the second segment. This segment is mandatory if any commands follow.
- 2n bytes - containing:
    - Either n two byte DALI frames in 8 bit address + 8 bit data format in case of the forward message. This segment is optional.
    - Or n two-byte response frames in 8 status + 8 bit DALI data format in case of the backward message (in order, one for each command received in the forward message). The status byte contains either 255 if a valid response was received from the DALI bus or 0 otherwise.

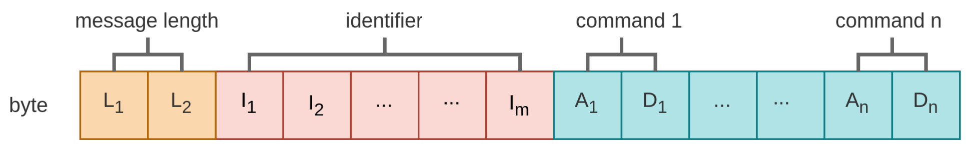


Figure 3.1: Forward message structure: Lx - bytes containing message length (m+2n bytes), Ix - identifier bytes, Ax - DALI address bytes, Dx - DALI data bytes [5]

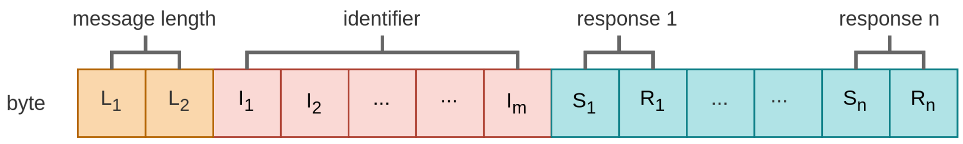


Figure 3.2: Backward message structure: Lx - bytes containing message length (m+2n bytes), Ix - identifier bytes, Sx - Status bytes, Rx - DALI response data bytes [5]

3.1.2 Gateway algorithm

To the communication protocol described above corresponds Algorithm 1 at the side of the gateway. First, the gateway microcontroller connects to a specified Wi-Fi AP and starts listening for TCP connections on a specified port. The DALI interface is also initialized and a buffer for messages is created. A unique *identifier* is set to the MAC address of the microcontroller, but any other arbitrary sequence of bytes can be chosen (lines 1-5). After a TCP connection is established via *socket* the first two bytes of the data stream containing the following message length in bytes are read and this number is parsed and stored in *msg_len* (lines 5-8). If *msg_len* is 0 no DALI commands follow. This type of message is used to check if the gateway is "alive" or when the control application does not know its unique identifier. The gateway responds by sending a message containing the length of its identifier and its identifier (lines 9-12). If there are DALI commands to be processed that is *msg_len* > 0 the specified number of bytes is read into *buffer*. It is then checked if the first length of *identifier* bytes in *buffer* after the two bytes containing *msg_len* match the identifier assigned to that particular gateway. Otherwise, the message is dropped (lines 14-15). After that, each pair of bytes following the identifier in *buffer* is interpreted as one DALI command. Each command is sent via the DALI bus, and an attempt to receive a response is made (lines 16-18). If a valid DALI backward frame is received, the first byte of the command in *buffer* is set to 255 and the following second byte is set to the DALI backward frame that was received. Otherwise, the two bytes are zeroed (lines 19-23). After that *buffer* now containing the backward message/response is sent via *socket* and *socket* is closed.

Algorithm 1: Gateway message handling

```
1  server = Connect to Wi-Fi AP and start a TCP server at specified port;
2  DALI_bus = Initialize DALI interface;
3  buffer = ∅;
4  identifier = MAC address of the device;
5  while server is running do
6      socket = Wait until a connection to server occurs and accept it;
7      buffer = Read 2 bytes from socket;
8      msg_len = Parse message length from data in buffer;
9      if msg_len == 0 then
10         Write length of identifier in first 2 bytes of buffer;
11         Write identifier in the following bytes of buffer;
12         Send buffer using socket;
13     else
14         buffer = Read msg_len of bytes from socket;
15         if buffer contains identifier then
16             foreach two byte command in buffer after identifier until
                 msg_len do
17                 Send command using DALI_bus;
18                 response = Read one byte response from DALI_bus;
19                 if Read successful then
20                     Write 255 to first byte of command in buffer;
21                     Write response to second byte of command in buffer;
22                 else
23                     Write 0 to the 2 bytes of command in buffer;
24             Send buffer using socket;
25     Close socket;
```

A communication using this gateway algorithm is depicted in Figure 3.3. First, the Control application sends an empty message to find out if there is a gateway available and what identifier does it have assigned. The gateway responds with a message containing its identifier (in my case, its MAC address). After that, the control application can send a message containing the received identifier and DALI forward frames to be sent via the DALI bus. To which the gateway responds with a similar message containing the status byte (255 if a valid response was received 0 otherwise) and the DALI backward frames in place of the corresponding DALI forward frames from the previous message. More messages can then be exchanged in the same way.

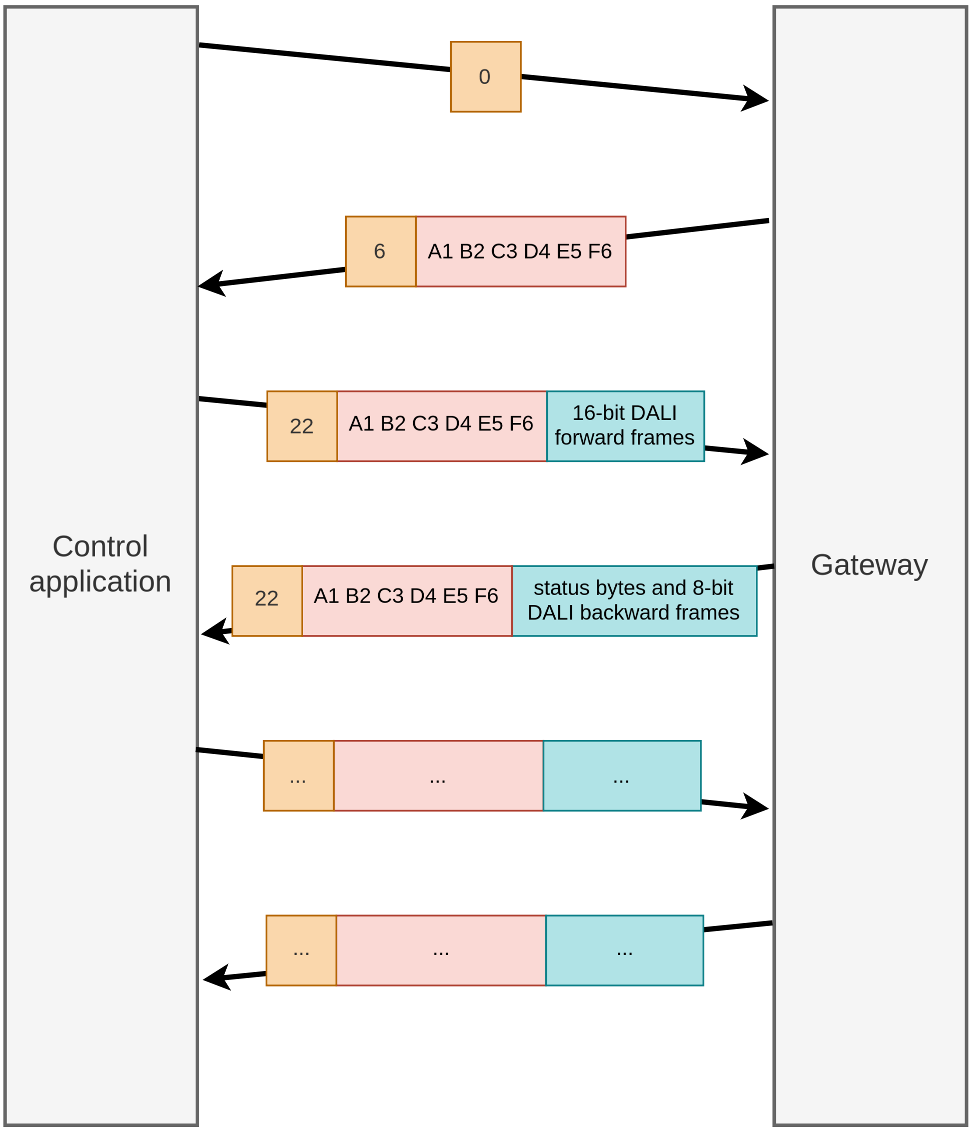


Figure 3.3: Communication example

3.1.3 ESP8266 gateway module

I have used a module with an ESP 8266 microcontroller [6] as a gateway between the DALI and the Wi-Fi network. This module was already available when I started working on this thesis and is not part of my work. Due to no being constructed specifically for this application, the module has some parts that are unused. The gateway module in its circuit diagram can be seen in Figures 3.4 and 3.5, respectively. The gateway consists of the following parts depicted in Figure 3.5:

- Top left - circuit for converting between 12V DALI bus signals and two 3.3V GPIO pins (DALI_TX and DALI_RX) of the ESP8266
- Top Middle - headers for external connections
- Top right - piezo element, which is not used in this thesis
- Bottom left - ESP8266 microcontroller board
- Bottom Middle - real time clock circuit, which is not used in this thesis
- Middle right - power source for the microcontroller and other 5V components
- Bottom right - current source for the DALI bus

Frames to and from the DALI bus are received using the two GPIO pins connected to the DALI bus. Two functions are implemented for the purposes of sending and receiving data in the microcontroller. The function *DaliTransmitCMD* accepts two bytes of a DALI command as an argument and transmits them to the DALI bus by setting the logic level of the DALI_TX with the appropriate delays according to the DALI protocol (see Figure 1.1). The receiving is done using the *DaliReadResponse* function. This function has two arguments, a pointer to a status byte and a pointer to a data byte. Then the DALI_RX pin is read. If a backward frame is received in a two-packet window, it is stored in the data byte and the status byte is set to 255. Otherwise, both bytes are zeroed, and the function returns. The rest of the ESP8266 code is implemented according to Algorithm 1 described previously.

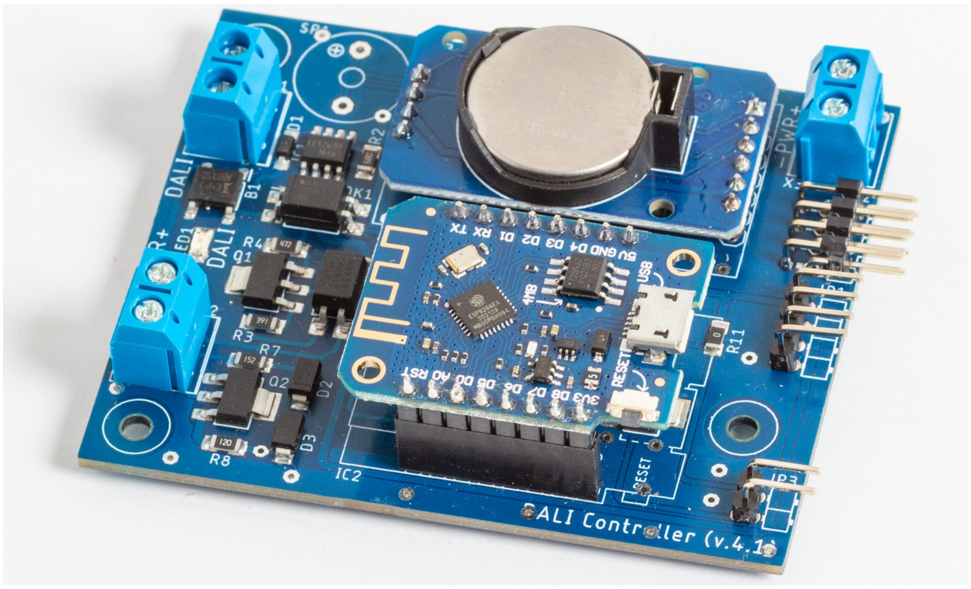


Figure 3.4: Gateway board with the ESP8266 microcontroller

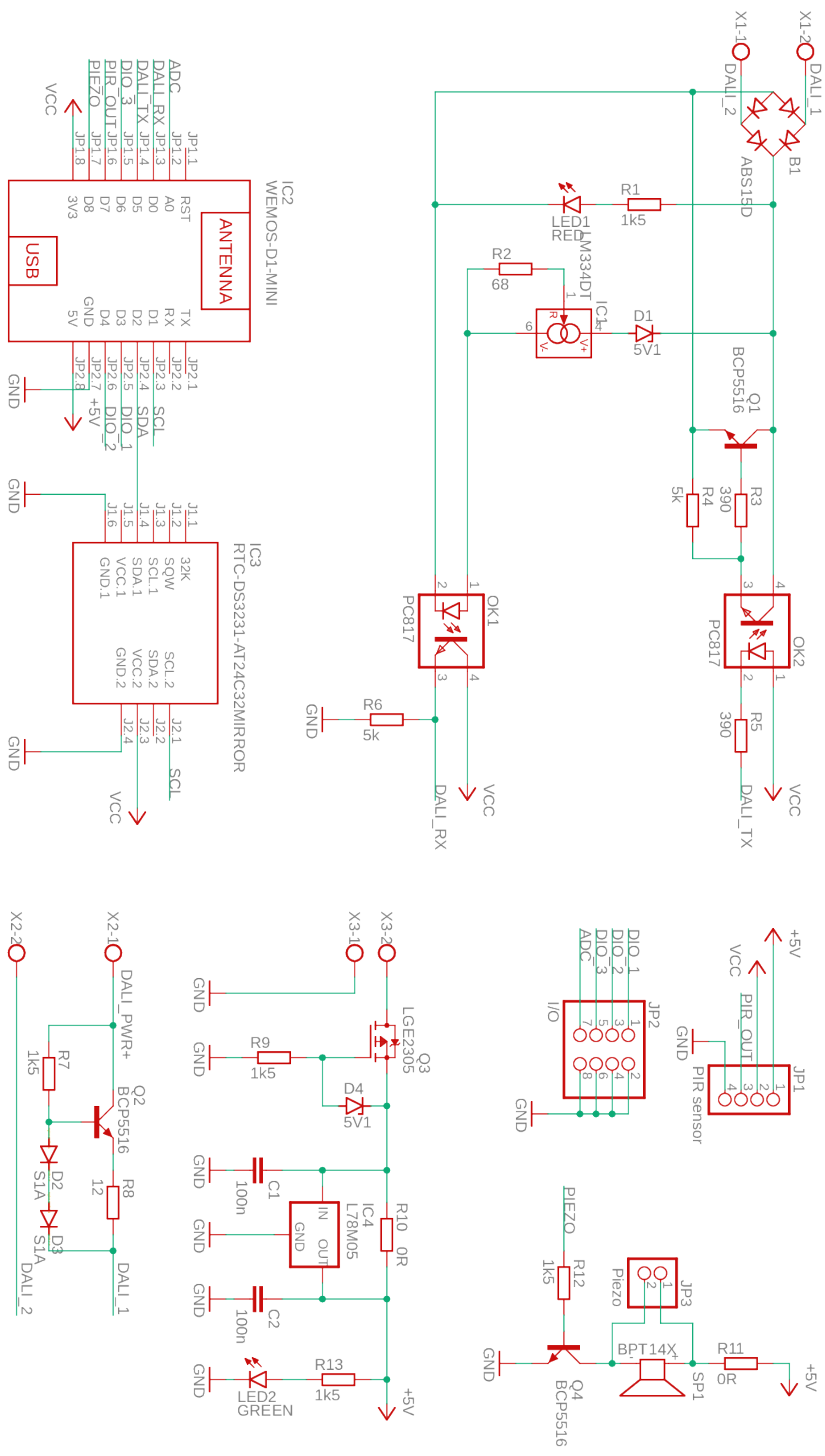


Figure 3.5: Gateway module circuit diagram

3.2 Central web server

I have decided to use a Raspberry Pi 4 (see Figure 3.6) as the central computer that runs the control web server. This single-board computer is small and has enough power to run the application. However, any other computer running Linux could be connected to the network with the gateways and used to run the web server. Other operating systems could also be used, though it would probably necessitate some changes in code. The server back-end is written in Python 3.12 using the Flask Web Framework. The front-end is written in HTML, CSS using Bootstrap 5, and Typescript using the Vue.js framework. The application is automatically run on startup of the Raspberry Pi from the script `start.sh` using the `systemd` tool.

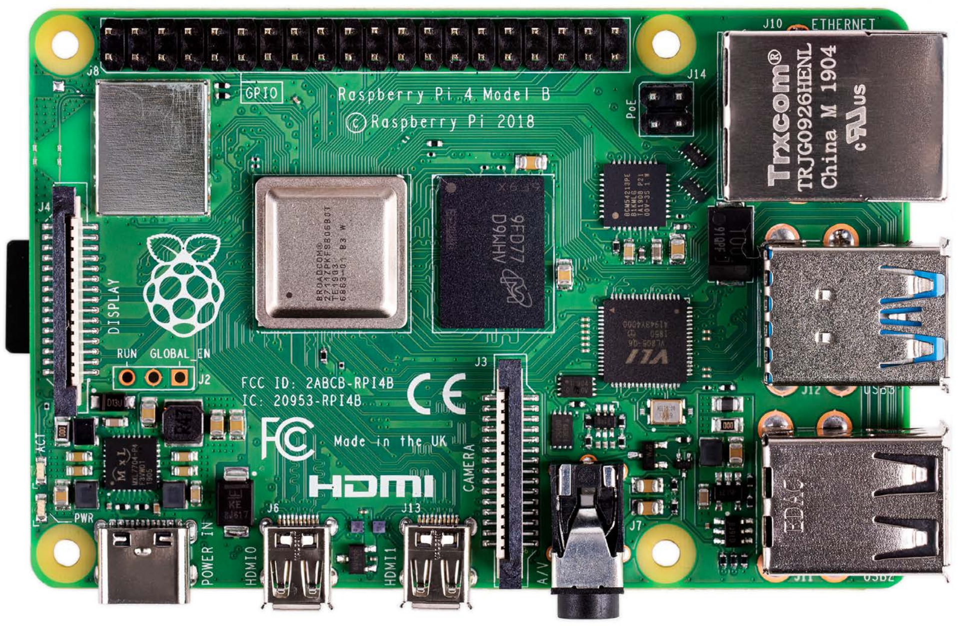


Figure 3.6: Central computer - Raspberry Pi 4 [22]

3.2.1 Front-end

The user interface is contained within a single web page where all the controls for the luminaires can be found. As the web page is implemented with the use of Bootstrap 5 all the elements and their layout are responsive. This allows the user interface to conform to the resolution of the device from which it is viewed. This user interface can be seen in Figures 3.7 (viewed from a full HD monitor) and 3.8 (viewed from a smart phone screen). Note that when using a smaller smartphone screen the whole interface does not fit on the screen at once and needs to be scrolled. For this reason, Figure 3.8 only contains a part of the interface.

The web interface structure reflects the relationships of the systems individual components. At the top of the page there is a dark bar containing general application elements such as a logo and controls, which are displayed independently of the configuration of the rest of the system. Each of the gray cards corresponds to a single gateway. Inside the gray cards white cards can be found. These each correspond to a luminaire connected to that particular gateway. The white cards then contain input elements that control the settings of their corresponding luminaire, such as its light intensity. If there are multiple devices/luminaires connected to a single gateway, additional input elements are displayed at the top of the gateway gray card. These inputs are used to control all the devices connected to the gateway at once. For example, if I want to turn off all the lights, I can use the red button at the top of the gray card and do not have to turn off each light individually using the white card inputs. Only inputs that all connected devices have in common are displayed, which in this case does not really matter as there is only one type of luminaire. However, it would be useful if more types of devices were connected. All the cards and input elements displayed on the Web page are automatically generated according to the current state of the system. If there is only one gateway active in the network, only one gray card is generated if another gateway is activated, another gray card is generated and so on.

In the example that can be seen in Figures 3.7 and 3.8, three gateways are active on the network, hence three gray cards (Gateway 1, 2, and 3) are displayed. Gateways 1 and 2 both have two connected luminaires, and Gateway 3 has only one. The control elements shown are kept simple for demonstration purposes and also no more was needed at this time; however, more inputs could be easily added. The inputs are following:

- Find Lights button - In the top right of the page there is a green „Find lights“ button. By clicking this button, the user can initiate a scan of the network which finds all the available DALI gateways. When these are found, the appropriate controls are shown on the web page. In the mobile view, this button is by default hidden under the "Hamburger" icon.
- Identifier text box - In the top left of each card, a text box containing the identifier of the luminaire/gateway can be seen. It is by default set to the gateway MAC address or luminaires short address, however, it can be easily changed by altering the contents of the text box.
- Init devices button - Below the identifier text box, a red „Init devices“ button can be found in the gray Gateways cards. This button performs

initialization of the DALI bus connected to that particular gateway and sets the short addresses of all the luminaires on the bus. It also creates the appropriate cards for these luminaires after the page is re-freshed.

- Intensity number input - Next to the identifier a number input can be found. Using this input, the intensity of the light source can be set. The input has a range of 0 - 100%. No other values can be entered. The intensity of 0% is equal to the light being OFF and 100% to full intensity.
- Intensity slider - Next to the number input is a slider which can be alternatively used to control the light intensity. Changing the slider also changes the number input. The slider also operates in the range 0-100% the same as the number input.
- Power OFF button - Next to the slider a red „Power OFF“ button can be found. This button can be used to turn OFF the light. This is the same as setting the intensity to 0%. This fact is reflected in the change of the slider and number input upon clicking the button.

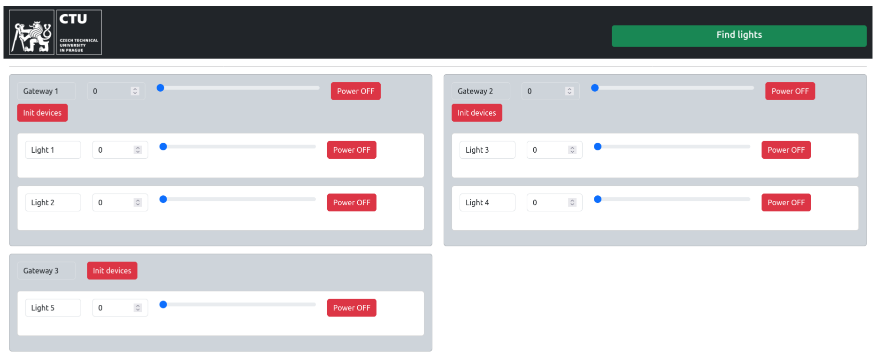


Figure 3.7: Control application web interface viewed from a PC in 1920 x 1080 pixels resolution.

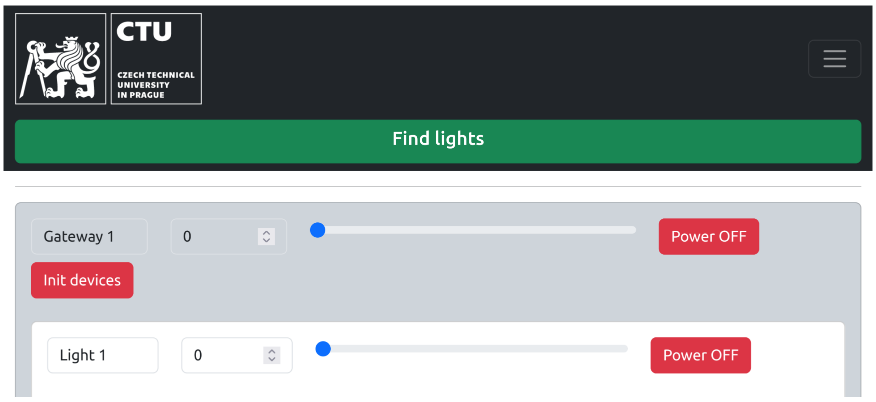


Figure 3.8: Part of the control application web interface viewed from a mobile phone in 915 x 412 pixels resolution.

The structure of the front-end code can be seen in Figure 3.9. The front-end is built on a default Vue template where *App* element is the root of the application. Under this root element *Navbar* element can be found. This is the dark gray bar on top of the page that contains the logo and green "Find lights" button. *DeviceControlPanel* element can also be found. This element encapsulates all the gateway and luminaire controls, but it is otherwise not visible on the web page. Under *DeviceControlPanel* element *Device* elements are recursively constructed. Under each of *Device* elements the user input elements are then added. These are *Toggle*, *Text*, *Number*, *Range*, and *Button* elements.

Device and user input elements are generated automatically. This is achieved using a Store which is a data structure in that provides a centralized access to data from across the web front-end. I have used an existing module implementing this functionality called Pinia. This Store is initialized from *device_store* script under *App* element. The Store is filled with a dictionary/JSON describing all the elements to be displayed. This data is then accessed from *Device* elements and displayed accordingly by Vue. The data is transferred from server back-end to the store/front-end using WebSockets [20] implemented in the SocketIO library which is supported by Pinia by default. This socket interface is initialized in *sockets* script and then used in the used in *device_store* script. An example of the JSON data structure describing which elements should be displayed can be seen below:

```
devices: {
  4C752534C48D: {
    controls: {Description_box: {..}, Intensity_box: {..}, ..}
    description: "Gateway 1"
      devices: {
        1: {
          controls: {
            Description_box: {type: "Text", .. }
            Intensity_box: {type: "Number", ..}
            Intensity_slider: {type: "Range", ..}
            Power_off: {type: "Button", ..}
          }
        description: "Light 1"
        id: ["4C752534C48D", "devices", "1"]
        }
      2: {description: "Light 2", controls: {..}}
    }
    id: ["4C752534C48D"]
  }
  E8DB84E0D442: {description: "Gateway 2", controls: {..}, ..}
  E8DB84E0D441: {description: "Gateway 3", controls: {..}, ..}
}
```

In this example the ".." represents data which was left out for the sake of readability. The example corresponds to the screenshot of the Web page in Figures 3.7 and 3.8. Each value in `devices` corresponds to one card displayed on the web page. `controls` then contain the description of user input elements. And `description` is the identifier that is displayed in the identifier textbox.

User input from the front-end is transferred to the back-end using the same WebSocket interface from the user input elements (*Button*, *Range* ...). To determine from which input element / device the input originated, `id` from the data structure described above is used.

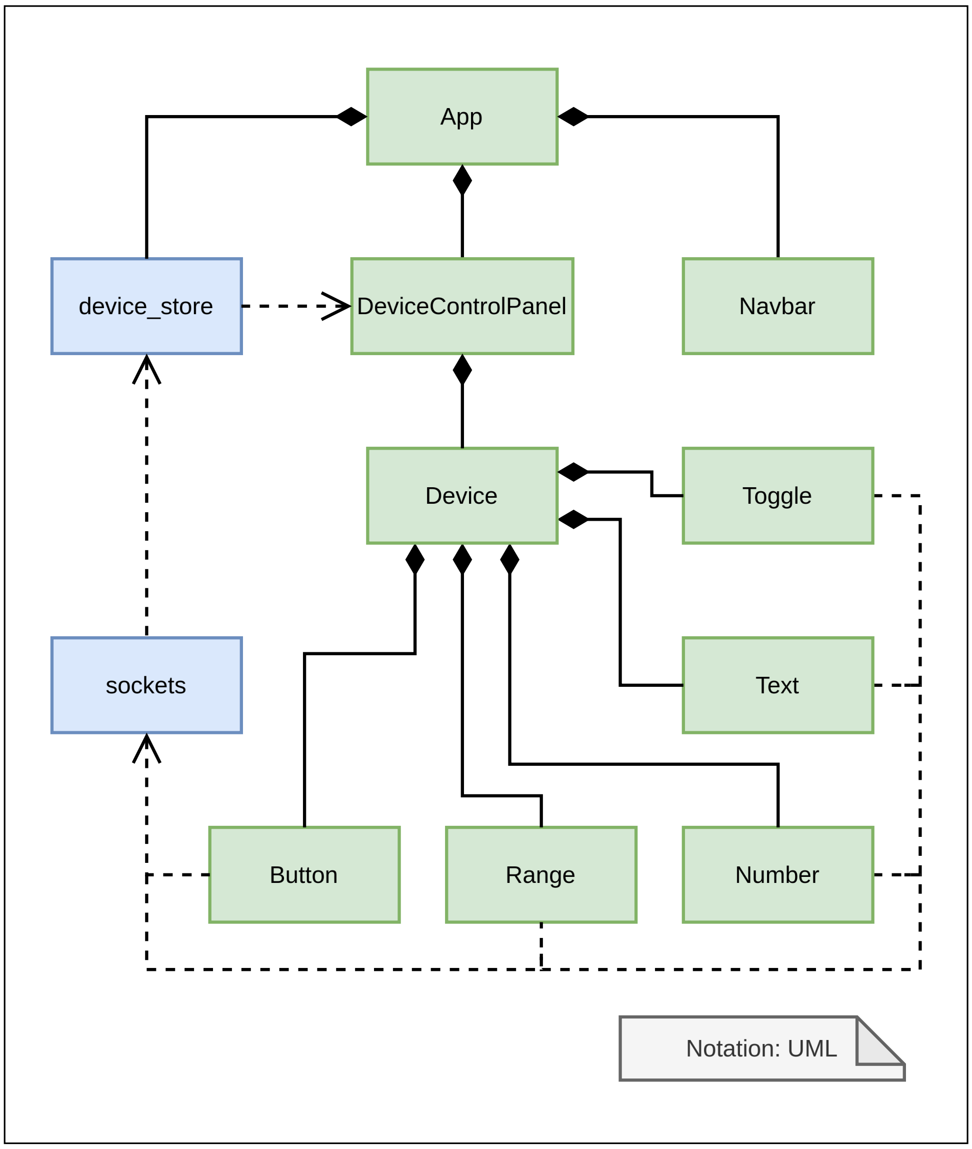


Figure 3.9: Vue app elements (green) and scripts (blue) UML diagram.

3.2.2 Back-end

The central web server back-end is implemented using the Flask web framework in Python version 3.12. The application structure is based on one of the commonly used Flask application structures [8]. The application is contained in one Python module called *app*. The module is then run by Python from the file `app.py`. In the `__init__.py` file of this module, the Flask application and classes used for controlling the light sources are initialized. Inside the `\static` folder the packaged files containing the Vue front-end application are stored so that they can be served by the Flask back-end which is done in `routes.py`. Front-end to back-end communication is implemented using the Flask-SocketIO module. The basic functions that handle receiving messages using WebSockets are implemented in `sockets.py`. The messages containing events from the inputs on the web interface are handled in `web_controls.py`. Updates to be displayed on the web interface generated by the back-end are also handled in this file. The functions that are called when a user input comes from the Web interface are implemented in *Controller* class. A Unified Modeling Language (UML) diagram of all classes and their relations can be seen in Figure 3.10. The classes are then described below.

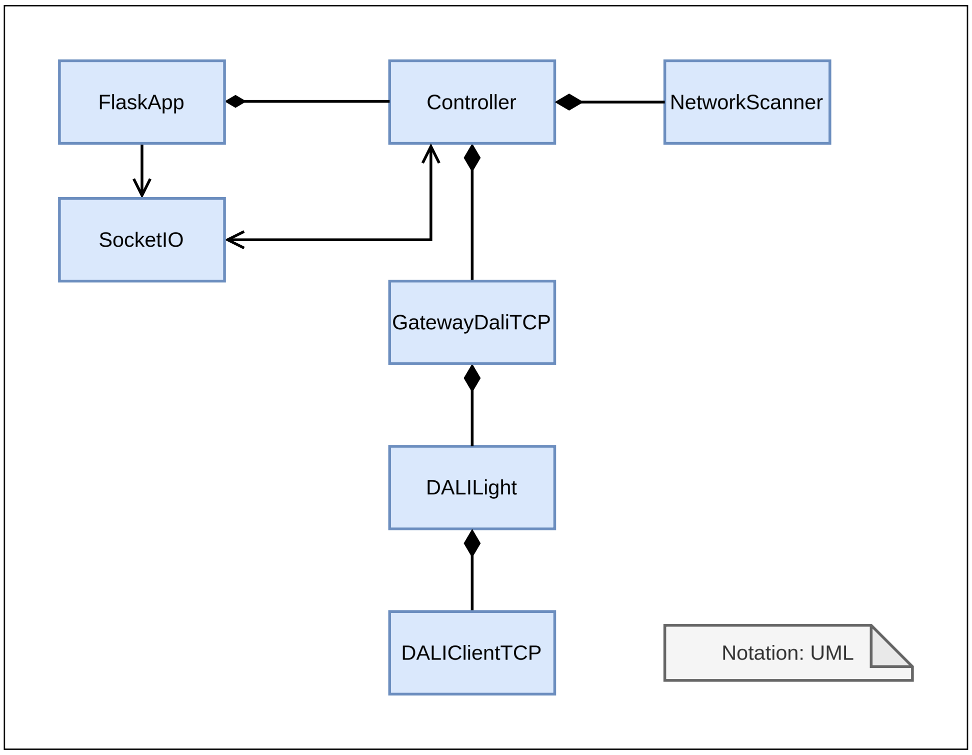


Figure 3.10: UML diagram of the Python classes used in the control application back-end

Controller class

The implementation of this class is located in `device_control.py` file. This is the main class and contains all the high-level logic used for the control of the light sources. It is instantiated only once in the `__init__.py` file. It has the following methods:

- *___init___()* method initializes its logger and all the data structures necessary for storing other subordinate class instances used for light source control. Instantiates *NetworkScanner* class which is later used to find available gateways. A queue to which updates that are to be displayed on the web interface are pushed. This queue is then read by a thread located in `web_controls.py`.
- *scan_for_devices()* method is called when the green "Find lights" button is pressed on the web interface. All available gateways connected to the network are then found using the *NetworkScanner* or loaded from `device_configurations.json` file if there are manually added gateways that were not found automatically. Using this data the appropriate *DALIGatewayTCP* and *DALILight* class instances are created and an update is sent to the front-end so the appropriate controls can be displayed.
- *get_device_description_dict()* method is called when the front-end needs to be updated. It returns the dictionary containing descriptions of all the control elements to be displayed on the web interface as described in the previous subsection 3.2.1.
- *control_device(device_id: List, control_data: Dictionary)* method is called when a user input for any of the gateways or luminaires is received from the front-end. The appropriate instance is found using *device_id* and *control_data* is passed to it for further processing.
- *set_device_description(device_id: List, description: String)* method is used to write data to `device_control.py` file.
- *get_device_description(device_id: List)* method is used to read data from `device_control.py` file.

NetworkScanner class

The implementation of this class is located in `scan_network.py` file. This class implements a simple algorithm to check the network in which the central computer is located for active gateways. It is instantiated only once in the *Controller* class. It has the following methods:

- *__init__()* method only initializes the logger for this class.
- *scan(address_range: List, ports: List)* method checks the specified ports of IP addresses in a specified range. First the individual addresses are "pinged" and if a response is received an empty messages as defined in 1 are sent to the specified ports. If a response from a gateway is received its address and port are returned from this method. All of this is done in parallel using the Asynchronous I/O Python library.

DALIGatewayTCP class

The implementation of this class is located in `device_control.py` file. This class provides a programming interface to interact with individual gateways. It is instantiated once per gateway in the *Controller* class. It has the following methods:

- *__init__(controller: Controller, ip: String, port: Int, identifier: String)* method first initializes its logger. Then instantiates *DALIClientTCP* class with the specified port and IP address. After that the gateways specification is stored in the `device_configurations.json` file if the gateway does not already have an entry in this file. Then *DALILight* class instances specified in `device_configurations.json` are created if there are any. Then a dictionary with a description of user inputs that should be displayed on the web interface for that particular gateway (not including its connected luminaires) is initialized. This dictionary is used by the *Controller.get_device_description_dict()* method.
- *handle_control(control_data: Dictionary)* method is called from *Controller.control_device()*. In this method the user input is parsed and required actions are executed. If the input is a change of the gateways identifier it is written to `device_configurations.json` and the update is sent to the front-end via the *Controller* queue. If the "Init devices" button was pressed an initialization of the DALI bus connected to that gateway is done using its *DALIClientTCP* class instance. *DALILight* class instances are then created for each device found on the bus during initialization. If its input for one or more luminaires it is passed to the appropriate *DALILight* class instances.

DALILight class

The implementation of this class is located in `device_control.py` file. This class provides a programming interface to interact with individual light sources. It is instantiated once per light source in the *DALIGatewayTCP* class. It has the following methods:

- *___init___(controller: Controller, gateway: GatewayDaliTcp, identifier: String)* method initializes its logger. After that, the light sources specification is stored in the `device_configurations.json` file if the light source does not already have an entry in this file. Then a dictionary with a description of user inputs that should be displayed on the Web interface for that particular light source is initialized. This dictionary is used by the *Controller.get_device_description_dict()* method.
- *handle_control(control_data: Dictionary)* method contains the executive code for user input that is related only to a single light source. If the input is a change of the light sources identifier it is written to `device_configurations.json`. The only other option is a change of light intensity either via the slider, the number input or the "Power OFF" button (sets intensity to 0). The intensity is converted from the 0-100% range to the 0-254 range as per DALI specification. This number is then sent to the DALI bus using the gateways *DALIClientTCP* instance via the short address DAPC DALI command. The updated values are then sent to the front-end via the *Controller* queue.

DALIClientTCP class

The implementation of this class is located in `py_dali_tcp.py` file. This class implements an interface for the DALI bus. In this class a method for sending DALI forward frames and receiving DALI backward frames is implemented. Standard DALI commands and advanced procedures are also implemented in this class. DALI messages and hexadecimal constants used in these messages are written in the `dali_commands.py` and `dali_constants.py` files respectively. The three files can be used as a standalone Python module for DALI communication via the gateways. The class has the following methods:

- *___init___(self, ip: String, port: Int, identifier: String or None)* method initializes its logger. If the gateways identifier is not specified, a TCP socket is created to the specified IP address and port, and an empty message is sent. The identifier is then retrieved from the response and stored.

- *send_commands(commands: DALICmd)* method has a list of individual DALI command instances as an argument. From the data contained in this list, a forward message is constructed as specified in Chapter 2 and sent to the gateway through a TCP socket. The response is then parsed according to each command specification, and the parsed responses are returned as a list of Python values. This is the implementation counterpart to Algorithm 1. An example usage where a max level is set to all lights on the bus may look like this:

```
client = DALIClientTCP("10.0.0.2", 4242, "1")
client.send_commands(BroadcastDAPC(254))
```

- *init_devices(address: Int)* method performs full initialization of the DALI bus. All device short addresses are reset and than assigned anew starting from the address specified as an argument. This is done using *send_commands()* method. The list of assigned addresses is then returned.
- There are many other methods implementing DALI commands in this class. Such method are for example *broadcast_dapc* or *address_query_actual_level*. These methods usually aggregate a few instances of *DALICmd* class send it via *send_commands()* method and return a response if appropriate. The implementations of these methods are very simple and self-explanatory, therefore I will not describe each individual method here.

DALICmd class

The implementation of this class is located in `dali_commands.py` file. This class serves to encapsulate the binary data of each individual DALI command. This data is stored in *data* attribute. It has a *parse* method that by default only returns `None`. Other command classes inherit from this class and write their corresponding binary data to *data* attribute in their *__init__* methods and may override the default parse method if the command is expected to have a backward frame containing data that will need to be parsed for further use in Python. An example of such class is *BroadcastDAPC*.

Data storage

The current configuration of devices/light sources known to the control application is stored in the `device_configurations.json` file. Such JSON configuration file has a defined structure, an example can look like this:

```json
"4C752534C48D": {
    "description": "Gateway 1",
    "ip": "192.168.1.121",
    "port": "4242",
    "devices": {
        "1": {
            "description": "Light 1"
        },
        "2": {
            "description": "Light 2"
        }
    }
},
"E8DB84E0D442": {
    "description": "Gateway 2",
    "ip": "192.168.1.120",
    "port": "4242",
    "devices": {
        "1": {
            "description": "Light 3"
        },
        "2": {
            "description": "Light 4"
        }
    }
},
"E8DB84E0D441": {
    "description": "Gateway 3",
    "ip": "192.168.1.120",
    "port": "4242",
    "devices": {
        "1": {
            "description": "Light 5"
        }
    }
}
```

Each key is a MAC address of one gateway with a specification of that gateway as its value. `"description"` is the gateways identifier displayed on the web interface. `"ip"` and `"port"` specify the network location of the gateway. `"devices"` contain the specifications of light sources connected to that particular gateway. There each key is a short address of the DALI light source and `"description"` is again the identifier of the light displayed on the web interface. According to this file, the appropriate devices and controls are then displayed on the web interface. Provided that the gateways are connected. The contents of this file are modified automatically if new devices are found after pressing either the "Find lights" or "Init devices" button on the Web interface. Or it can be modified manually, for example, if initialization of the DALI bus is not desired.

Logging

The whole back-end application has function call logging setup. This is done using the Python Logging module. In each file or class, a logger from this module is initialized. This logger is then used to output from a *@log* Python decorator. The function name, the module from which it was called, and its arguments are logged. If an exception occurs, it is also logged. The log is saved to the `log.log` file. This decorator is implemented in the `log.py` file.

Chapter 4

Functionality verification

After implementing the system, I have verified that everything is working correctly. I have tried passing different edge-case inputs to the web interface. Of course, I found some minor bugs such as inputting a decimal number into the light level number input caused an exception because the back-end was expecting an Int. However, these were quickly fixed, and the web server behaved as expected.

The main part to which I dedicated the most attention was the communication between the Web server, the gateway, and the light source connected to the DALI bus. I verified that it works correctly on multiple scenarios. One of the scenarios looked like this (see Figure 4.1): I sent an empty message to a gateway and read the identifier with which it responded. I checked that the identifier matches the MAC address of the gateways that I knew. Then I sent a message using this identifier that contains an address DAPC DALI command. I knew that the address of the light source on the other side of the gateway was 1, so the command was to set the current light level of the light source with address 1 to the value of 142. I actually found that the minimum value that the particular DALI drive supported was around 70, therefore I selected a higher value. After that, I checked that I received the expected response of identifier and zeros. After the light source lit up to the expected brightness, I sent a query actual level DALI command to the same address to see that the level the light source was set to actually matched the 142 I sent earlier.

During all of that I monitored the voltage on the DALI bus with an oscilloscope. The actual bytes the messages consisted of can be seen in Figure 4.2. Only the last two bytes of these messages were transmitted over the DALI bus in the case of the forward frames. In the case of the backward frames, it was only the last one byte. In Figure 4.3 it can be seen that the Manchester encoded bits picked up by the oscilloscope match the expected value of 02 8E of the first DAPC forward frame sent. After that, no backward frame was sent, as expected. Then in Figure 4.4 the 03 A0 matching the query actual level command was sent. To that the expected response of 8E (142 decimal) was received. Therefore, I concluded that the protocol and the system worked as intended.

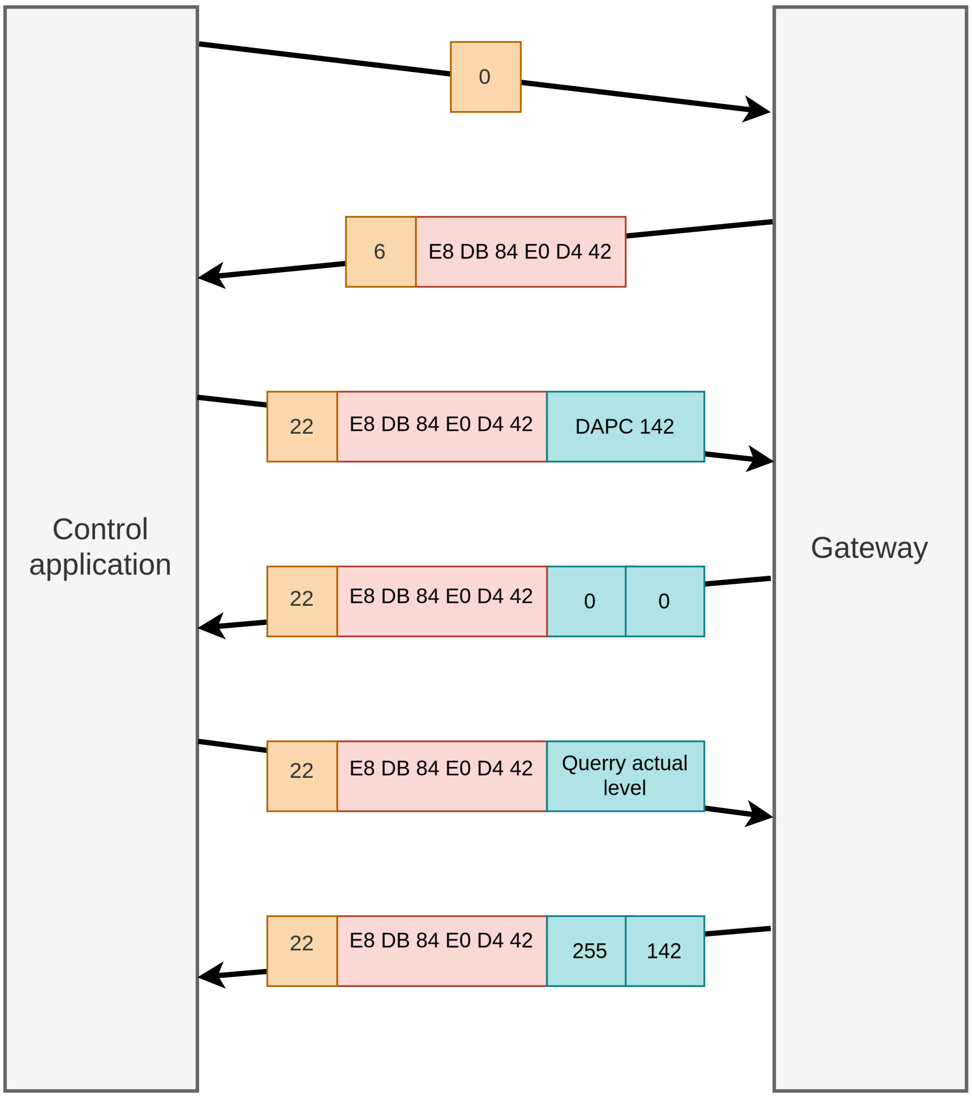
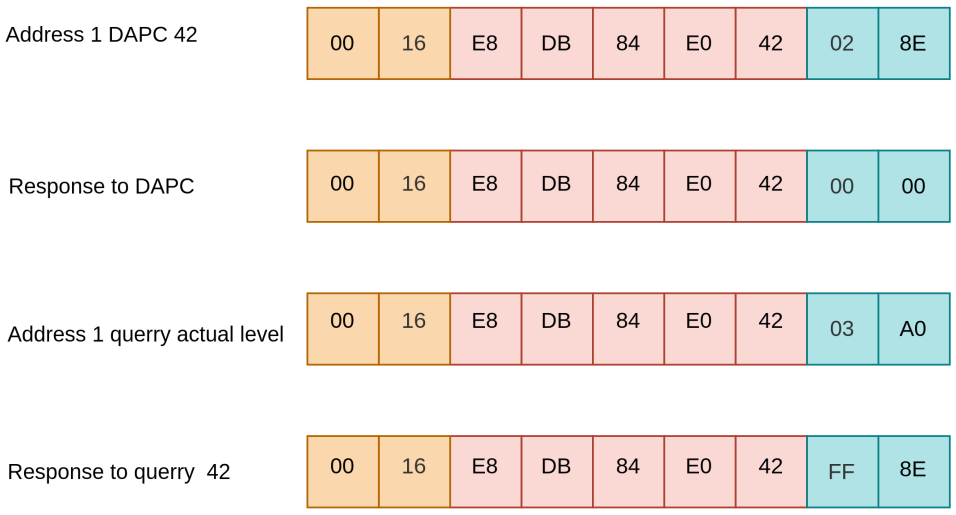
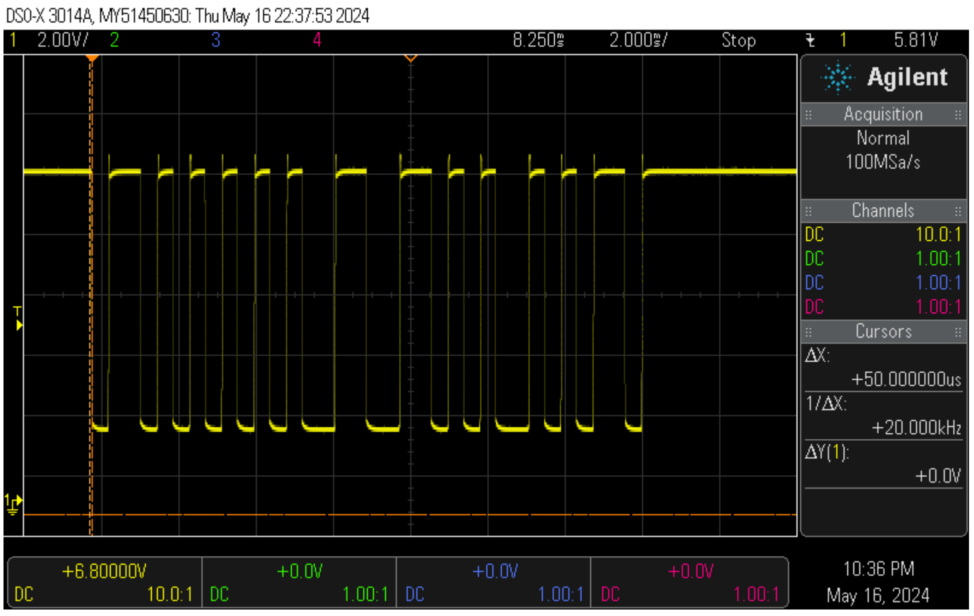


Figure 4.1: A diagram of messages sent between the gateway and control application in the testing scenario



Address 1 DAPC 42	00	16	E8	DB	84	E0	42	02	8E
Response to DAPC	00	16	E8	DB	84	E0	42	00	00
Address 1 querry actual level	00	16	E8	DB	84	E0	42	03	A0
Response to querry 42	00	16	E8	DB	84	E0	42	FF	8E

Figure 4.2: Messages sent during testing depicted by individual bytes in hexadecimal format



DSO-X 3014A, MY51450630: Thu May 16 22:37:53 2024

1 2.00V/ 2 3 4 8.250s 2.000s/ Stop 1 5.81V

Acquisition
Normal
100MSa/s

Channels
DC 10.0:1
DC 1.00:1
DC 1.00:1
DC 1.00:1

Cursors
ΔX: +50.000000us
1/ΔX: +20.000kHz
ΔY(1): +0.0V

+6.80000V DC 10.0:1 | +0.0V DC 1.00:1 | +0.0V DC 1.00:1 | +0.0V DC 1.00:1

10:36 PM
May 16, 2024

Figure 4.3: Oscilloscope measurement of DALI bus transmission of DAPC 142 to short address 1 forward frame

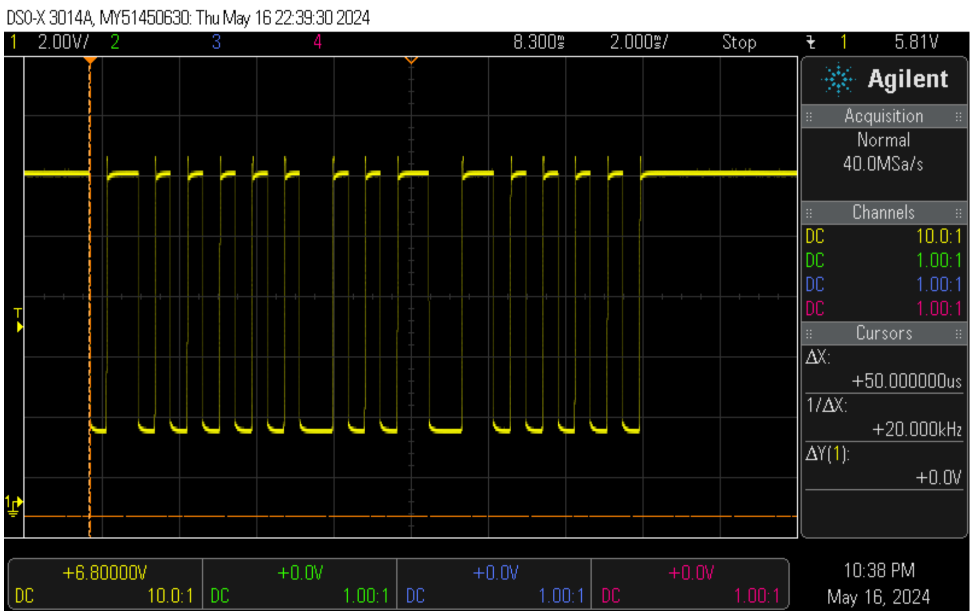


Figure 4.4: Oscilloscope measurement of DALI bus transmission of query actual from short address 1 level forward frame

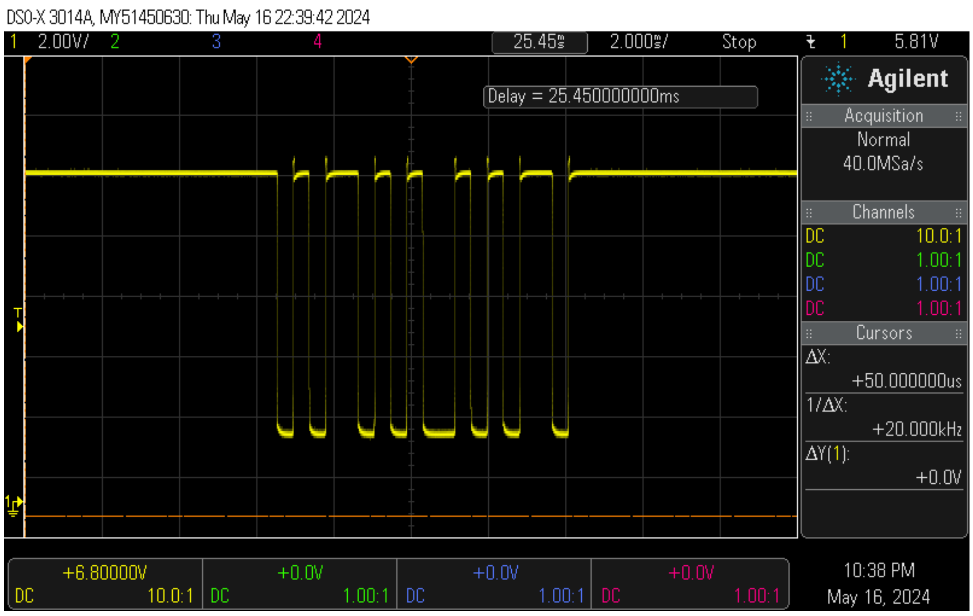


Figure 4.5: Oscilloscope measurement of DALI bus transmission of query actual from short address 1 backward frame

Chapter 5

Application

The control system designed and implemented in this thesis is being used in a research project of Architectural and Festive Lighting in the Context of Historic Buildings and Spaces (NAKI). One of the aims of this project is investigating how to implement architectural lighting of historical buildings in the context of the surrounding environment and ecology. That is how to light the building for the best experience by people viewing it and also for minimal negative impact on the surrounding natural environment.

The system was required to control four independently positioned light sources powered by a battery bank. The lights were to be first used for some laboratory measurements and then moved outside to illuminate part of a castle. The light sources consisted of around 20 individual LED chips with varying light characteristics mounted on heat sinks and four DALI drivers for those LEDs. The intention was to have only four lights active at the same time and be able to swap the LEDs quickly.

Therefore, I have assembled four LED driver boxes for those four light sources. The boxes were designed to be placed in an outside environment. One of the boxes can be seen in Figure 5.2. Each box contains the following parts:

- The DALI LED driver - this cannot really be seen in the Figure as it is located on the bottom of the box.
- A 12V power supply for the DALI bus and gateway electronics - located in the top part of the box.

- The ESP8266 gateway module - located in the middle of the box on top of the driver.
- A standard power connector type IEC C14 to connect the 12V power supply and LED driver to the power grid - located on the right side of the front panel of the box.
- An XLR female connector for connecting the LEDs to the DALI driver in the box - located on the left side of the front panel of the box.

The LEDs were fitted with an XLR male connector. This allows for connecting the LEDs to the boxes using a standard XLR extension cable and provides a reliable connection, as the LEDs are expected to be swapped frequently. The box is then connected to the battery power supply using a standard power cable. The Raspberry Pi was housed in a similar box with a power supply and the same power connector. The completed setup can be seen in Figure 5.1

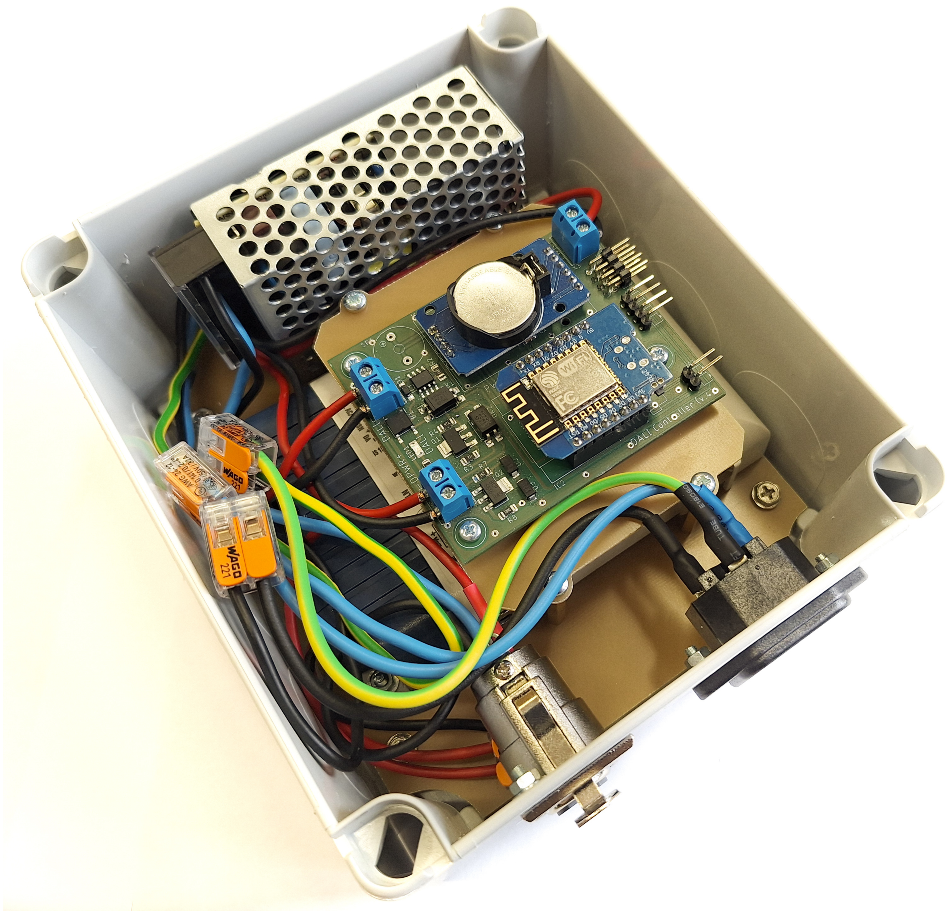


Figure 5.1: Physical realization of the proposed DALI gateway. The module with ESP 8266 microcontroller and DALI bus circuitry can be seen on top. Photo courtesy of the NAKI group.

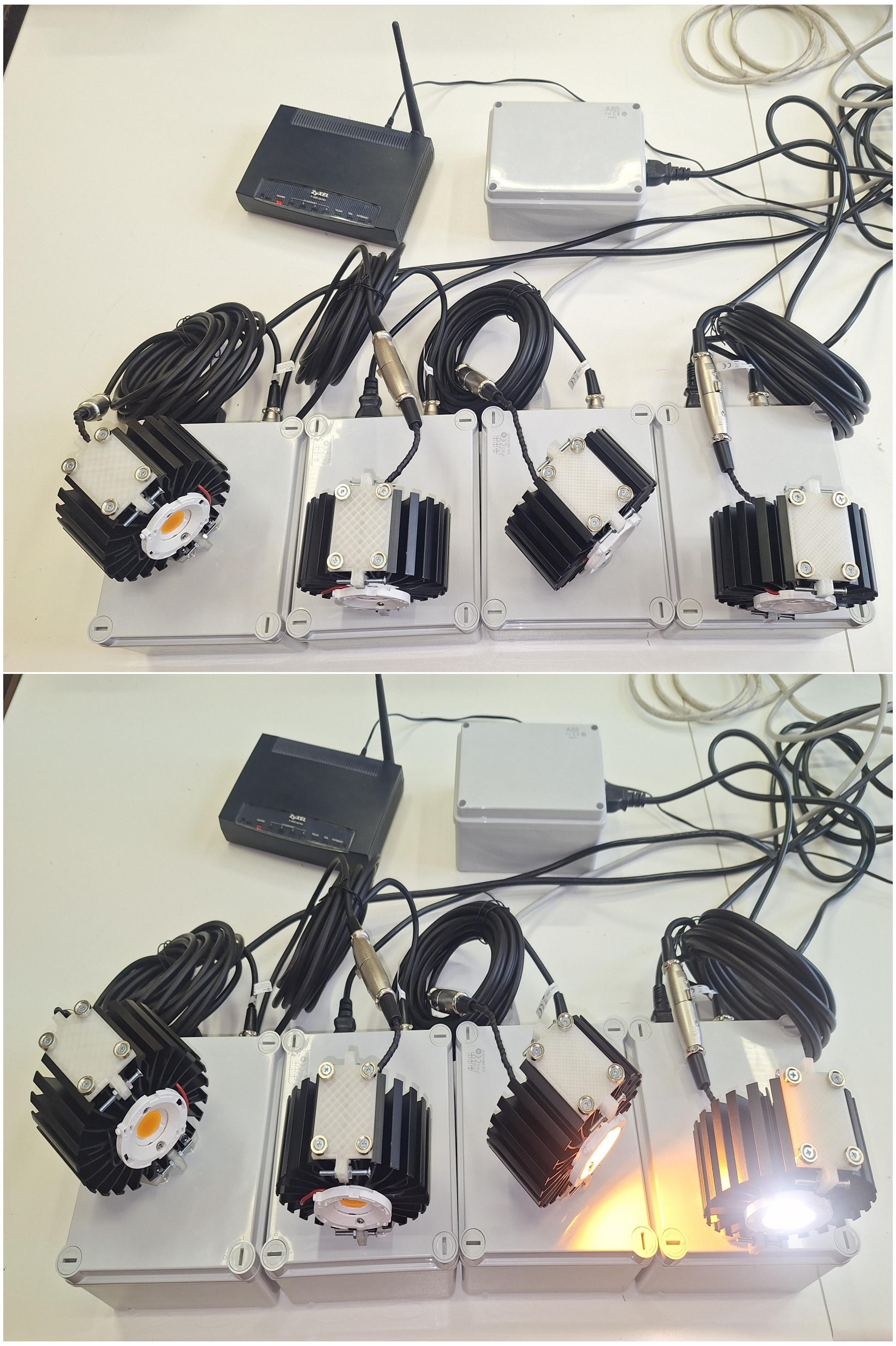


Figure 5.2: Completed setup with four driver boxes a Raspberry Pi box a Wi-Fi router and LEDs. Photo courtesy of the NAKI group.

I also slightly modified the web interface for this application after consulting with the team that would be using it. As there is ever only one light source connected to each gateway, the gray gateway cards have been removed from the interface. I also removed the input labels because the interface is very simple. In this way, I was able to save space and fit more of the interface on a screen of a smart phone, as smart phones were the only type of device used in this application. The modified interface can be seen in Figure 5.3.

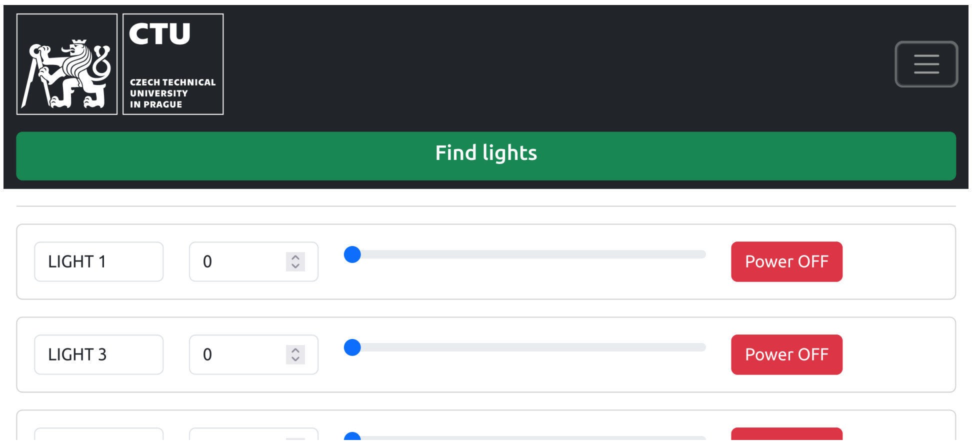


Figure 5.3: Modified control system web interface.

The control system has already been used in several laboratory measurements. Photos from some of the experiments can be seen in the Figures below. The first two Figures 5.4 and 5.5 depict a setup where light spectra resulting from a combination of multiple LEDs with different color temperatures were measured. The light sources were positioned on tripods, and some of the control system boxes can be seen behind them. In the second two Figures 5.6, and 5.7. A study of how color temperature and light intensity of the lights illuminating the little 3D printed bird affect human ability to perceive depth. These are some of the preliminary experiments that the team has conducted to gather data about the light sources that they have. These learnings will then be used to actually light a historical building. It is evident that the full capabilities of the control system were not used in these measurements and they could have been conducted with a much simpler wired setup. However, the measurements were invaluable for testing the control system and proving that it would be capable of running the light sources in any other application, including the planned lighting of historical buildings.

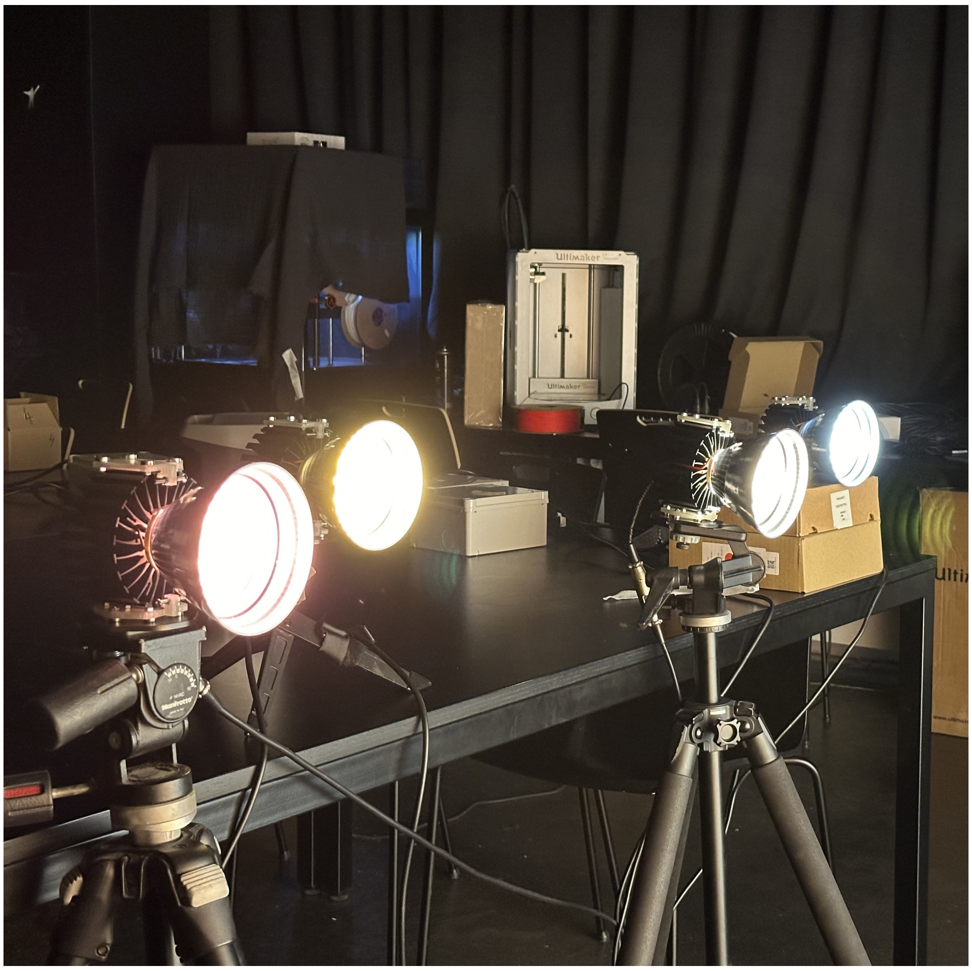


Figure 5.4: Light sources connected to the control system viewed from the front. Photo courtesy of the NAKI group.

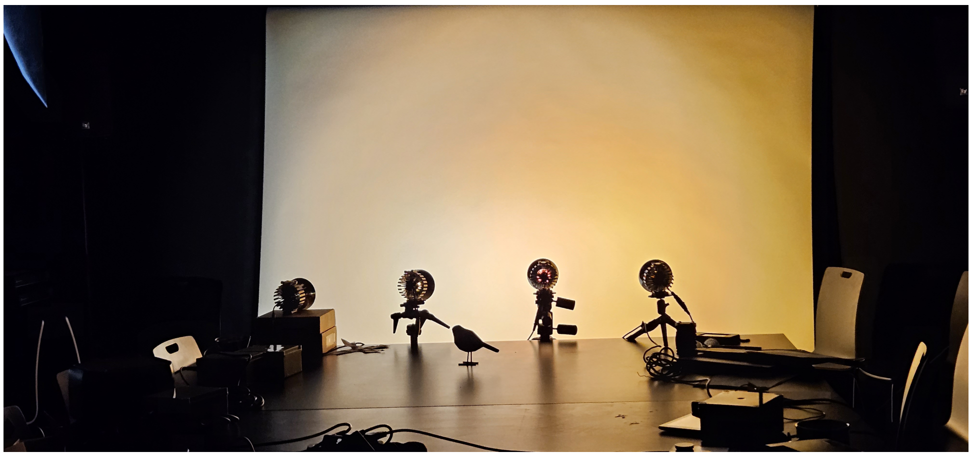


Figure 5.5: Light sources connected to the control system pointed to a canvas - setup for measuring light spectrum of mixed LEDs. Photo courtesy of the NAKI group.

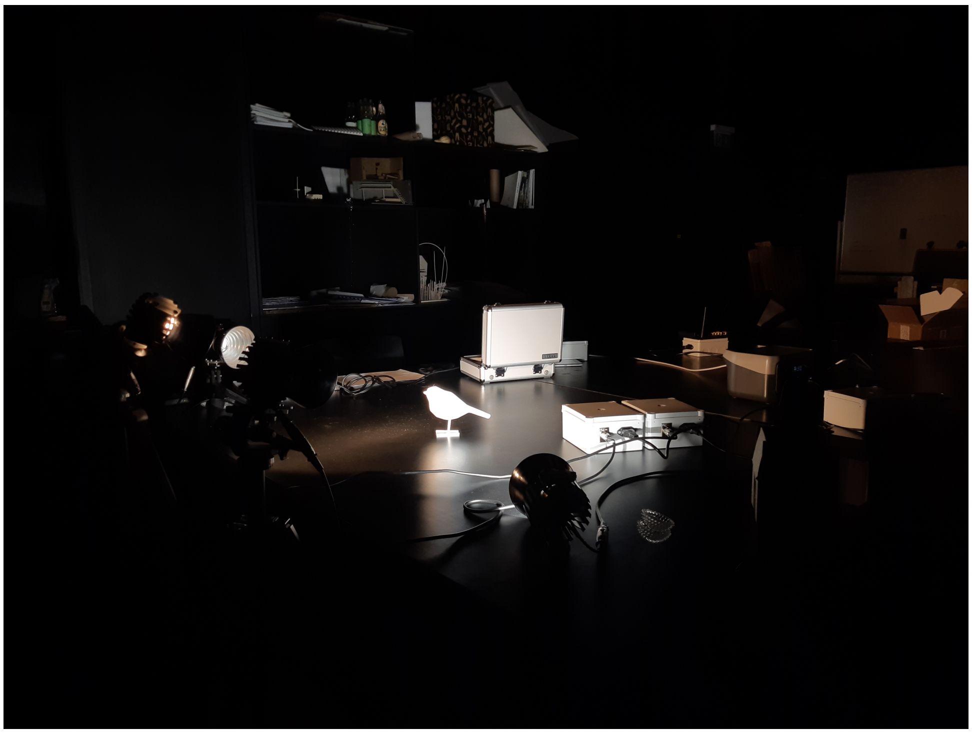


Figure 5.6: Light sources setup for studying influence of different lighting characteristic on human depth perception. Photo courtesy of the NAKI group.

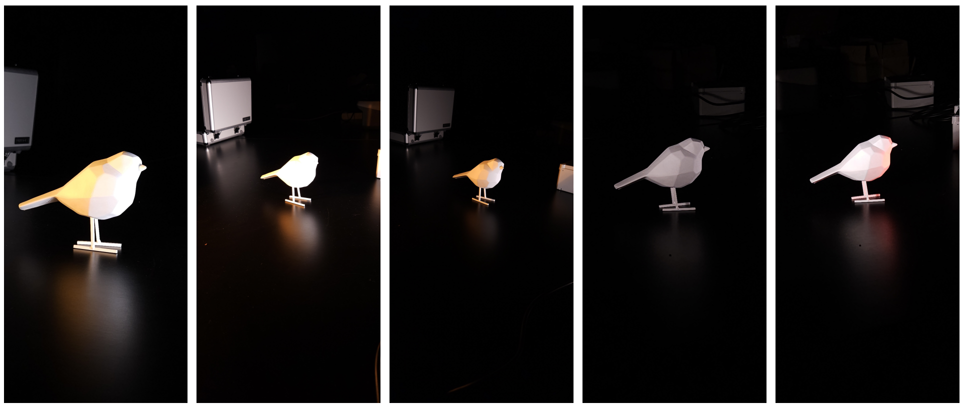


Figure 5.7: Test figure used in depth perception study illuminated with different spectra and light intensities. Photo courtesy of the NAKI group.

Chapter 6

Conclusion

As awareness of the influence of lighting, its properties and changes during the day on us humans grows, more sophisticated control systems for light sources are needed. Such systems must support independent control of many light sources, to allow for setting the lighting conditions to best suit human needs. This entails having control over the intensity and spectrum of the light sources illuminating different parts of a space.

In this thesis, I have proposed a design of a control system that would meet the needs of modern lighting. The control system is designed for the wireless control of independent light sources driven by DALI. It consists of a central control web application/user interface which uses Wi-Fi to connect wirelessly via gateways to the DALI bus controlling the light sources. A communication protocol defined specifically for this application is used for communication between the control application and the gateways. It is built on TCP and allows for directly transmitting DALI protocol forward and backward frames through the wireless network. This architecture is advantageous, as this type of protocol inherently supports any DALI commands without the need to change anything in the gateway implementation, provided that the frames have a supported length. The light sources are also controlled from a Web interface, which allows the user to access the system from any device of their choice that has a web browser. This combined makes the control system very flexible and suitable for various applications. The use of Wi-Fi and TCP theoretically also allows the system to work over the Internet, thus allowing for remote control from anywhere in the world.

I have physically implemented the control system according to this design. I have written the Web interface using the Flask Web framework in Python and Vue.js. I have hosted this web server on a Raspberry Pi. I have implemented the gateways using an already existing module that has the electronics necessary to send and receive messages over the DALI bus and is controlled by an ESP8266 microcontroller that has Wi-Fi capabilities. I then verified the functionality of this system by connecting it to light sources with DALI drivers. I tested that the commands sent via the interface yielded the expected results and measured the traffic on the DALI bus with an oscilloscope to make sure that the frames are being sent and received correctly. I have concluded that the system works as designed and meets all the criteria defined in the thesis assignment.

I made a fully fledged version of the control system complete with containers for the electronics, necessary power supplies, cables, and connectors. This setup consists of a Wi-Fi router, a box with the Raspberry Pi, and four gateway boxes each with a DALI driver for one light source integrated. This setup was requested for the Architectural and Festive Lighting in the Context of Historic Buildings and Spaces project. The setup was already successfully used by the team running this project for several laboratory measurements and will soon be deployed on a castle for outdoor measurements. In the future, I would suggest the following improvements/changes to the control system:

- Adding support for advanced DALI functionality - The control system only supports basic DALI commands using the 16-bit forward frame and 8-bit backward frame. The system does not support any sensors or user input devices other than the control application. Therefore, it would greatly benefit from implementing support for longer frames and DALI 2 functionality.
- Securing the TCP based protocol - Currently the data is transmitted over the wireless network unencrypted. This is not advisable and could be easily resolved with the use of Transport Layer Security protocol (TLS).
- Securing the web server - The web server currently uses the HTTP protocol and should really be switched to using HTTPS. It also does not have password protection which should also be implemented.
- Improving the front-end - Currently controls to be displayed on the interface are specified in the back-end Python code. Having a single front-end element which could be modified from the back-end code was advantageous in the prototyping phase. However I would implement the interfaces completely in the front-end in the future, as it would provide better customizability for the interfaces an also better scalability.

This control system design and implementation was also submitted as a conference paper for the 29th international conference on applied electronics. The paper is currently in the review process.

Appendix A

Bibliography

[1] Peter Robert Boyce. *Human factors in lighting.* CRC Press, 2003.

[2] C Cajochen, M Freyburger, T Basishvili, C Garbazza, F Rudzik, C Renz, K Kobayashi, Y Shirakawa, O Stefani, and J Weibel. Effect of daylight led on visual comfort, melatonin, mood, waking performance and sleep. *Lighting Research & Technology*, 51(7):1044–1062, 2019.

[3] DALI specification by DigiKey. https://www.digikey.ro/en/articles/designing-wired-lighting-control-networks-to-dali-standard Acessed 2024-05-17.

[4] R091 DALI / Modbus TCP converter. https://products.domat-int.com/en/communication-converters/463-m090.html Acessed 2024-05-17.

[5] Petr Douda, Jan Havlík, and Lenka Maierová. Independent light source control using wi-fi to dali gateways. unpublished, 2024.

[6] ESP8266 microcontroller. https://www.espressif.com/en/products/socs/esp8266 Acessed 2024-05-17.

[7] Foxtron Interface pro bezdrátové ovládání DALI sběrnice. https://www.foxtron.cz/e-shop/sbernice/dali/daliweb Acessed 2024-05-17.

[8] Miguel Grinberg. *Flask web development.* " O'Reilly Media, Inc.", 2018.

[9] IEC 62386-101:2022, Digital addressable lighting interface - Part 101: General requirements - System components. Standard, International Electrotechnical Commission, 2022.

[10] IEC 62386-104:2019, Digital addressable lighting interface - Part 104: General requirements - Wireless and alternative wired system components. Standard, International Electrotechnical Commission, 2019.

[11] IEEE 802.11-2020, IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Standard, IEEE, 2020.

[12] IEEE 802.11-2020, IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Standard, IEEE, 2020.

[13] IEEE 802.15.1-2005, IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN). Standard, IEEE, 2005.

[14] IEEE 802.15.4-2020, IEEE Standard for Low-Rate Wireless Networks. Standard, IEEE, 2020.

[15] William DS Killgore. Effects of sleep deprivation on cognition. *Progress in brain research*, 185:105–129, 2010.

[16] Michael S Langer and Heinrich H Bülthoff. Depth discrimination from shading under diffuse lighting. *Perception*, 29(6):649–660, 2000.

[17] LoRa Specification. https://lora-alliance.org/about-lorawan/ Acessed 2024-05-17.

[18] MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3. https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf Acessed 2024-05-17.

[19] RFC 2616, Hypertext Transfer Protocol – HTTP/1.1. Standard, RFC, 1999.

[20] RFC 6455, The WebSocket Protocol. Standard, RFC, 2011.

[21] RFC 793, TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION. Standard, RFC, 1981.

[22] Raspberry Pi 4 Model B. https://rpishop.cz/raspberry-pi-4/1598-raspberry-pi-4-model-b-4gb-ram.html Acessed 2024-05-17.

[23] Guido van Rossum, Barry Warsaw, and Nick Coghlan. Style guide for Python code. PEP 8, Python Software Foundation, 2001.

[24] Z-wave specification. https://z-wavealliance.org/development-resources-overview/specification-for-developers/ Acessed 2024-05-17.

[25] Zencontrol. https://zencontrol.com/ Acessed 2024-05-17.

Appendix B

Control system code

The code of the central web server implementation can be found attached to this thesis in the directory `/code/server`. The back-end is written in Python 3.12 and formatted according to PEP 8 [23]. It is located in the `/code/server/backend` directory. The project package requirements can be found in the file `requirements.txt` in the same directory. The front-end is written in Typescript using Vue.js. It is located in the `/code/server/frontend` directory.

In the `/code/server/device_configuration.json` file the light source and gateway configuration are stored. The web server can be run using the `start.sh` script in the same directory.

The code of the ESP8266 gateway implementation can be found attached to this thesis in the directory `/code/gateway`. It is written in C++ using the Arduino framework.