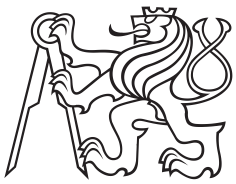
Master's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Electromagnetic Field

Infrastructure of Indoor VLP System

Bc. Martin Suda

Supervisors: Ing. Stanislav Vítek, Ph.D., Jenq-Shiou Leu, Ph.D.
November 2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Suda Martin** Personal ID number: **474251**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Electromagnetic Field**

Study program: **Electronics and Communications**

Specialisation: **Radio Communications and Systems**

II. Master’s thesis details

Master’s thesis title in English:

Infrastructure of Indoor VLP System

Master’s thesis title in Czech:

Infrastruktura indoor VLP systému

Guidelines:

Design and implement the infrastructure for indoor localization using a VLP system. Follow these guidelines:
1. Design and implement an application to configure a network of VLP transmitters over LPWAN.
2. Study the fingerprinting methods used in VLP systems.
3. Design and implement driving firmware for VLP transmitters.
4. Design and implement an algorithm to identify VLP transmitters.

Bibliography / sources:

[1] GUAN, Weipeng, et al. High-accuracy robot indoor localization scheme based on robot operating system using visible light positioning. IEEE Photonics Journal, 2020, 12.2: 1-16.
[2] HUANG, Nuo, et al. Design and demonstration of robust visible light positioning based on received signal strength. Journal of Lightwave Technology, 2020, 38.20: 5695-5707.
[3] RÁTOSI, Márk; SIMON, Gyula. Robust VLC beacon identification for indoor camera-based localization systems. Sensors, 2020, 20.9: 2522.

Name and workplace of master’s thesis supervisor:

doc. Ing. Stanislav Vítek, Ph.D. Department of Radioelectronics FEE

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

Date of master’s thesis assignment: **07.02.2022** Deadline for master's thesis submission: _____

Assignment valid until: **30.09.2023**

doc. Ing. Stanislav Vítek, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. 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

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors, professor Jenq-Shiou Leu, Ph.D and doc. Ing. Stanislav Vítek, Ph.D. for their continuous support during the preparation of my Master thesis, for their enthusiasm, motivation, immense knowledge, and their time they have devoted to me.

Further, I would like to thank my family for their patient support throughout my study.

Finally, I would like to thank my colleague Štěpán Bosák for an outstanding team collaboration on this project.

Declaration

I declare that I have written this thesis by myself and that I have listed all information sources in accordance with Methodological Guidelines on Compliance with Ethical Principles.

In Prague, **20.11.2022**

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, **20.11.2022**

Abstract

The increasing demand for indoor positioning systems and the poor accuracy of Global Positioning System (GPS) in enclosed premises has necessitated other viable solutions. A vastly exploited technique in this field is Visible Light Positioning (VLP), due to its promising properties that allow high accuracy and cost-effective solution compared to other techniques.

The academic research proposes various VLP-based indoor navigation algorithms that tackle issues coming with Visible Light Positioning. Unfortunately, no algorithm has solved the issues well enough to be massively adopted on the market yet.

Recent studies propose unique localization algorithms utilizing different methods. Despite their differences, we noticed one common characteristic shared by all. The papers always assemble an infrastructure prototype to test their approach. This procedure seems rather unnecessary since the core infrastructure is the same and could be shared. We believe a shared assembly would enable the algorithm's rapid prototyping and eventually intensify the team's focus on the main goal, the localization itself.

Therefore, the thesis proposes a VLP system that solves this issue. The system allows the development of localization algorithms with different system settings, thus serving the developer's needs.

Firstly, the thesis presents the system infrastructure and discusses its elements. Secondly, it reviews standard fingerprinting techniques and selects Pulse Width Modulation (PWM) and sinusoidal modulation to demonstrate the functionality. Custom firmware is proposed to drive VLP transmitters with a fingerprinting method. Thirdly, the thesis proposes an application that configures the whole infrastructure and delivers a user-friendly environment. Lastly, it proposes a detection algorithm to localize VLP transmitters within a single-frame image utilizing a mobile robot with a CMOS camera.

Keywords: Visible Light Positioning (VLP), VLC fingerprinting, LoRaWAN®, MQTT, Circle Hough Transfrom, edge detection, web applications

Supervisors: Ing. Stanislav Vítek, Ph.D., Jenq-Shiou Leu, Ph.D.

Abstrakt

Zvyšující se poptávka po polohovacích systémech v uzavřených prostorech a nízká přesnost tradičně využívaného Global Positioning Systemu (GPS) vyžaduje nová, lepší řešení. Díky slibným vlastnostem se nabízí široce zkoumaná metoda Visible Light Positioning (VLP). Její výhodou, ve srovnání s ostatními metodami, je velká přesnost a nižší cena instalace.

Akademický výzkum představil mnoho různých VLP algoritmů ve snaze řešit problémy spojené s Visible Light Positioning. Zatím však žádný z nich nenabízí natolik kvalitní řešení, aby se dal systém masivně využít na trhu.

Publikované práce představují algoritmy založené na velmi rozdílných přístupech. Všimli jsme si, že navzdory jejich značným rozdílům obsahuje vědecká literatura jednu společnou vlastnost. Práce vždy sestavují svůj prototyp infrastruktury, aby mohly otestovat vlastní návrh. Tento postup se zdá poněkud nadbytečný, jelikož hlavní část infrastruktury je vždy stejná a mohla by být sdílena. Věříme, že sdílená infrastruktura by umožnila rychlé prototypování algoritmů, a tak vedla k intenzivnějšímu zaměření týmu na hlavní cíl, kterým je samotná lokalizace.

Tato práce proto implementuje VLP systém, který problematiku jednoho společného rozhraní řeší. Navržený systém umožňuje vývoj lokalizačních algoritmů s různým nastavením systému a slouží tak potřebám vývojáře.

První část diplomové práce představuje infrastrukturu systému a rozebírá její jednotlivé prvky. Druhá část práce představuje rešerši běžných technik fingerprintingu, kdy vybraná PWM a sinusová modulace demonstruje funkčnost navrženého systému. Dále je také navržen firmware pro ovládání VLP vysílačů využívající fingerprinting. V třetí části práce je představena webová aplikace, která systém konfiguruje a přináší uživatelsky přívětivé prostředí. Nakonec práce implementuje detekční algoritmus, který lokalizuje VLP vysílače na snímku z CMOS kamery.

Klíčová slova: Visible Light Positioning (VLP), VLC fingerprinting, LoRaWAN®, MQTT, Circle Hough Transfrom, edge detection, webové aplikace

Překlad názvu: Infrastruktura indoor VLP systému

Contents

Figures

Tables

List of Listings

List of Algorithms

Chapter 1
Introduction

Indoor navigation is an emerging field with many possible applications. Despite the thorough research conducted in the past two decades, no system has been widely adopted yet.

The first notable technology is Global Positioning System (GPS). GPS has been well established for outdoor navigation. Unfortunately, its performance indoors is very limited due to the significant power attenuation caused by passing through the walls in the buildings. Therefore researchers developed solutions based on other technologies. Visible Light Positioning (VLP) is one of the possible solutions. As the name implies, VLP systems localize objects based on transmitted visible light signals. This approach has several advantages compared to RF-based (RF-) positioning. Firstly, it can employ the existing infrastructure of LED luminaries in the buildings, which can considerably lower the cost of the system deployment. Secondly, VLC is immune to Electromagnetic Interference (EMI), and itself does not produce any electromagnetic pollution. Therefore, VLC can be used in environments prone to EMI, such as hospitals. Thirdly, VLP systems can gain a high localization accuracy because they are less sensitive to the multipath effect, and their propagation is more predictable in contrast to RF. Lastly, a VLP system can simultaneously employ positioning and still retain conventional lighting.[6]

In the past few years, researchers proposed numerous VLP systems [2][7][8][9][10] introducing unique solutions to the positioning problem. They utilize different fingerprinting and modulation techniques and develop various architectures and detection algorithms. Despite many differences, we noticed a common characteristic among them. Each solution builds its infrastructure from scratch, even though the core is the same and could be shared.

Therefore, we present a core VLP infrastructure to enhance and simplify development.

1.1 Motivation

Our motivation is to develop a robust and user-friendly VLP infrastructure that would allow testing of various VLC-based indoor navigation algorithms or techniques. Our architecture enables a simple system configuration and operation. Furthermore, it can simplify and speed up development process as the team could entirely focus on positioning techniques and proper VLC instead of lengthy environment designing.

1.2 Contribution

In this thesis, we enhance our previous work [11][12] and propose a VLP infrastructure utilizing modern communication protocols to tackle the problem outlined above. Our

infrastructure comprises four main elements; Nodes, Gateway, System Application, and Robot. These elements are also the main contribution of the project.

This thesis proposes the node firmware that operates the Nucleo B-L072Z-LRWAN1 board equipped with a custom light driver designed by my colleague [13]. Furthermore, the work proposes a system application powered by a Flask back end written in Python, and the front end is designed as a single page application. Finally, we developed a node detection algorithm that localizes nodes within an image captured by the robot [13].

1.3 Organization

Chapter 2 introduces the system infrastructure and presents individual system elements. Furthermore, it connects our previous work [11][12] to this thesis. Chapter 3 conducts a review of VLP fingerprinting techniques. Section 3.3 presents the conclusion of the review, and proposes our aim regarding modulation techniques. Chapter 4 proposes the custom node firmware that employs the fingerprinting techniques and drives the VLC transmitter based on the system configuration received through LoRaWAN®.

Chapter 5 presents the entire system application development. It discusses the architectural types and concludes our choice. Furthermore, it presents our implementation and summarizes the application's functionality. Chapter 6 proposes a node detection algorithm evaluated in Section 6.3.4. The chapter introduces circle object detection techniques, especially Circle Hough Transform. Moreover, it includes a thorough theory introduction to edge detection since it is a crucial requirement for sufficient Circle Hough Transform-based detection employed in our detection algorithm. Finally, Chapter 7 presents the thesis conclusion.

Chapter 2

System Infrastructure

This chapter presents a high-level overview of the project's infrastructure. It introduces the system and functionality of each component. The proposed infrastructure builds on our previous work [11][12], which carries out a node prototype, a LoRaWAN® gateway, and a system application. In [11], we presented a node firmware, the gateway architecture, and a simple system application. The above architecture allows only a single-device configuration and does not support light control. In [12], we introduced a light driver shield that controls the light source. The driver was redesigned and significantly improved to the current status in [13].

This thesis proposes a new enhanced infrastructure supporting multi-node configuration and node detection. Furthermore, it presents its new software implementation. The system comprises four elements: Nodes, Gateway, System Application, and Robot. The thesis introduces and designs the first three elements. This chapter briefly discusses the objective of the last element, but the robot's design is presented in [13]. Figure 2.1 illustrates the new infrastructure, and each element is discussed separately below.

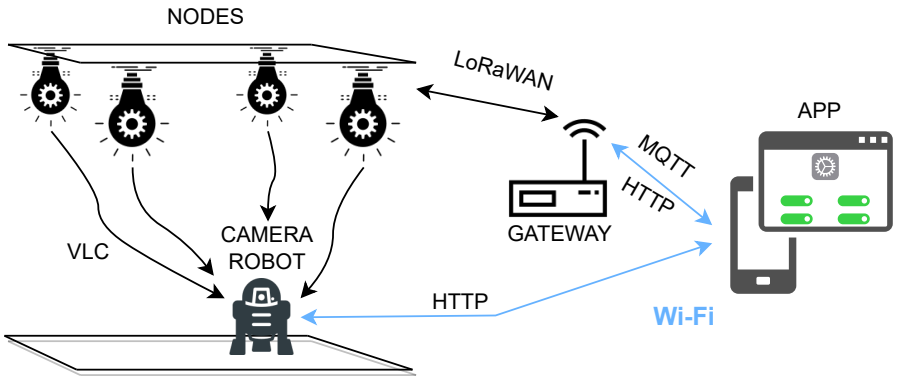


Figure 2.1 The new enhanced system infrastructure.

2.1 Nodes

This element contains three identical VLC transmitters (nodes). Each node comprises a microcontroller (MCU) equipped with a custom light driver shield [13] and a light source. The node's objective is to receive configuration messages from the user and reconfigure itself based on the message content. The configuration messages consist of a modulation type and unique parameters for each node. Thus, nodes are distinguishable and could be considered a VLP source.

The new node supports multiple-node structures and implements LoRaWAN® class C mode operation. The class C operation carries out significantly lower latency of system configuration, which is a beneficial property for the system efficiency and operation workflow — more details about LoRa®/LoRaWAN® protocol in [11].

In order to achieve the above, we had to upgrade the NUCLEO-F446RE board equipped with the I-NUCLEO-LRWAN1 shield to Nucleo B-L072Z-LRWAN1. The new board naturally supports LoRa® communication and delivers higher performance and memory. On the contrary, we had to redesign the entire firmware to accomplish our goals. Chapter 4 presents the new firmware. Finally, the Nucleo B-L072Z-LRWAN1 main board was equipped with the custom light driver shield [13] and placed into a 3D-printed cover to store the node hardware safely. Figure 2.2 shows the prototype.

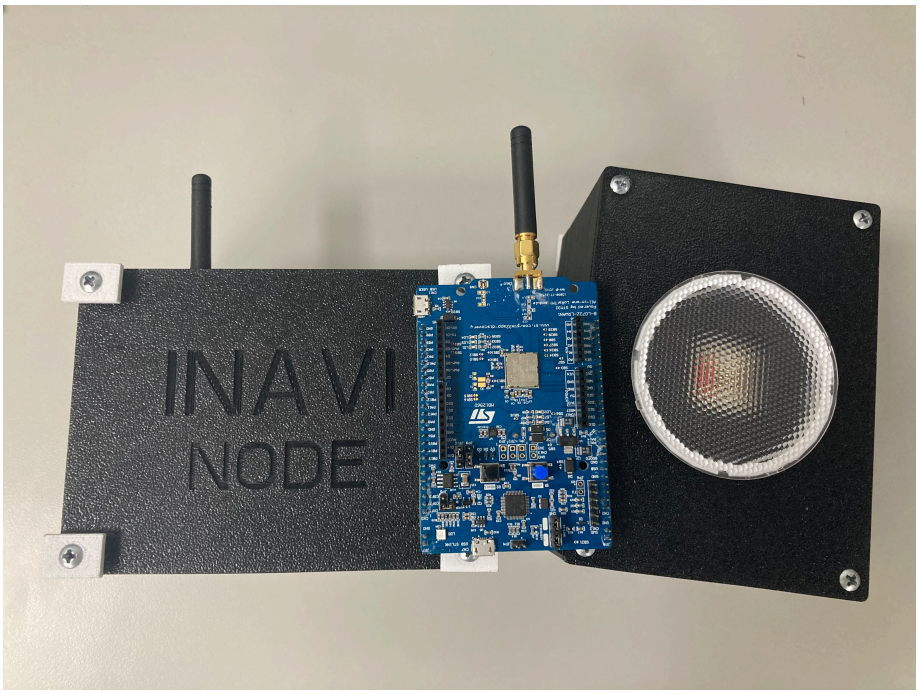


Figure 2.2 Node prototypes and the B-L072Z-LRWAN1 board.

2.2 Gateway

The Gateway element creates a communication bridge between System Application and Nodes. It receives system configuration from the application via MQTT protocol and generates LoRaWAN® messages to distribute the configuration to individual nodes. Furthermore, it processes the LoRaWAN® uplink messages from nodes and delegates them through the MQTT broker. In [11], we present the gateway's architecture.

Figure 2.3 The gateway prototype.

2.3 System Application

System Application is a fundamental structure element. It allows configuring each node separately with a unique configuration or the entire system at once. The application creates a user-friendly interface where the user can easily configure the entire system while primarily focusing on positioning algorithms. Furthermore, the user can manage the existence of nodes in the system.

Finally, the application presents an object detection algorithm localizing the nodes in an image captured by the robot. Moreover, it allows controlling the robot and CMOS camera. Chapter 5 introduces the implementation.

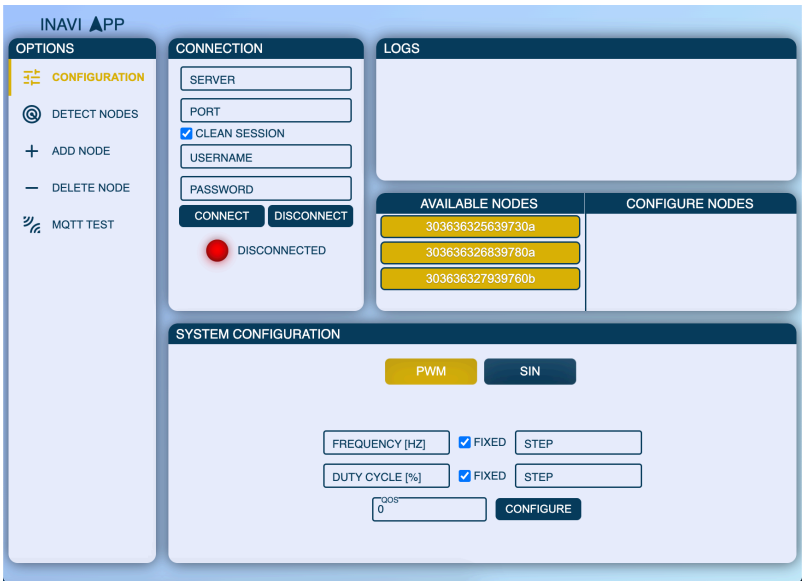


Figure 2.4 The system application.

2.4 Robot

The last system element consists of a customized robot equipped with a CMOS camera. The robot's objective is to travel underneath the nodes and capture images of the environment. The images are sent to the application back end upon a request, where an object detection algorithm is utilized. The detection algorithm determines the node's position in the image and delivers the processed image, including regions of interest delimited by a bounding box. Chapter 6 proposes the algorithm and theory.

Chapter 3

VLP Fingerprinting Techniques

There is a vast number of modulation techniques available. However, only a few are appropriate for VLP systems since they must abide by the VLC requirements to be correctly decoded. These requirements will be further discussed below.

Modulation schemes utilized in VLP are commonly referred to as fingerprinting methods. Their goal is to differentiate system transmitters (nodes) from each other. In other words, the system assigns a unique fingerprint to individual transmitters upon which they are later recognized.

This chapter conducts a review to acquire a broader understanding of the topic. Moreover, it summarizes gathered knowledge and concludes the implementation goals for the Node firmware (Chapter 4).

Our VLP system focuses on a CMOS rolling shutter mechanism at the moment, and thus techniques in the chapter will mainly discuss this approach. Rolling Shutter Effect (RSE) is caused by row-by-row exposure and reading. This effect considerably improves the communication data rate and allows reducing or completely suppressing the LED flickering visible to the naked eye (approximately 100 Hz).[1] A detailed RSE introduction is available in [14][15].

Nowadays, high-resolution CMOS sensors are commonly deployed in ordinary mobile phones. Hence, it makes CMOS-based VLP a promising candidate for massive market adoption.[14] Unfortunately, there are still several challenges to be solved. Firstly, the CMOS sensor is sensitive to pixel saturation (”blooming” effect) which can cause a high error rate. The study [14] shows that the ”blooming” effect can be mitigated by a proper luminaire that uniformly distributes the light intensity. Secondly, the CMOS sensor suffers a read-out time gab during which the sensor is blind and does not receive signals. Therefore, the transmission length must be taken into account and should not surpass the frame length. Lastly, the CMOS sensor suffers from a low extinction ratio limiting the transmission distance. The low extinction ratio increases the interference of background noise.[14] Nevertheless, the advantages still prevail, and the approach deserves attention.

3.1 Related work

Before moving on to the review, the VLC techniques have the following requirements. Firstly, the average light brightness must remain constant regardless of the data content. Otherwise, flickering will be introduced. Secondly, CMOS-based methods cannot continuously receive data due to the time gap between the bottom and top line exposure. Hence, the modulation scheme needs to account for the length of the transmitted signal to be shorter than the frame length.[1] Lastly, the fingerprinting scheme should employ a multiplexing method that guarantees robust distinction between transmitters. The following sections introduce fingerprinting schemes commonly used in VLP.

3.1.1 On-Off Keying Modulation

On-Off Keying (OOK) is a widely adopted modulation technique utilized in many CMOS-based VLP systems. OOK is a simple modulation that encodes binary data by switching the light source (1 → ON, 0 → OFF). When OOK modulation illuminates the CMOS sensor, the data creates bright and dark stripes proportional to its data rate.

Unfortunately, long series of the same binary value break the constant brightness requirement and cause flickering. Therefore, the OOK modulation is commonly enhanced with an additional encoding scheme to prevent this behavior. The authors in [1] present several encoding solutions; They propose Four-Pulse Position Modulation (4PPM) to safely encode 2-bit code words and Variable-Pulse Position Modulation (VPPM) to encode a 1-bit words. The 4PPM modulation encodes each word in four time slots by shifting binary "0" across the slots. For instance, the code word "00" and "01" would be encoded as "0111" and "1011", respectively. Similarly, VPPM encodes 1-bit by shifting an off-state between first to last position (0 → 0111, 1 → 1110). Figure 3.1 illustrates both encoding. The cost paid for maintaining constant brightness this way is that the 4PPM's data rate decreases by 1/2 and VPPM's by 1/4. Besides above mentioned, studies often employ Manchester code or bar-coding to acquire required behavior.

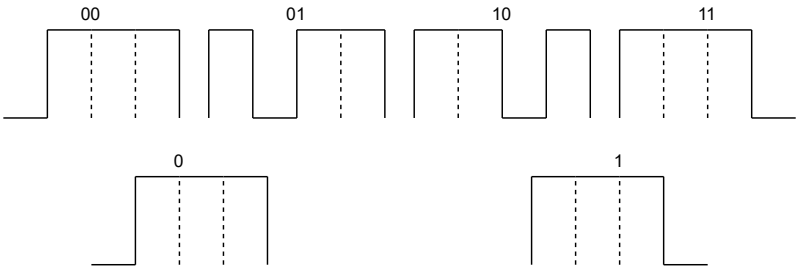


Figure 3.1 Constant brightness techniques presented in [1], 4PPM illustrated on the top and VPPM on the bottom.

References [8][16] utilize the Manchester encoded identifiers. The authors in [8], introduce Manchester code to increase the channel capacity and allow constant brightness. Moreover, they combine pure tone (PWM), and Manchester encoded data in a hybrid design which mitigates the distance dependence. It takes advantage of data transmission for transmitters close to the receiver and retain PWM frequency decoding of transmitters far away and thus decreasing the system's distance dependence.

Reference [16] presents an another Manchester code approach utilizing frequency multiplexing. The proposed simulation avoids harmonic spectral overlaps by using odd frequency carriers (47 kHz, 59 kHz, 83 kHz, and 101 kHz).

On the other hand, the authors in [2] exploit OOK with bar-code encoding. Similar to those above, the method assigns a unique identifier to each transmitter. Identifiers are modulated by the OOK and then further encoded by ITF bar-code mechanism to obtain constant brightness and dimming functionality. The ITF bar-coding was implemented as follows. Firstly, the four-digit identifier was encoded with OOK, where 5-bits represented each digit. Secondly, binary one/zero was assigned to a specific fringe width to represent the value. Lastly, the first/third (resp. second/fourth) digit of the identifier was encoded by five dark (resp. bright) fringes. This approach overpowers Manchester code in the

efficiency (bit per symbol). It carries out an efficiency of 0.66439 compared with 0.5 presented by the Manchester code. Figure 3.2 illustrates the encoding mechanism.

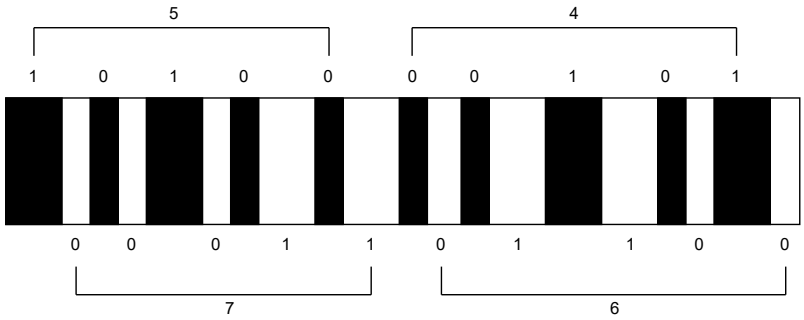


Figure 3.2 The bar-code encoding of 5746 proposed in [2].

The system presented in [7] proposes a different multiplexing approach. It employs the OOK modulation with Time Division Multiplex (TDM). The VLP sources transmit data periodically in non-overlapping times slots. Therefore, the transmitters need synchronization, unlike the previous techniques. Lastly, the code words are encoded with a constant average probability to obtain constant brightness.

Finally, *Jianli Jin et al.* in [17] emphasize the importance of the multiplexing techniques when they propose two methods based on Code Division Multiplexing (CDM). They state that CDM is preferable in VLP systems to FDM or TDM because of its superiority in resistance to asynchronous interference and low complexity. They employ the Walsh code and propose a Novel Signature code to demonstrate their thesis.

3.1.2 Pulse-Width Modulation

PWM is another binary pulse modulation that is commonly used in CMOS-based VLP. As the OOK, PWM leaves bright/dark stripes on a CMOS sensor. The frequency affects the number of stripes, and the duty cycle controls the stripe width. A significant advantage of PWM is that it naturally satisfies the brightness condition if the duty cycle remains constant. Moreover, the duty cycle allows dimming functionality, commonly used lighting property. On the other hand, PWM's data transmission is less efficient than OOK's, and thus the channel capacity is narrower.

In [8], [15], and [3], they introduce a PWM approach with frequency multiplexing. Each transmitter acquires a unique frequency/duty cycle identifier upon which it is identified. Authors in [15] and [3] utilize an additional feature when they measure the area of the LED projection on the CMOS sensor. This feature is used to estimate the distance of the transmitter. Figure 3.3 illustrates PWM modulated transmitters captured by a CMOS camera.

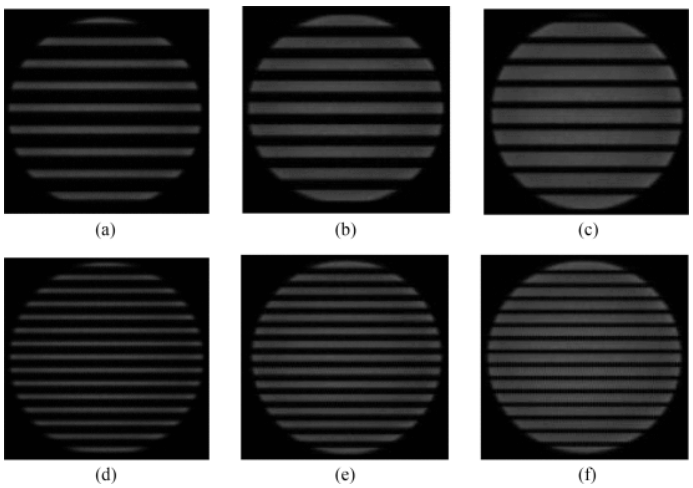


Figure 3.3 A PWM modulated transmitter with various frequencies and duty cycles.[3]

3.1.3 Sinusoidal Modulation

Recent studies show an alternative approach to pulse modulation by exploiting sinusoidal signals and photodiode receivers.[18]

References [10] and [19] present a method utilizing sinusoidal frequency multiplexing and Received Signal Strength (RSS). The frequency works as a unique identifier, and RSS determines the distance between the receiver and transmitter. A similar but reversed approach is introduced in [9]. They placed receivers, constructed by photodetectors, on the ceiling and VLC transmitters were mobile on the floor. This work focuses on multi-target localization, and thus the system contains multiple receivers, unlike ours. The detection is similar to the previous studies. They utilize band-pass filters that separate individual sinusoidal fundamentals, and RSS estimates the distance.

Authors in [20] propose a Time Difference of Arrival (TDOA) based method. The method assigns one transmitter as a reference node, and the node obtains a unique frequency. The TDOA of the remaining nodes is calculated concerning the reference node. To avoid interference, they modulated the remaining nodes with odd multiples of the reference frequency. A similar approach is presented in [21].

Lastly, studies [22] and [23] propose an interesting approach utilizing Ambient Light Sensor (ALS) embedded in a mobile phone. The studies propose a sinusoidal frequency multiplexing where each transmitter obtains a unique sinusoidal signal as in previous cases. The signals are then received and evaluated in the ALS instead of the external photodiode. Since the ALS behaves as a frequency filter and performs analog to digital conversion, the studies can acquire the fundamental transmitter frequency by applying Discrete Fourier Transform. With the fundamental frequencies, they can easily distinguish individual transmitters.

3.2 Discussion

This discussion summarizes the above review and draws conclusions about individual techniques which will determine the aims of our implementation in the next section. In the discussion, we primarily focus on VLC compliance, sufficient channel capacity regarding the number of nodes, and implementation costs. We have concluded the following.

The OOK modulation is the most common CMOS-based modulation scheme. However, On-Off Keying tends to suffer from flickering due to payload transmission (inconsistent switching). This issue is commonly solved by combining OOK with an additional encoding method that delivers constant brightness. Following methods were introduced: Manchester code, 4PPM, VPPM, and bar-coding. These additional encoding methods add implementation and processing costs. On the other hand, OOK allows the transmission of unique identifiers containing node location. However, the reference [8] states that the data are hard to decode farther away from the transmitter.

The second pulse modulation, PWM, holds great VLC properties. It naturally delivers a constant brightness for any duty cycle. Hence, it does not need an additional encoding like OOK. Furthermore, PWM supports dimming functionality, a widely used feature in conventional lighting. On the other hand, PWM delivers worse data transmission properties than OOK and generally suffers lower channel capacity. We can conclude from the above properties that PWM is a sufficient OOK substitution for a small number of transmitters whose architecture does not necessitate data transmission.

Lastly, the sinusoidal modulation recently has drawn a great deal of attention and delivers a promising alternative by utilizing an embedded ALS in mobile phones.

3.3 Conclusion

In conclusion, we decided that our system will implement one of the above pulse modulations to prove the project's concept. The decision has been made upon the previously mentioned requirements.

Firstly, let us assume the channel capacity. Since our test environment consists of only three transmitters, we do not require any data transmission at the moment. Secondly, we should consider VLC compliance and implementation costs. PWM with a constant duty cycle is naturally VLC compliant. On the other hand, OOK requires some additional encoding, which adds extra implementation costs. Furthermore, the study [8] shows considerable distance dependence of OOK's payload decoding. To robustly identify nodes farther away from the receiver and nearby nodes requires a hybrid modulation alternating between OOK and PWM. They show that the nodes in close proximity can take advantage of data transmission, but nodes farther away need a pure signal of PWM to be correctly identified. Therefore, we believe the OOK modulation would eventually have to be combined with PWM eventually. Our clear choice is to implement the PWM modulation before continuing to the more complex schemes such as OOK.

In addition, we decided to implement sinusoidal modulation. Even though the system focuses on CMOS-based techniques, our vision is to create an environment to test various indoor-navigation algorithms regardless of the receiver type. Furthermore, we aim to utilize sensor technologies currently available in mobile phones. The studies [22], and [23] show

sinusoidal modulation meets these requirements.

Chapter 4

Node Firmware

This chapter proposes a custom firmware developed for the system's node. It introduces firmware requirements, its architecture, and implementation. Furthermore, it presents end-to-end test results to confirm the correct functionality.

4.1 Functional requirements

The proposed implementation should fully support the LoRaWAN® communication protocol and satisfy the following;

The node should be able to establish a connection with the gateway through a secure link utilizing the OTAA authentication. After the authentication, the node should switch to the class C operation to obtain the maximum downlink flexibility. Furthermore, the firmware should implement the system configuration functionality as follows;

Firstly, the node must be able to parse the configuration message sent from the application and deliver the configuration data in a structured format. Secondly, it must be able to generate VLP signals. As we concluded in the review above, PWM and sinusoidal signals will be the modulation prototypes. Finally, the implementation should automatically configure the node based on the received configuration message and send the generated signal to the appropriate GPIO pin of the light driver shield.

4.2 Architecture

Our architecture combines the robust implementation of the I-CUBE-LRWAN Expansion Package with the custom functionality needed for the project.

This section introduces the architecture of the I-CUBE-LRWAN Expansion Package and the custom architecture developed for driving individual system nodes. The upcoming part introduces the vital architectonic principles important to our integration. The complete package documentation is available in [4].

4.2.1 I-CUBE-LRWAN package

The I-CUBE-LRWAN package provides a robust LoRaWAN® stack implementation, compliant with the LoRaWAN® specification issued by LoRa Alliance®. Furthermore, it provides APIs to drive the necessary MCU hardware.

The package composes of three logical layers. Figure 4.1 illustrates the architecture. The structure's lowest layer is the HAL library. It brings the hardware APIs and initialization. The second layer comprises two parts. Firstly, it introduces a Sensor driver, which enables the communication between sensors and the Application layer. The driver

approaches sensors through the I2C bus utilizing the HAL API. Secondly, it presents a part called Middleware, responsible for the LoRa® communication and utilities. Finally, the Application layer carries out the high-level logic and initializes the necessary hardware (ADC, UART).

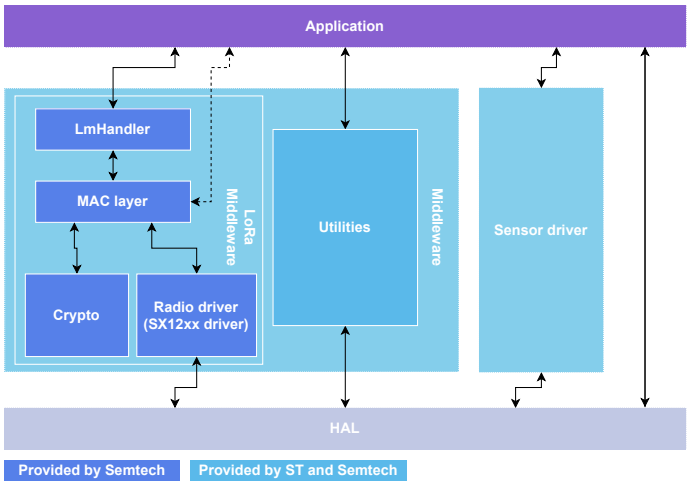


Figure 4.1 The I-CUBE-LRWAN architecture.[4]

LoRa® Middleware

As Figure 4.1 shows, LoRa® Middleware consists of several sub-parts. Firstly, the radio driver utilizes the GPIO and SPI HAL library to control the radio functionality. Moreover, it implements an API for higher-level software.

Secondly, the MAC layer implements the LoRaWAN® functionality. It controls the physical layer through an API, manages the timed tasks, and calculates the transmission time-on-air. In addition, it monitors the transmission duty cycle to prevent exceeding the limitations mandated by the LoRa® protocol. Finally, it ciphers the payload and header utilizing the Crypto library. The Crypto library implements the AES encryption/ decryption algorithm.

Lastly, the architecture presents LmHandler on top of the MAC layer. LmHandler sets an API to access the LoRaWAN® MAC services without worrying about the LoRaWAN® state machine. LmHandler can be bypassed in more complex applications to access the MAC layer directly. This way, the user can obtain the full extent of the MAC layer instead of limited functionality delivered through the LmHandler API.

Utilities

Middleware's Utilities provide necessary APIs; the primary role will play Sequencer, System Time, Time Server, and Trace. Utilities also implement Low-Power management, but due to space limitations, this text will not further discuss its API.

Sequencer provides a robust task manager that prevents the race condition. In other words, it protects shared data from being changed by multiple tasks at once, which would bring undesirable results. Moreover, it implements a task triggering functionality upon

an event occurrence represented by an interrupt. It is worth mentioning that Sequencer is not an OS and uses only a single-memory stack. Unlike in RTOS, every Sequencer task is run to completion and cannot switch to another task. The only way to simulate the functionality above is to call `UTIL_SEQ_WaitEvt()` within the task being executed. Finally, Sequencer could be interpreted as advanced while-loop centralizing tasks and event bitmap identifiers.[4] Listing 4.1 introduces an example of the Sequencer application.

Listing 4.1 An example of the Sequencer application.

```
#include "stm32_seq.h"

#define flag 0      // default value for unused parameter
#define priority 0          // priority definition
#define SEQ_DEFAULT (~0U)   // sequencer initial value

// Register Task
/* taskId_bm1 and taskId_bm2 are bitmasks => task identifiers*/
UTIL_SEQ_RegTask(taskId_bm1, flag, func1);
UTIL_SEQ_RegTask(taskId_bm2, flag, func2);

// Run Sequencer
While(1) {
    UTIL_SEQ_Run(SEQ_DEFAULT);
}

// Define idle state
void UTIL_SEQ_Idle( void ) {
    /* Action while sequencer idle */
}

// Trigger function definition
void func1Callback(void) {
    // set task to execute func1
    UTIL_SEQ_SetTask(taskId_bm1, priority);
}
```

Time Server delivers a reliable clock source for the application and the LoRaWAN® stack. The timer object utilizes the RTC and enables timed-task execution. Furthermore, this architecture allows a timer to count even when in low-power mode. Time Server can trigger timed tasks and create timers necessary for the application. The number of requested timers is not limited. The server has to be initialized by calling `UTIL_TIMER_Init()`.[4] Listing 4.2 presents an example of a timer handling utilizing the Time Server.

Listing 4.2 An example of Timer Server API.

```
#include "stm32_timer.h"

// Define helper variables
unit32_t period = 0xFFFFFFFFU;
UTIL_TIMER_Mode_t mode = UTIL_TIMER_PERIODIC;

// Create a Timer object
static UTIL_TIMER_Object_t timer;

// Create a new timer instance in Time Server
/*
mode options: UTIL_TIMER_PERIODIC, UTIL_TIMER_ONESHOT,
```

```
NULL => callback has no arguments
*/
UTIL_TIMER_Create(&timer, period, mode, callbackFunc, NULL);

// Start timer
UTIL_TIMER_Start(&timer);
// Stop timer
UTIL_TIMER_Stop(&timer);
```

System Time delivers an API that allows recording MCU time based on UNIX epochs. Our application references the time to the last MCU restart, and it uses the API to retrieve the timestamp in the tracing messages. The API implements the following functions; `SysTimeSet`, `SysTimeGet`, `SysTimeMkTime`, and `SysTimeLocalTime`. The first function initializes the system time recording. The `SysTimeGet` function retrieves the recorded time, and lastly, `SysTimeMkTime` (resp. `SysTimeLocalTime`) converts local time into UNIX epoch (resp. UNIX epoch into local time).[4]

Finally, the Trace API provides functions to handle application messages and prints them to the serial port interface. The implementation must call `UTIL_ADV_TRACE_Init()` to initialize the hardware upon which a complete DMA transmission sets UART in DMA mode with a registered callback.[4] Listing 4.3 provides an example of tracing.

Listing 4.3 An example of tracing.

```
#include "stm32_adv_trace.h"

#define TS 0  // timestamp omitted
#define VT 2  // verbosity level
#define REG 0 // region of the trace omitted

const char* s = "Message to be printed to the serial interface.";

// Send the message to the trace
UTIL_ADV_TRACE_COND_FSend(VT, REG, TS, s);
```

4.2.2 Custom Firmware

Figure 4.2 presents our architecture. The structure consists of two layers. The lower layer comprises previously discussed I-CUBE-LRWAN extensions that provide a robust hardware interface and LoRaWAN® stack. The higher layer, called Application, integrates the project's functionality. The Application layer consists of Core and App libraries.

The Core library handles hardware initialization and setup. It sets the UART in a DMA mode interface for tracing, ADC interface for digitizing the sensor data, and RTC interface to handle the timers. Moreover, it provides system initialization and the main program loop. The App library includes configuration files that provide the system initialization and callback functions. Furthermore, the library holds the Indoor_navigation directory that handles the node configuration logic discussed thoroughly in Section 2.1. To clarify the purpose of each segment in the architecture, let us assume the following scenario described below.

The node initializes itself by running the main program. When initialized, the node sends an OTAA JOIN request by utilizing the callback functions and LoRaWAN® stack. It successfully joins the network. An App configuration file creates a timer within the system

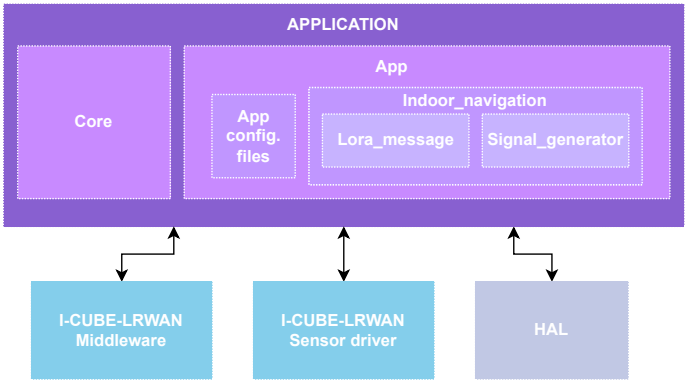


Figure 4.2 The firmware architecture.

initialization and triggers an uplink Alive message containing sensor data. A callback function retrieves the sensor data by utilizing the Core library and sends the message. After a few minutes, the node receives a downlink LoRa® message containing the system configuration data in the payload. The LoRaWAN® stack retrieves the payload data from the message and provides them to the Indoor_navigation library, which takes over the data control. The Lora_message module in the Indoor_navigation library transforms the payload data into a structure. The Signal_generator module takes the structured data and carries out an automatic node re-configuration. In other words, the node changes the modulation signal according to the configuration data processed by the Indoor_navigation library. Besides the automatic configuration, the Signal_generator module provides the hardware initialization and setup. The following section presents further implementation details.

4.3 Implementation

This section introduces the implementation details of the fundamental firmware components. The following components are discussed:

- Main Program
- Configuration files
- Indoor Navigation library
  - Lora Message handler
  - Signal Generator

We implemented our firmware in language C utilizing the stm32CubeIDE environment. stm32CubeIDE is an Integrated Development Environment (IDE) built on the Eclipse framework and provides a user-friendly interface for stm32 development. The IDE manages the source code as a stm32CubeIDE project. Thus, it contains XML configuration files that keep track of source code paths. We maintained our implementation by exploiting the git version control framework and the university GitLab cloud. Finally, the GCC compiler handled the C-code compilation.

4.3.1 Main program

The main.c file is a crucial program that implements the main infinite while loop. It initializes necessary components; the HAL library, the system clock, the LoRaWAN® stack, and Signal Generator. Moreover, it provides the default light configuration and calls the Sequencer process. A simplified example of the main.c is illustrated in Listing 4.4.

Listing 4.4 A simplified example of main.c file.

```c
int main(void) {

    // Init system
    HAL_Init();
    SystemClock_Config();
    MX_LoRaWan_Init();
    SignalGenerator_Init();

    // Set default configuration to sinus with f=100Hz a=75%
    sin_t config = {.frequency=100, .amplitude=75}
    LghtConfigData_t data = {.modulation=2, .sinParams=config};
    set_light_configuration(data);

    // Register a Sequencer Task
    UTIL_SEQ_SetTask((1 << defaultConfig_bitmask_id), priority_0);

    // Run Sequencer
    while(1) {
        UTIL_SEQ_Run(UTIL_SEQ_DEFAULT);
    }
}
```

4.3.2 Configuration files

Several configuration files are necessary to set the firmware environment. The lora_app.c file defines the LoRaWAN® callbacks and connects the firmware's Application layer with LmHandler. Listing 4.5 presents an example of a callback definition handling the node configuration after the received message. The second file worth noting is lora_info.c, which provides a structure containing LoRaWAN® configuration. It holds the activation mode, active region, and key-management type. Lastly, the app_lorawan.c file implements helper functions that deliver initialization at the firmware start-up.

Listing 4.5 A simplified example of `OnRxData()` callback.

```c
static void OnRxData(LmHandlerAppData_t *appData,
                     LmHandlerRxParams_t *params) {

    if ((appData != NULL) && (params != NULL)) {
        /* perform further data validation */
        /* perform actions on special messages received
            (e.g., CLASS change, LED effects) */

        // Configure the node
        /* reset configuration data structure */
        pwm_t pwmConfing = {0,0};
        sin_t sinConfing = {0,0};
```

```
13          LghtConfigData_t data = {0, pwmConfig, sinConfing};
14
15          // Parse received data
16          parse_rxData(appData->Buffer, appData->BufferSize, &data);
17
18          // Set the signal generator
19          set_light_configuration(data);
20
21          // Register task to Sequencer
22          switch (data.modulation) {
23              case 1:
24                  UTIL_SEQ_SetTask(1 << pwm_bitmap_id), priority_0);
25                  break;
26              case 2:
27                  UTIL_SEQ_SetTask(1 << sin_bitmap_id), priority_0);
28                  break;
29              default:
30                  APP_LOG(TS, VL, "UNKNOWNMOD: Sig. generator idle");
31          }
32
33          // Reset appData buffer for next Rx
34          memset(appData->Buffer, 0, appData->BufferSize);
35          appData->BufferSize = 0;
36          appData->Port = 0;
37      }
38
39  }
```

4.3.3 Indoor Navigation library

The Indoor Navigation library is part of the App directory. The library implements the Lora Message handler (Lora_message) and the Signal Generator component (Signal_generator). The library source code is provided in the Indoor_navigation directory. This section introduces the main design features of the library.

Lora Message handler

The Lora Message handler contains parsing functions that handle the raw payload received from the LoRaWAN® stack. Listing 4.6 shows a simplified example of the `parse_rxData` function. The function takes three parameters; The first represents the pointer to the received data. The second holds the data buffer size, and the last carries the pointer of the node configuration structure used by the Signal Generator.

Listing 4.6 A simplified example of `parse_rxData()`.

```
1  void parse_rxData(uint8_t* rxData, uint8_t rxDataSize, LghtConfigData_t*
       light_config_data) {
2
3      // Retreive modulation type
4      light_config_data->modulation = rxData[0];
5
6      switch(light_config_data->modulation) {
7          case 1: // pwm
8              light_config_data->pwmParams.frequency=
```

```
                    ((rxData[1] << 24) | (rxData[2] << 16)|
                    (rxData[3] << 8) | rxData[4]);

            light_config_data->pwmParams.dutyCycle = rxData[5];
        case 2:  // sin
            /* equivalent parsing logic as presented above */
        default:
            APP_LOG(TS,VL, "UNKNOWNMOD: Parser Failed!")
    }
}
```

Signal Generator

The Signal Generator is essential for the node operation while it implements the automatic node configuration and controls the necessary hardware. The Signal_generator directory comprises hardware and generator handlers, as shown in Figure 4.3.

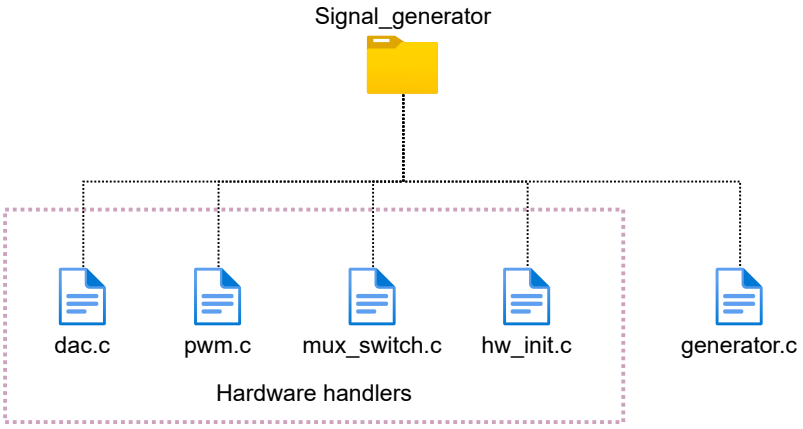


Figure 4.3 The Signal_generator file structure.

The dac.c file provides an initialization function that configures DAC. The DAC output is mapped to PA4(A2)[1] pin while using the `DAC_OUT1` with `DMA_CH2`. The function utilizes memory mapping provided by the HAL library. Besides the initialization, it configures the GPIO to analog mode, sets TIM6 as the DAC trigger with the sampling frequency of 10 kHz, and maps the DMA memory to the address holding the sinusoidal lookup table.

The pwm.c file initializes the `TIM21_CH2` timer in the PWM mode and maps the output pin to PB14(D12). The PWM mode is configured with a UP-direction counter, edge-aligned mode and enables the preloading. Finally, the file implements PB12(D9) bypass to prevent pin damage caused by the incompatible pinout of the light driver shield with the main B-L072Z-LRWAN1 board.

The mux_switch.c file implements multiplex logic. It switches between two outputs on the light driver shield based on the current fingerprinting configuration. The switch is placed on PA9(D8). The logical zero sets the light driver shield to the PWM signal mode, and the logical one switches the mode to a DAC signal.

[1] Appendix B provides the Nucleo B-L072Z-LRWAN1 pinout

The hw_init.c file provides a helper function that enfolds the generator initialization. The `hw_init_light_generator` function helps to maintain the clarity and readability of the code. Table 4.1 summarizes the hardware configuration.

	PIN	ARDUINO PIN	MODE	TIMER	DMA
PWM	PB14	D12	PWM	TIM21_CH2	NONE
DAC	PA4	A2	ANALOG	TIM6	YES
MUX	PA9	D8	OUTPUT	NONE	NONE

Table 4.1 Signal Generator hardware configuration summary.

Finally, the generator.c file provides four crucial functions; The `adjust_PSC_ARR` function adjusts the PSC and ARR registry to prevent registry overflow. In other words, the function finds proper PSC and ARR registry factorization values within its 16-bit (65535) limit and simultaneously satisfies equation 4.1. The equation presents the dependence of the signal frequency on the timer clock frequency, PSC registry, ARR registry, and the number of sample points N_s. The adjustment algorithm loops over the possible PSC values and acquires ARR based on equation 4.2. The registries are set accordingly if both acquired values fit within the 16-bit limit.

$$f_{sig} = \frac{f_{clk}}{(PSC+1)(ARR+1)N_s} \tag{4.1}$$

$$ARR = \frac{f_{sig}}{f_{clk}(PSC+1)N_s} - 1 \tag{4.2}$$

The `set_light_configuration` function handles the modulation choice based on provided data and employs the `switch()` statement to conduct the decision.

The remaining functions `set_sin_modulation` / `set_pwm_modulation` set the required modulation type and configure the hardware accordingly. These handlers are executed based on the `switch` statement evaluation in the `set_light_configuration` function. Listing 4.7 provides a simplified example of `set_sin_modulation`.

Listing 4.7 A simplified example of `set_sin_modulation()`.

```
1
2  void set_sin_modulation(sin_t sin_data) {
3      // Stop DAC using macro
4      DAC_STOP;
5
6      // Adjust the PSC and ARR registry
7      adjust_PSC_ARR(TIM_DAC_MOD, sin_data.freq, SIN_LOOKUP_SIZE);
8
9      // Fill the DMA with appropriately changed lookup table values
10     for(int i = 0; i < SIN_LOOKUP_SIZE; i++) {
11         sin_output[i] = (uint16_t)
12             ((float) sin_data.amp/100 * (float)sin_lookup[i]);
13     }
14 }
```

4.4 Functional testing

We designed multiple tests to verify the firmware's functionality. The testing process had two stages.

The first stage investigated the core signal generation functionality with the following scenario: a dummy signal configuration was hard-coded into the firmware, and we monitored the generated signal on an oscilloscope. The signal configurations were randomly picked and manually added to the firmware to cover possible permutations.

The second stage designed an end-to-end test that confirms the correct node behavior within the infrastructure. The tests had the following scenario: a user sends configuration messages through the network to the tested node utilizing the system application (Chapter 5). The node receives the message and configures itself accordingly. We observed the node through the serial connection output and signal outputs on the oscilloscope during the test execution. The output of these interfaces was analyzed, and we concluded test results. Figures 4.5, 4.6, and 4.4 present an example of collected data. The firmware implementation successfully passed all designed tests within both stages. Besides the core functionality, the test verified a low frequency-offset error below 0.5 %.

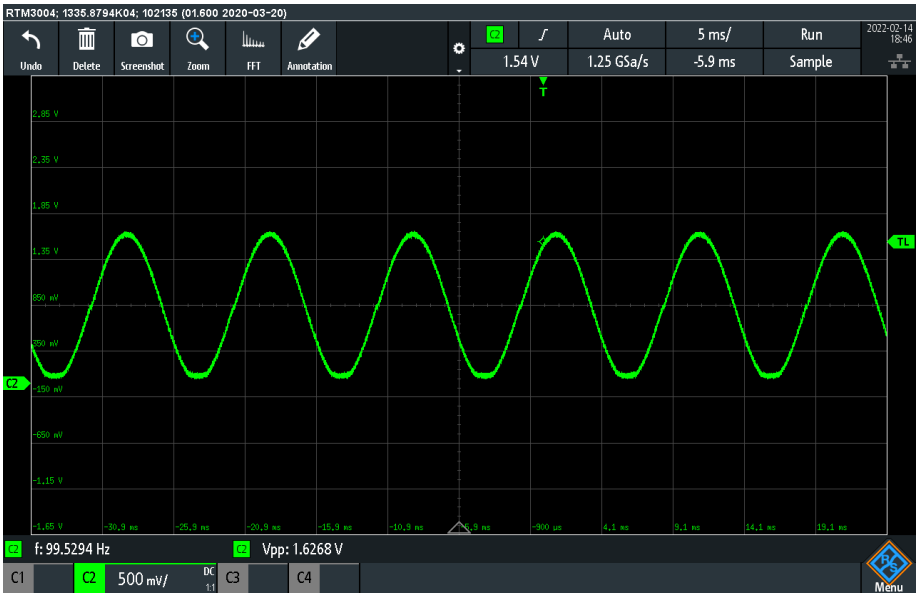
```
LGHT PARSER: Received configuration:
Modulation:     SINUS,
Frequency:      1000 Hz,
Amplitude:      100 %,
Phase:  0 deg
1713s809:LIGHT CONF: Setting light modulation
1713s810:SIN: Setting ...
1713s810:SIN: Parametrs:
Frequency:      1000 Hz,
Amplitude:      100 %,
```

(a)

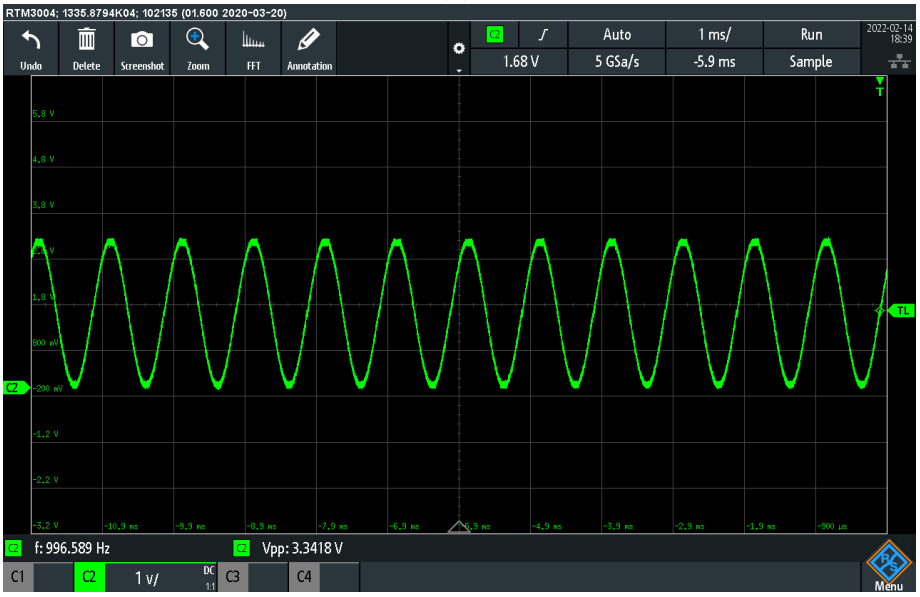
```
LGHT PARSER: Received configuration:
Modulation:     PWM(1),
Frequency:      500 Hz,
DutyCycle:      50
1236s102:LIGHT CONF: Setting light modulation
1236s102:PWM: Setting ...
1236s102:PWM: Parametrs:
Frequency:      500 Hz,
Duty Cycle:     50 %
```

(b)

Figure 4.4 An example of serial line communication under the test execution; a) Sinusoidal modulation, and b) PWM modulation.



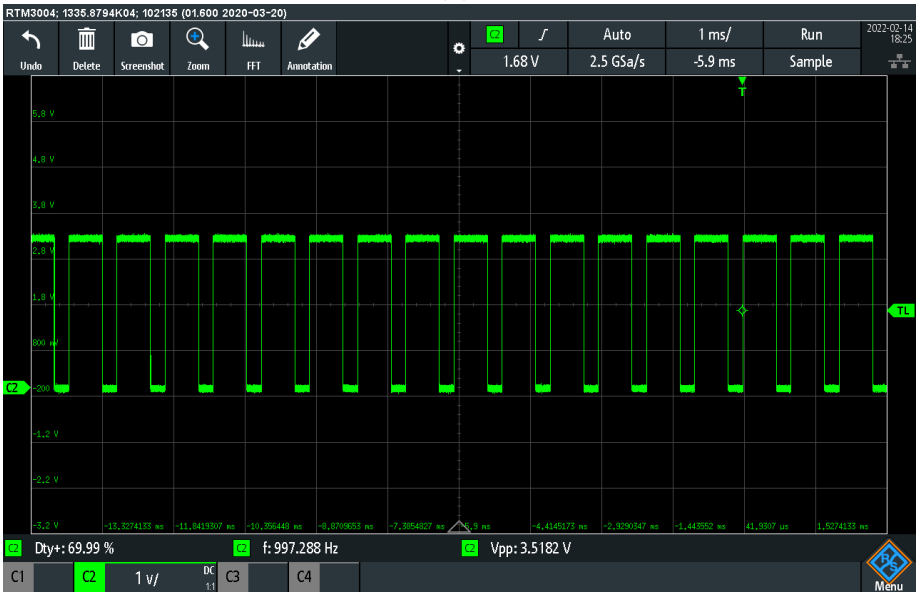
(a)



(b)

Figure 4.5 Sinusoidal signals generated by the firmware; a) 100 Hz with 50% amplitude, and b) 1 kHz with 100% amplitude.

(a)



(b)

Figure 4.6 PWM signals generated by the firmware; a) 500 Hz with 50% duty cycle, and b) 1 kHz with 70% duty cycle.

Chapter 5

System Application

This chapter revises the project's current state and suggests necessary improvements that we ought to implement before the development can continue further. Moreover, the chapter presents the functional and non-functional requirements that the implementation needs to fulfill. Secondly, the chapter conducts a review that considers several architecture strategies and draws our conclusion. Lastly, the chapter proposes our architecture, implementation, and functionality overview.

5.1 Current state & revision

The proposed implementation builds on our previous work [11]. We designed the application [11] as a proof of concept (minimum viable product). Therefore, we had to review the implementation before we continued our work on new features. The following crucial changes were made upon the revision.

Firstly, we restructured the application to a dynamic back-end/front-end architecture, unlike the previous static implementation. Section 5.5 discusses the details of the new architecture. Secondly, we exploited the latest Vanilla JavaScript features. For instance, we separated the code into modules to enhance the front-end readability and the code organization. Moreover, we moved the application rendering to a new static class designed to hold the crucial front-end logic. Lastly, we completely restructured and redesigned the entire layout. Our new styling powered by the Vanilla CSS replaced the previous bootstrap. This approach allowed us to have a complete control over its layout and styling.

5.2 Functional requirements

Firstly, the application must support the MQTT protocol for the system configuration logic. The application needs to connect with the system gateway through the MQTT WebSocket. Furthermore, it must process configuration messages and publish them to the MQTT topic. The configuration message will contain modulation parameters for setting the VLP nodes. It is worth mentioning that the application needs to handle various environment settings. In other words, it must support an arbitrary number of nodes or different parameter combinations. The implementation must support the selected modulations (Section 3.3) and log the system behavior upon the user's interactions.

Secondly, the application must support testing logic for the MQTT WebSocket. It should allow the MQTT interaction, such as publishing and subscribing a message to an arbitrary topic.

Thirdly, the application should implement a node management functionality. The following life cycles must be supported. The first assumes the new node has been created

in the gateway but not in the application's database. The user should be able to add the node to the application's database using the node name and Device EUI. The second scenario assumes the new node does not exist in the system. Then, the user should be able to create a node object in the gateway and the application's database within one request. The delete procedure should request a Device EUI and remove the node from the entire system.

Lastly, the application must implement a node detection window where the user can fetch images from the robot. The window must support image rendering, robot controls, and camera settings. Furthermore, our detection algorithm should process the image before it is rendered. In other words, the window will present the image with detected nodes delimited by bounding boxes.

5.3 Non-functional requirements

The application design must be responsive to acquire a positive user experience on any device. Furthermore, the design should be easy to scale to satisfy the dynamic nature of the project. That being said, the application architecture will play a crucial role in determining the further adaptation of the system. Thus, the decision must be made wisely. The architecture should possess modern properties and carry out a fast performance.

Lastly, the back-end API should be compatible with a future mobile version of the application or allow transition with minimal changes required.

5.4 Architecture types

The architecture design is a crucial decision in application development. A well-thought-out architecture can handle various loads or adapt to changing requirements during development. Thus, it is easier to integrate new features. Moreover, good architecture can significantly improve the application performance and enhance the user experience.[24][25]

Due to the importance of this topic, this section presents common architecture strategies that we have considered. The following architectures are discussed:

- Static Web Application
- Dynamic Web Application
- Single-Page Application (SPA)
- Multiple-Page Application
- Progressive Web Application (PWA)

5.4.1 Static Web Application

Static Web Application delivers the content to the user's browser without altering the server-side code. While this site can be easy to develop, it could lead to unchanging and flat applications. Nowadays, most Static Web Applications are accompanied by a rich JavaScript code to deliver the missing responsive content.[26][27][25] Therefore, the page

may not seem static when dynamic content such as rollover images or Flash content is on the page. However, the authors in [27] state the following: the page is static if the browser receives the content without modification on the server-side. This implies that the application remains static, even if the page seems to have dynamic content because the application still does not support dynamic interaction with the user.[28]

5.4.2 Dynamic Web Application

On the other hand, Dynamic Web Applications modify the content of the page based on the user's request and thus enable interactions with a user.[25][27] The majority of today's web applications are dynamic. A dynamic application is any site that supports profile creation, making reservations or posting. Unlike the static application, the dynamic application utilizes server-side programming languages (PHP, Python, or Ruby) to change the content based on the user interaction. The server-side, responsible for the modifications, is generally called the application server. Dynamic applications are often associated with the term CRUD, which stands for Create, Read, Update, and Delete, because of application server database interactions.[28]

5.4.3 Single-Page Application

Another approach is a Single-Page Application. All necessary code for a SPA is retrieved from the server at a single initial load, and data are then managed through JavaScript APIs (XMLHttpRequest, Fetch, or AJAX).[29][25] The data are loaded in either a synchronous/asynchronous manner, allowing users to appreciate a dynamic experience with faster transitions, similar to a native application. Furthermore, since the architecture exploits APIs to communicate with the server, no additional work is required to use the API with the mobile application. In other words, the same API can be used for both, the mobile, and the web application. On the other hand, the implementation and the Search Engine Optimization (SEO) are more challenging and require more effort.[29][30] SPA is a modern architecture used by many well-known companies (Facebook, Google or GitHub).

5.4.4 Multiple-Page Application

Multiple-Page Application is a more traditional approach than SPA. Every page change requires a reload of the entire page, which leads to limited performance and worse user experience. Companies with extensive portfolios/features often prefer this architecture since they cannot fit the content to the SPA.[31] The application structure is more complex compared to SPA's because it splits its functionality into multiple independent pages. It brings more transparent navigation around the application, and SEO is easy to optimize. However, the development is more demanding than SPA's due to the structural complexity.[25][31]

5.4.5 Progressive Web Application

The architecture uses a SPA logic running alongside a service within the browser. The progressive approach allows supporting offline features, synchronization and can deliver even more native experience than SPA.[30] PWA utilizes WebAssembly to support native

functionalities that have not been accessible before. WebAssembly is a new type of code that runs in modern browsers. It is a compact binary format carrying a near-native performance and provides low-level languages (C/C++, Rust) with a compilation target. Thus, developers can use C/C++ alongside JavaScript and exchange the functionality.[32] It is worth mentioning that PWA can deliver features even if the connection is weak or none.[33]

5.5 Architecture

This section discusses the architectonic designs mentioned above in contrast to our requirements and decides our architecture aim.

As we state in our requirements, a modern application should be fast and responsive. Furthermore, our non-functional requirements demand easy scaling and adaptation of new features. Also, our implementation should develop an API design that would accommodate the future mobile application with minimal changes required. Lastly, the only architecture that cannot fulfill our functional requirements is Static Web Application. Therefore, we decided to rule this architecture out of consideration.

Our design estimation believes that we will most likely require only a small data volume for application loading, and the application will not need an extensive number of features. We believe the Multiple-Page architecture would be an unnecessary complex choice. Similar reasons lead us to rule out the PWA architecture. We believe PWA development is significantly more time-demanding compared with Dynamic Web Application or SPA and does not make sense in the project's prototyping stage. From the remaining architectures, we decided to exploit SPA design. Considering the development time demands, the SPA might be slightly more demanding than Dynamic Web Application. However, the functional properties of SPA significantly surpass the Dynamic Web Application design, since they deliver a much faster response and more native experience.

Lastly, the SPA's native properties will bring us closer to our goal of developing a multiplatform native application. In addition, PWA design employs SPA architecture which may be a valuable asset in future and could create a possible alternative to a native application development.

5.6 Implementation

This section introduces the system application implementation and discusses the back-end and front-end architecture. It explains the function of each structural component. Furthermore, it presents the functionality overview of application features.

5.6.1 Back End

We developed our back end in Python. The implementation exploits the Flask[1] environment, a widely used framework in the Python community. Flask allows building a web application back ends in a fast prototyping manner since it provides powerful APIs to handle HTTP communication and page rendering. We separated the back-end structure into logical parts. Figure 5.1 illustrates the back-end structure.

[1] https://flask.palletsprojects.com/en/2.0.x/

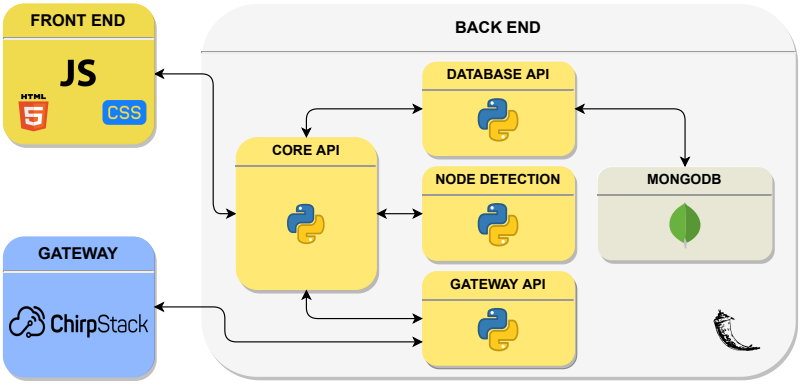


Figure 5.1 The back-end architecture.

Core API

Core API is responsible for the front-end/back-end communication. It provides the core functionality and data required by the user. It implements the initial rendering logic and handles the data exchange based on the user's interactions on the front end. The API exploits the below-presented components to satisfy the user needs on the front-end side.

Database

The Database component utilizes the MongoDB server database structure. Data in MongoDB are organized into collections. Each collection holds entities called documents in a form of the JSON. This structure allows building databases much faster and still maintains all necessary functionalities. Our database stores authentication and system data. The structure is divided into multiple collections. The first collection maintains the authentication data, and the other collections are responsible for storing the system environment data (nodes, last configuration, robot IP).

Database & Gateway API

Database API handles database authentication, requests, and responses. It is built on top of the pymongo package offered by MongoDB®. The pymongo package allows accessing the database content and handles the low-level communication. Database API implements project data querying and handling on the back-end side.

Gateway API implements authentication between the application and the gateway back end. It handles the HTTP communication and implements the node management functionality. This API can be further enhanced to cover the entire gateway management features.

Node Detection Module

The Node Detection module provides object detection algorithms and image processing techniques. Its architecture allows an easy way to include new object detection techniques in the future. Section 6.3.1 introduces the module architecture, and Section 6.3.2 discusses the implemented detection algorithm.

5.6.2 Front End

The front-end architecture comprises several parts: *Templates*, *Styles*, and *Scripts*.

Templates provide HTML files that contain pieces of the application layout. The `index.html` file presents the render hook for JavaScript and the Flask back end, which provides a special syntax to combine multiple HTML files. Therefore, we assemble the final layout by referencing the remaining templates to the `index.html`.

Styles comprise `app.css` and `icons.css` files. The `app.css` file styles the entire application, and `icons.css` provides an offline Google Fonts repository exploited for the application icons. The proposed styling employs the latest Vanilla CSS features, such as transformations/transitions, to obtain responsive and dynamic design.

Scripts contain the SPA logic that powers the page rendering, front-end/back-end communication, and MQTT communication. The scripts are discussed below.

app.js

The `app.js` script is the main JavaScript module that holds a static class called App. The class consists of multiple variables maintained by the class during one session. The class implements methods that deliver the SPA functionality. They handle the rendering logic, MQTT interactions, and set event listeners. Lastly, the `app.js` module provides the app initialization at the first load.

utils/

The `utils` directory holds helper functions that implement low-level application logic. The following scripts are provided;

The `utils.render.js` script is responsible for the application's rendering at an event occurrence. The `utils.effects.js` script implements responsive features. It transforms the sidebar menu to the "burger" menu on specific screen sizes and handles connection light bulb status. Finally, the `utils.requests.js` file provides the HTTP request functionality. It implements a custom POST/GET request function.

mqtt/

The `mqtt` directory consists of scripts delivering MQTT functionality and functions connected to the MQTT communication. The following scripts are implemented;

The `mqtt.callbacks.js` script implements MQTT callbacks. The callback functions handle incoming/outgoing messages, failed connections, and lost connections. On the other hand, the `mqtt.utils.js` file implements MQTT connection and data handling. Lastly, the directory provides the Paho[2] MQTT client developed by Eclipse Foundation. This library provides a robust MQTT over WebSocket open-source implementation.

[2] https://www.eclipse.org/paho/index.php?page=clients/js/index.php

5.6.3 Functionality overview

This section introduces an overview of functionalities and their placement in the application. Figure 5.2 presents the desktop layout, which shows all core elements, unlike the mobile version. Since our application has an adaptive design that rearranges its layout based on the window size, the mobile version hides the navigation sidebar to save space for the core functionalities. Apart from the layout differences, the functionality of both versions remains the same. This section discusses the layout from Figure 5.2.

Our implementation explores the dashboard layout, separating each core functionality into a brick element. Moreover, the application exploits popup windows to embrace other functionalities which do not fit the dashboard structure. Lastly, the application design utilizes the Single-Page Application architecture, bringing fast rendering and a native-like experience. Each application's functionality is discussed below.

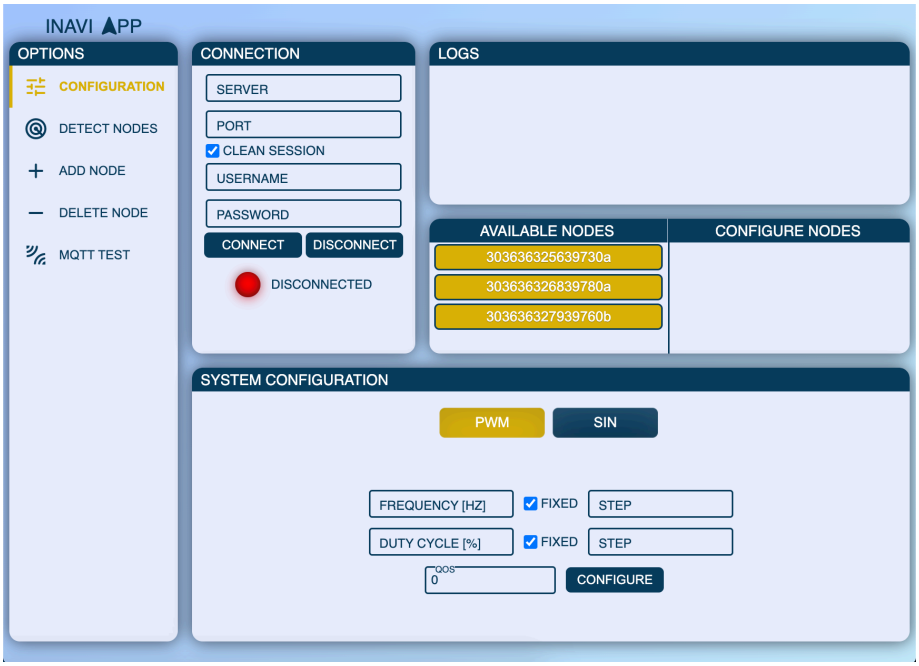


Figure 5.2 The desktop implementation of our application.

System Navigation & Tracing

The navigation sidebar lists all application views. It comprises infrastructure configuration (Figure 5.2), node detection, node management, and MQTT testing. If an option is clicked, the application renders a new view utilizing JavaScript, and required data are fetched from the back end asynchronously. Lastly, it is worth mentioning that the mobile version hides the sidebar layout and utilizes a "burger" menu that rolls out from the left side upon the user's request.

The system tracing is placed in the top right corner under the header — the tracing reports system changes performed by the user during one session. For instance, it tracks the MQTT connection status, system configuration, node detection setup, and MQTT communication.

Infrastructure configuration

The infrastructure configuration is the application's core functionality allowing the system adaptation to various fingerprinting setups. The configuration is placed at the bottom of the application. It consists of two tabs that select between PWM and sinusoidal modulation. Furthermore, each tab supports either a single node or multi-node configuration. If the user chooses the multi-node configuration, one must select nodes to be configured. The selection is performed by drag and drop, placed above the configuration element. Moreover, the user must specify at least one parameter with a variable step to acquire a unique configuration for each node. If any false procedure occurs during the configuration, the user is notified in the system trace.

Node Management

Node management is another core functionality that allows creating, adding, and deleting nodes from the system. The creation procedure appends a new node to the gateway and the application's database. On the other hand, adding node procedure only inserts the node's information to the database. This option should be used if the node object is already created through the gateway. Figure 5.3a shows the add-node popup window after clicking the *ADD NODE* option in the navigation sidebar. Similarly, the *DELETE NODE* option renders a popup-window form that submits a delete request to the gateway and application's database.

(a) (b)

Figure 5.3 The popup windows; a) Create/Add node, and b) Node detection.

Node Detection

Figure 5.3b and 5.4 introduce the node detection window. It consists of two control parts and an image space. The image space is located above the controls. It provides room for rendering an image with detected nodes (Figure 5.4). Below the image, we placed the control elements. The first element presents camera settings, robot connection, and image fetching techniques. The application supports fetching a single image or setting a fetching period to receive images periodically. The second element implements the robot control,

which allows driving the robot in different directions and adjusts the speed of the motion. My colleague in [13] provides a detailed discussion about the camera settings and robot control.

Figure 5.4 The node detection window with detected nodes.

MQTT Broker connection & Basic communication

To the right of the navigation sidebar, we placed the MQTT Broker connection. This core functionality allows inputting the Broker's authentication details and performing the connection. If the connection is established, the user is notified through the system tracing, and the notification LED becomes green. A change of the connection status is always reported in the system log upon an implemented callback.

Furthermore, the application supports MQTT communication under the *MQTT TEST* option in the navigation bar. This functionality is mainly purposed for debugging. It implements an arbitrary topic subscription that logs the topic's incoming messages into the tracing. Moreover, it allows publishing messages to a topic, including the QoS and retain-message flag.

Chapter 6
Node Detection

This chapter introduces the circle detection and edge detection mechanisms. It focuses on Circle Hough Transform and thoroughly introduces the edge detection theory. Furthermore, it proposes the node detection algorithm and our implementation.

The objective of our detection is to recognize VLP sources in an image. The algorithm will propose regions of interest (ROI) that delimits the detected object (node) and carries out its location in the image. The ROIs will be further processed to distinguish individual nodes based on their fingerprint. However, fingerprint detection is beyond the scope of this thesis and will not be discussed.

6.1 Circle Object Detection

Object detection is one of the most challenging problems in computer vision, recently powered by deep learning. Unfortunately, deep learning methods require a significant amount of data to train the prediction model. Since we do not have enough data to perform proper model training, we decided to take a more algorithmic approach.

The objective of our detection has a simple circular shape. Therefore, we needed to design a detection technique that can localize circles. There are several techniques introduced in the literature.

One utilizes geometric symmetry to detect the center candidates and separates the input image into smaller pieces based on the candidates. Finally, the method uses geometric symmetry again to detect the circles.[34] A second technique exploits the least-square method that minimizes a distance metric between image points and constraint equation.[35] Authors in [36] propose a technique that detects a circle by minimizing the algebraic distance. Similarly, a third method called Randomized Circle Detection (RCD) randomly selects four edge points and, based on a defined distance criterion, determines the location of possible circles.[37] Lastly, the most utilized circle detection technique is Circle Hough Transform (CHT). Since this technique is well documented and commonly used, we decided to exploit Hough Transform in our algorithm. The following section discusses the mathematical fundamentals and implementations in broader detail.

6.1.1 Circle Hough Transform

The general concept of Hough Transform (HT) is to convert the detection problem from image space to a parameter space, where the problem is easier to solve. In the parameter space, the detection is performed by acquiring local peaks in an accumulator array.[38]

Since the HT detects parametric curves, it needs a constraint equation to search for the shape.[39][38] Our objective is to localize circular-shaped nodes. Thus, let us assume a parametric interpretation of a circle, as in (6.1) or (6.2).

$$(\mathbf{x} - a)^2 + (\mathbf{y} - b)^2 = r^2 \tag{6.1}$$

$$\begin{aligned} \mathbf{x} &= a + r\cos\theta \\ \mathbf{y} &= b + r\sin\theta \end{aligned} \tag{6.2}$$

The parameters (a, b) denote the circle's center with radius r, θ is an angle from 0° to 360°, and (x, y) represents an edge point in the image space. In the case of a circle constraint, each point in the image space represents a circle of known radius in parameter space. If the radius is unknown, the projection in parameter space constructs a cone shape as in Figure 6.1. A simple proof of above mentioned can be obtained by reorganizing the (6.2) as follows:

$$\begin{aligned} \mathbf{a} &= x - \mathbf{r}\cos\theta, \\ \mathbf{b} &= y - \mathbf{r}\sin\theta. \end{aligned} \tag{6.3}$$

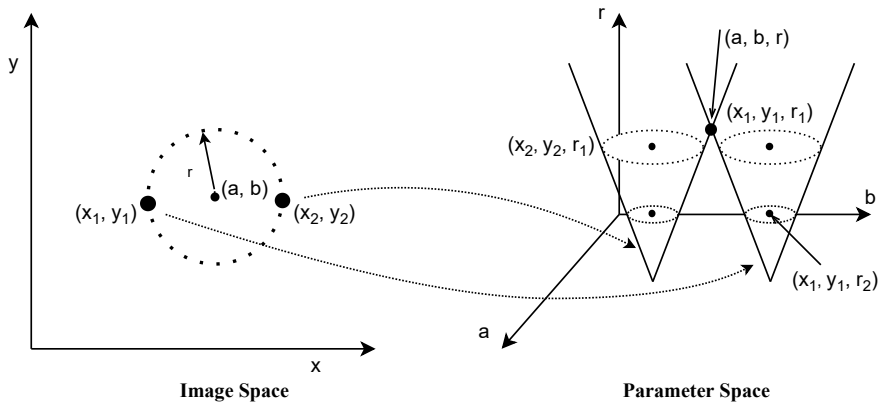


Figure 6.1 Transformation between the image space and parameter space.

Figure 6.1 demonstrates the HT algorithm. Firstly, the image is converted to an edge representation. Secondly, each edge point is transformed into the parameter space and votes coherently into the accumulator array. The accumulator array is a discrete representation of the parameter space where each discrete point represents a counter. In other words, the array contains zeros at initialization, and each edge point votes for the closest parameters in the array of its shape.[38]

Let us assume the two edge points converted in Figure 6.1 as an example. The accumulator array would be incremented at the cone's surface. The only place that would be incremented twice is the cross-section of both cones at the point (a, b, r). If the remaining edge points are applied, the accumulator array would contain all the votes. The object location is predicted based on the accumulator's local maxima or threshold that carries out the center and radius of the detected circle.

Unfortunately, the conventional HT presented above suffers from complex computations, large memory requirements, and center localization inaccuracy in noisy images. Therefore, many improved methods were introduced to tackle these limitations.[39]

H. K. Yuen et al., in [38] present a comparative study considering the Standard Hough Transform for circles (SHT), Gerig Hough method (GHT), Fast Hough Transform (FHT), and Two Stage Hough Transform (21HT).

The GHT presents a space-saving method that replaces the 3D accumulator of size N^3 with three 2D arrays of size N^2. The method performs a series of HTs in which each stage has a constant radius. The first array is used as a working space for transform accumulation, and the remaining arrays store the position, size, and radius of the candidate peaks.[38]

The 21HT separates the algorithm into two stages to reduce the computation complexity and storage. The first stage integrates along the gradient direction and radius axis at a single value of (a, b) of all edge points to find circle centers. In other words, centers are searched along the gradient direction upon which they must lie. Moreover, the 2D accumulator array stores the voting used to identify the center candidates by local peak detection. In the second stage, the method determines the radius by constructing histograms from the center parameters acquired in the first stage and the constraint equation (6.1). A radius histogram is acquired for each center candidate, and the histogram peaks indicate the circles.[38] The 21HT method is currently implemented in the OpenCV[1] library and will be utilized in our implementation.

The FHT presents a multidimensional quadtree structure that simultaneously accumulates and detects peaks in the HT. This approach can be thought of as a hierarchical search and reduces memory storage.[38]

Since edge detection plays a crucial role in HT-based algorithms, the following section was dedicated to a detailed introduction to this topic.

6.2 Edge Detection

Edge detection is another fundamental problem in computer vision. It plays an important role in object recognition, object proposal generation, and image segmentation. With the computer vision growth, edge detection has been notably improved, and complex research is still being conducted.[40][41] Before introducing the edge detection techniques, it is important to define the term *edge*.

A *physical edge* is a set of points that delimits the boundary of two distinct physical surfaces. Even though physical edges have many similarities with edges in an image, there are still significant distinctions caused by the projection from 3D to 2D scene representation.[5] Consequently, edges in the image do not necessarily correspond to physical edges. For instance, let us assume illumination at an arbitrary angle in respect to the object. The shadow cast across the object or its surroundings creates a boundary on an otherwise uniform surface. Conversely, some physical edges do not appear in the image as well. The cause is often the object's shape or lighting properties.[5] Hence, it is important to distinguish between the two types. The section will discuss edges in the context of images only.

An *edge* in an image is defined as an abrupt change in brightness intensity. It delimits an object's contours and delivers a unique feature set used for object detection. Edge features are usually extracted from the gray level change, but color or texture can sometimes be used as well. As previously discussed, illumination is an important issue regarding edge detection. It adds unwanted noise that could lead to misclassified false edges. Thus, the detection techniques propose different methods to suppress the noise and efficiently detect

[1] https://github.com/opencv/opencv-python

edges.[5]

Edge detection techniques can be grouped into two major categories:

- Traditional detection
  - First Derivative-Based (Gradient-Based)
  - Second Derivative-Based (Laplacien-Based)
- Deep Learning-based detection

Traditional edge detection techniques exploit image gradients and derivatives. They are further split into groups based on the order of the derivative used within the algorithm. These techniques and detectors are discussed in the following sections 6.2.1 and 6.2.2.

The second detection group based on deep learning is not further discussed in the chapter because it is not relevant to our development. A recent review presented by *Rui Sun et al.* in [42] discusses deep learning-based edge detection in broad detail and classifies them into logical groups.

6.2.1 Gradient-Based detection

The First Derivative algorithms exploit the gradient operator (∇) to acquire the abrupt changes in the image. The *gradient* is a vector consisting of partial derivatives. In other words, it is a vector that carries out a slope of change in a direction. Since the image space has two dimensions and each dimension is considered as the direction of the derivative, the 2D gradient needs to be defined:

$$\nabla f(x,y) = \frac{\partial f}{\partial x}\mathbf{i}_x + \frac{\partial f}{\partial y}\mathbf{i}_y = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}, \tag{6.4}$$

where $\mathbf{i}_x$ and $\mathbf{i}_y$ are unit vectors in the x and y direction.

The gradient has a magnitude and direction like every vector. The magnitude represents the maximum intensity change, and the direction points to the greatest intensity increase (points "uphill").[5][43] Eq. (6.5) defines the gradient magnitude $|\nabla f(x,y)|$ in 2D space by utilizing the Pythagorean theorem on the basis vector elements f_x and f_y. The gradient direction θ is defined in (6.6).

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2} \tag{6.5}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{f_y}{f_x}\right) \tag{6.6}$$

Since the derivative operator behaves like a high-pass filter, edge detection based on derivatives is sensitive to noise. Noise can corrupt the detection by overwhelming the edges and make the detection insufficient or even useless. The gradient itself does not deliver satisfactory edge detection unless we appropriately take care of the noise. Many detection techniques solve noise, and other challenges edge detection holds. Before the section introduces individual gradient-based detectors, it presents a typical structure that

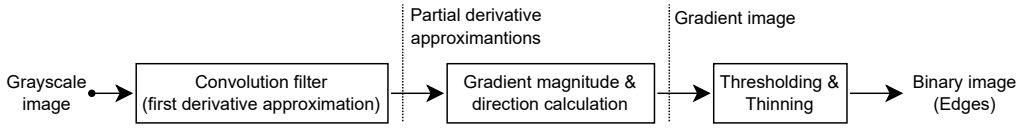


Figure 6.2 Structure of gradient-based edge detection techniques.

these detectors share. The structure is shown in Figure 6.2. Let us walk through the structure.

The structure's input is a grayscale image. Firstly, the image needs to be converted to grayscale. Then, the image is filtered through a convolution mask specific to the exploited detector. The detectors are introduced later in the section. Since the convolution is calculated in x and y direction, it approximates the first partial derivatives. Secondly, we acquired the partial derivatives, and thus we can calculate the gradient magnitude and direction by exploiting (6.5) and (6.6). The result is called a *gradient image*, which already holds some edge features, as shown in Figure 6.3b.

However, edge detection aims further. It proposes techniques that fully construct contours of individual objects with single-pixel thick boundaries — furthermore, the techniques endeavor to suppress stochastic events. They employ the structure's last step to achieve these thin boundaries. The procedure comprises two stages: Thresholding and Thinning.[5]

The Thresholding stage compares the gradient magnitude of each pixel with a predefined threshold T as defined in (6.7). If the pixel magnitude surpasses the threshold, it is considered an edge pixel. Since Thresholding tends to create strips thicker than a single pixel, the second stage, Thinning, is applied. The Thinning stage endeavors to create single-pixel boundaries by searching over a neighborhood of the edge candidates and finding the local maximum in the neighborhood. The pixels being the local maxima are then evaluated as edge points, and they construct the wanted single-pixel contours.[5]

$$|\nabla f(x, y)| \geq T \tag{6.7}$$

The question is, how to find local maxima. As mentioned above, it requires a definition of a neighborhood to search over. Let us discuss the neighborhood possibilities.

Firstly, let the neighborhood be a 2D region of any shape, for simplicity, a rectangle. Local maxima found over 2D rectangle regions within the image deliver a set of isolated points rather than fully constructed contours. Figure 6.4a) presents an example of this unwanted behavior. The 2D regions seem to be a poor choice; an alternative approach needs to be presented.

Now, let the neighborhood reduce its dimensionality over which maxima are searched. This reduction allows searching local maxima over 1D neighborhoods instead, namely, let us assume a finite line with a direction crossing the edge. This alternative approach acquires fully constructed edges if the search direction is chosen correctly. This section will present an example supporting this assumption after determining the correct direction to search the maxima. Let us introduce a couple of options;

The first option classifies the edge pixel candidate as an edge point if the pixel's gradient magnitude is a local maximum at least in one of the possible directions. Unfortunately, this approach is rather inefficient due to calculations in every direction. Moreover, it tends to create false edges.

The second option is significantly more efficient. It searches only a single direction, the gradient direction. Looking in the gradient direction means searching the region perpendicular to the edge. Thus, the approach delivers great edge localization accuracy.[5] Let us assume the gradient direction is the correct choice.

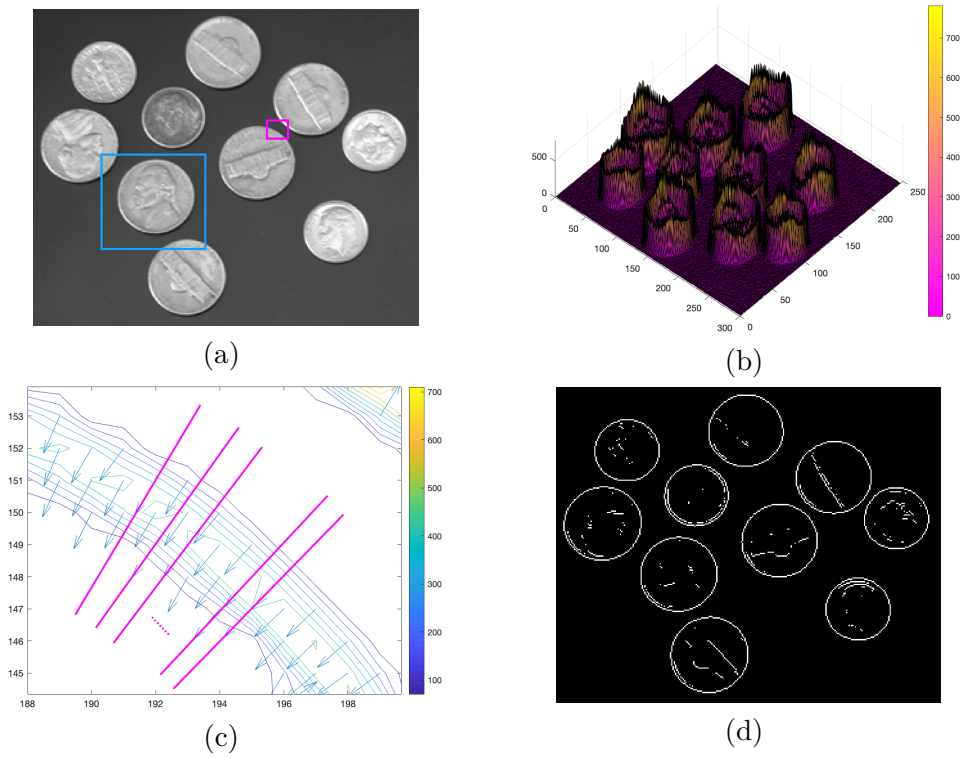


Figure 6.3 Concept of finding local maxima over 1D neighborhood; a) original image, b) pixel gradient magnitude, c) gradient magnitude (depicted as contours) and gradient direction with 1D line regions located in the ping segment of Figure 6.3a, and d) fully constructed boundaries utilizing gradient based edge detection with Sobel convolution mask.

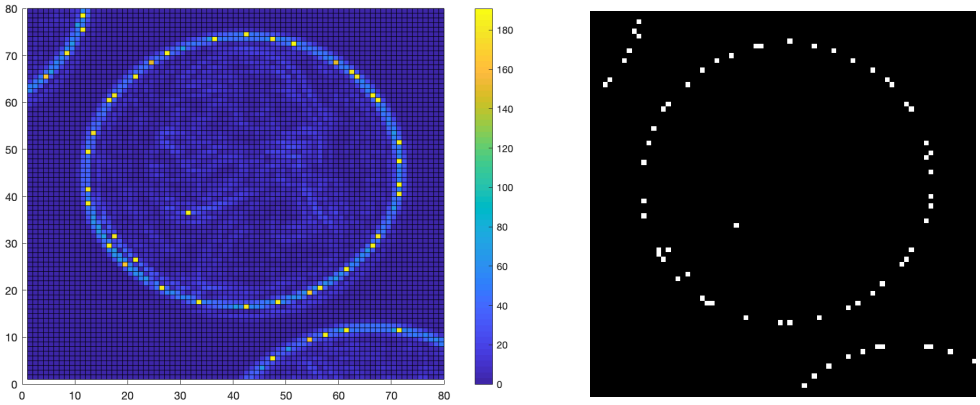
As the typical gradient-based structure was fully introduced, we can move to the above-mentioned example[2]. Let us take an actual image (Figure 6.3a) that will introduce the concepts again and prove the dimensionality reduction purpose. The image initially comes from MATLAB documentation and it was downloaded from [44]. It is a gray-level image; thus, no additional processing is needed before entering the structure.

The first step is to calculate the gradient in each pixel. The gradient must be approximated because the image space is discrete. To acquire the approximation, we need to apply one of the detectors introduced later in the section. The example uses the Sobel detector (Section 6.2.1). The second step uses equations (6.5) and (6.6) to obtain the gradient magnitude and direction from the partial derivatives. Figure 6.3b depicts the resulted gradient magnitude. The ping segment in Figure 6.3c illustrates gradient directions crossing the gradient magnitude. It is worth mentioning that the gradient direction is perpendicular to the magnitude contours. The last step to perform before obtaining edge representation is Thresholding and Thinning. Thresholding suppresses some of the unwanted noise

[2]The MATLAB script implementing the example's processing is attached to the thesis.

and obtains edge pixel candidates. The example applied the thresholding according to (6.7). Finally, the example achieved thin and accurate edges by performing Thinning. As discussed earlier, the thinning needs direction to be performed. The example uses the gradient direction to search for local maxima. Figure 6.3c illustrates the 1D line segments that were searched. The resulting representation will form fully constructed contours if each edge-candidate's local maximum is found in its 1D gradient direction neighborhood. Similar to the contours depicted in Figure 6.3d.

Conversely, if the example considered the 2D neighborhood, the representation would be as in Figure 6.4a. The figure illustrates the gradient magnitudes after Thresholding and Thinning over 2D 5×5 pixel regions. Figure 6.4b shows the edge detection utilizing the Sobel detector. It demonstrates that local maxima represented by white pixels do not construct continuous object boundaries like in the previous 1D region demonstration (Figure 6.3d).



(a) (b)

Figure 6.4 Local maxima found over 5×5 pixel 2D neighborhood in the blue segment of Figure 6.3a; a) gradient magnitude, and b) edge detection result.

Discrete Gradient Operators

The gradient cannot be calculated analytically in this application and needs to be approximated by convolution. Before introducing the edge detectors, we need to define discrete gradient operators.

Let $f(n_1, n_2)$ represent the discrete image space, where n_1 describes the horizontal and n_2 the vertical axis. The positive orientations of the n_1 and n_2 directions are right and upward, respectively. Since the gradient is calculated from pair of orthogonal directional derivatives, it is necessary to create a pair of orthogonal convolution kernels (filters) $h_1(n_1, n_2)$ and $h_2(n_1, n_2)$. Based on (6.4), the gradient approximation is defined as:[5]

$$\widehat{\nabla} f(n_1, n_2) = f_1(n_1, n_2)\mathbf{i}_{n_1} + f_2(n_1, n_2)\mathbf{i}_{n_2}, \tag{6.8}$$

where

$$\begin{aligned} f_1(n_1, n_2) &= f(n_1, n_2) * h_1(n_1, n_2), \\ f_2(n_1, n_2) &= f(n_1, n_2) * h_2(n_1, n_2). \end{aligned} \tag{6.9}$$

Similarly, the discrete gradient magnitude and direction can be derived from (6.5) and (6.6), respectively, as follows:

$$\begin{aligned} |\widehat{\nabla} f(n_1, n_2)| &= \sqrt{f_1^2(n_1, n_2) + f_2^2(n_1, n_2)}, \\ \widehat{\theta}(x, y) &= \tan^{-1}\left(\frac{f_2(n_1, n_2)}{f_1(n_1, n_2)}\right). \end{aligned} \tag{6.10}$$

The filters above define a derivative, and thus they should respond to flat image parts with zero gradients. Consequently, the filter's coefficients must satisfy the zero mean requirement. Otherwise, the previous is not fulfilled, and the derivative property is violated. The violation can result in a biased edge detection in particular directions.[45] The simplest derivative approximation schemes in 1D space are derived from the definition. Eq. (6.11) presents the first derivative and central derivative, respectively.

$$\begin{aligned} f'(x) &= \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \\ f'(x) &= \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}. \end{aligned} \tag{6.11}$$

To derive the kernel, let us assume the space is discrete and the minimal step h is equal to one. The first difference and central difference changes, respectively, as follows:[5]

$$\begin{aligned} f'(n) &= f(n+1) - f(n), \\ f'(n) &= \frac{1}{2}(f(n+1) - f(n-1)). \end{aligned} \tag{6.12}$$

Finally, the discrete form of the 1D convolution filter based on the system's impulse response would be the following (boldface denotes the origin position):[5]

$$\begin{aligned} h(n) &= \delta(x+1) - \delta(x) = \begin{bmatrix} \mathbf{1} & -1 \end{bmatrix}, \\ h(n) &= \frac{1}{2}(\delta(x+1) - \delta(x-1)) = \frac{1}{2}\begin{bmatrix} 1 & \mathbf{0} & -1 \end{bmatrix}. \end{aligned} \tag{6.13}$$

Discrete operators commonly exploit the close relationship between discrete correlation and convolution operators. Since the image space consists of real numbers, the complex conjugation in the definition of correlation can be omitted and the only remaining difference is the dimensional reversal. Hence, the discrete correlation and convolution response differ only in the sign of the gradient magnitude, not in its value. Since the magnitude value is often the part that matters in edge detection practice, the dimensional reversal is incorporated into the convolution kernel, and correlation is calculated instead to save some computations. The previously provided 1D kernels would look as follows:[5]

$$\begin{aligned} h(-n) &= \delta(x+1) - \delta(x) = \begin{bmatrix} -\mathbf{1} & 1 \end{bmatrix}, \\ h(-n) &= \frac{1}{2}(\delta(x+1) - \delta(x-1)) = \frac{1}{2}\begin{bmatrix} -1 & \mathbf{0} & 1 \end{bmatrix}. \end{aligned} \tag{6.14}$$

Lastly, let us extend the kernels to the two dimensions. Firstly, we omit the scaling factor in the central difference caused by the two-pixel distance between non-zero points. This scaling factor is easily compensated by thresholding and thus insignificant in the

expression. Both pairs of the 2D first difference and central difference correlation kernels are provided in (6.15), respectively. The h_1 is mainly sensitive to vertical edges and h_2 detects the horizontal ones. Henceforth, let the dimension reversal $-n_x$ be n_x for $x \in \{1, 2\}$ to maintain the clarity of the mathematical interpretation. Further on, kernels will be presented in this adjusted correlation interpretation.[5]

$$h_1(n_1, n_2) = \begin{bmatrix} 0 & 0 \\ -\mathbf{1} & 1 \end{bmatrix}, \qquad h_2(n_1, n_2) = \begin{bmatrix} 1 & 0 \\ -\mathbf{1} & 0 \end{bmatrix},$$
$$h_1(n_1, n_2) = \begin{bmatrix} 0 & 0 & 0 \\ -1 & \mathbf{0} & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad h_1(n_1, n_2) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & \mathbf{0} & 0 \\ 0 & -1 & 0 \end{bmatrix}. \tag{6.15}$$

Roberts detector

One of the first edge detectors was introduced in 1963 by L.G. Roberts.[43] The Roberts detector enhances the first derivative kernel from (6.15). The derivative kernels present zero-crossings at different positions, as shown in Figure 6.5. This mismatch causes an error in the estimated gradient due to measuring horizontal and vertical characteristics at different locations.[5]

The Roberts detector transforms the first derivative kernel to resolve the mismatch. It rotates the first derivative kernel by $\frac{\pi}{4}$. If origins are placed at boldface positions in (6.16), the zero-crossing is shared at one location for both new masks, in the middle of Robert detector. The detector detects sufficiently the diagonal edges. Unfortunately, the new structure produces a different issue. The zero-crossing is located off-grid, yet the edge location must be assigned to the origin pixel. It presents a location bias that can lead to significant errors. We can solve this bias by using the central difference kernel instead. It has an inherently positioned zero-crossing to an exact pixel location in the middle of the kernel. The downside of the solution is that the kernel size must be increased.[5]

$$h_1(n_1, n_2) = \begin{bmatrix} 0 & 1 \\ -\mathbf{1} & 0 \end{bmatrix}, \quad h_2(n_1, n_2) = \begin{bmatrix} 1 & 0 \\ \mathbf{0} & -1 \end{bmatrix} \tag{6.16}$$

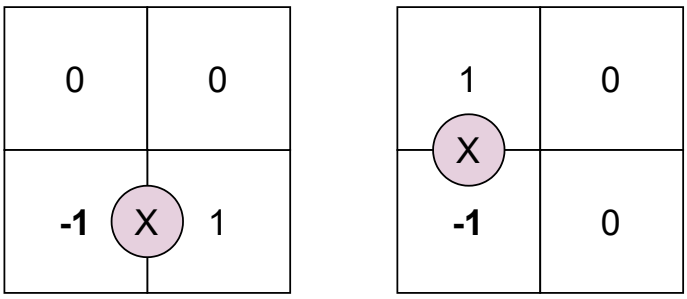


Figure 6.5 Problem of zero-crossing in first derivative kernel.

Prewitt detector

The kernels above have not yet handled the noise. As mentioned earlier, the derivative operator is sensitive to noise since it behaves like a high-pass filter. The Prewitt detector presents a smoothing mechanism that leads to noise suppression. It exploits the local averaging as the smoothing technique. The detector simultaneously calculates derivatives in one coordinate direction and suppresses noise in the orthogonal direction. We need to perform several steps to acquire the Prewitt detector. First, let us assume two filters. The first utilizes the central derivative kernel, and the second performs the three-sample averaging smoothing in the orthogonal direction. The filters look as follows:[5]

$$h_{smooth}(n_1) = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad h_{diff}(n_2) = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{6.17}$$

Since both filters are independent of their orthogonal direction, one can exploit element-wise product and form a derivative filter with an incorporated smoothing based on local averaging. Eq. (6.18) illustrates the multiplication.[5]

$$h_1(n_1, n_2) = h_{smooth}(n_1) \cdot h_{diff}(n_2) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{6.18}$$

Finally, the same procedure must be repeated for the other coordinate permutation to obtain the pair of Prewitt detectors with the following form:

$$h_1(n_1, n_2) = \begin{bmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_2(n_1, n_2) = \begin{bmatrix} 1 & 1 & 1 \\ 0 & \mathbf{0} & 0 \\ -1 & -1 & -1 \end{bmatrix}. \tag{6.19}$$

Sobel detector

The Sobel detector improves Prewitt's fundamental smoothing technique based on local three-sample averaging. It enhances the smoothing properties by designing a suitable low-pass filter kernel. Eq. (6.20) defines the Sobel detector for vertical and horizontal edge detection. The filtering produced by the low-pass filtering kernel $\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$ carries out a smoother frequency response than three-sample averaging. Thus, the Sobel detector is often selected at the expense of the Prewitt detector and enjoys wide popularity in gradient edge detection.[5]

$$h_1(n_1, n_2) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_2(n_1, n_2) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & \mathbf{0} & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{6.20}$$

6.2.2 Laplacian-Based detection

Laplacian-Based methods search the second derivative for zero-crossings.[46] Since the zero-crossing determines the precise edge location, the second derivative detection produces

single-pixel contours without Thinning. These methods represent the second derivative by the 2D Laplacian operator defined as:[5]

$$\nabla^2 f(x,y) = \nabla\nabla f(x,y) = \frac{\partial f(x,y)^2}{\partial x} + \frac{\partial f(x,y)^2}{\partial y}. \tag{6.21}$$

The Laplacian operator is isotropic and hence does not favor any edge direction. Unfortunately, the two derivatives are more prone to be affected by noise than gradient-based techniques. Moreover, the edge intensity is not considered, so even a slight intensity change makes a zero-crossing in the second derivative.[47] This behavior leads to detecting phantom edges. A *phantom edge* is a second derivative zero-crossing that does not create a local maximum in the gradient magnitude. On the contrary, it produces a local minimum. Figure 6.6 illustrates a phantom edge x_p as the first derivative minimum. Thresholding must be applied to suppress phantom edge detection. However, an incorrect threshold disrupts the continuity of the edge contours.[5][47] A couple of thresholding techniques are often used.

The first determines an edge point if it exceeds the gray-level variance threshold. Another technique thresholds the gradient magnitude or the slope of the Laplacian output at the zero-crossing. Both techniques reject some of the weak edges that are most likely caused by the noise.[5]

Finally, the Laplacian filter must obey the zero mean requirement as any derivative filter. If the zero mean requirement is broken, it creates a bias that violates zero response to a constant intensity.[5][45]

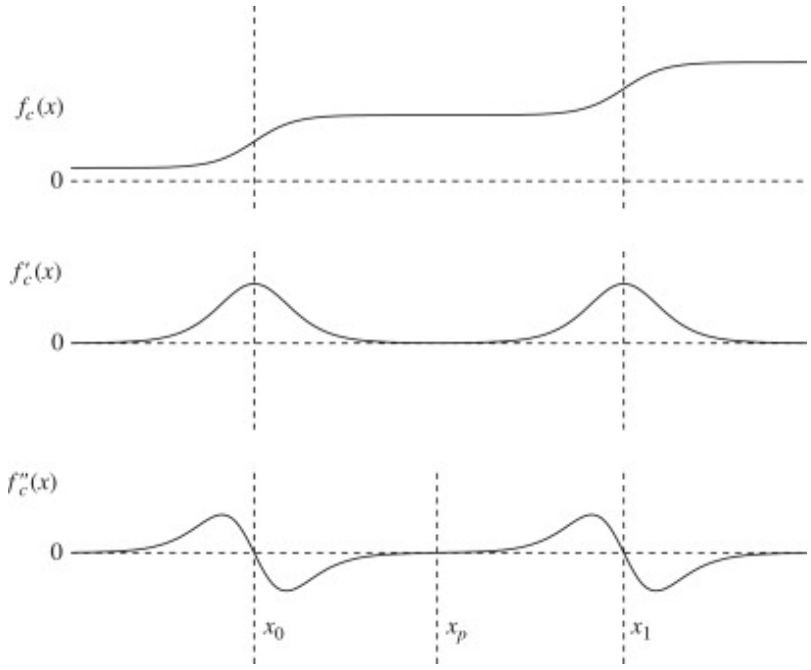


Figure 6.6 Arbitrary one dimensional function and its derivatives.[5]

Laplacian detector

Since the Laplacian is a scalar unlike gradient, we can sufficiently represent the Laplacian detector with only a single filter $h(n_1, n_2)$. To derive the filter representation, let us define the Laplacian operator in discrete space as follows:

$$\widehat{\nabla}^2 f(n_1, n_2) = f(n_1, n_2) * h(n_1, n_2), \tag{6.22}$$

where $f(n_1, n_2)$ denotes the image. Now, we need to express the first and second derivative approximations. We can use the first difference (6.12) to obtain the following:

$$\begin{aligned} \frac{\partial f(n_1, n_2)}{\partial x} &\approx \hat{f}_x(n_1, n_2) = f(n_1 + 1, n_2) - f(n_1, n_2), \\ \frac{\partial f(n_1, n_2)^2}{\partial x^2} &\approx \hat{f}_{xx}(n_1, n_2) = f_x(n_1, n_2) - f_x(n_1 - 1, n_2). \end{aligned} \tag{6.23}$$

As discussed earlier, the first difference presents an edge location error caused by the zero-crossing being placed off the grid. The second derivative solves the location error issue since it counteracts the shift presented by the first derivative.[5] Combining the two equations above, we obtain the horizontal partial derivative, and if the same is applied for the vertical direction, we acquire both partial derivatives as follows:

$$\begin{aligned} \hat{f}_{xx}(n_1, n_2) &= f(n_1 + 1, n_2) - 2f(n_1, n_2) + f(n_1 - 1, n_2) = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}, \\ \hat{f}_{yy}(n_1, n_2) &= f(n_1, n_2 + 1) - 2f(n_1, n_2) + f_x(n_1, n_2 - 1) = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}. \end{aligned} \tag{6.24}$$

Lastly, the Laplacian detector is estimated by exploiting (6.21) as:

$$\widehat{\nabla}^2 f(n_1, n_2) = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{6.25}$$

Above Laplacian detector does not consider smoothing or diagonal information. The literature presents several Laplacian filters that enhance the conventional structure and consider smoothing and diagonal information. Eq. (6.26) introduces an example. It is worth noting that the new structure obeys the zero mean requirement as the previous — the bias would violate the results otherwise. Lastly, since the Laplacian operator does not account for the direction, the filters with inverted signs maintain valid results, and thus both filters in (6.26) are equally valid.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{6.26}$$

Laplacian of Gaussian detector

Marr-Hildreth detector, called Laplacian of Gaussian (LoG), is based on the same principle as the Laplacian detector. Moreover, it adds a Gaussian kernel filtering before finding the

zero-crossings. The Gaussian mask behaves as a narrow band-pass filter and improves the noise sensitivity of the zero-crossing detection. Since the Gaussian function is smooth and localized in both spatial and frequency domains, the detection introduces fewer false edges for the same edge location accuracy than the Laplacian detector. Lastly, LoG allows effective calculations due to the separability of the Gaussian function. It enables to construct appropriate pair of 1D filters that are applied to the horizontal and vertical dimensions. The computational costs are significantly decreased because the 2D convolution becomes unnecessary.[5]

The 2D Gaussian function with zero mean is analytically expressed as:

$$g(x,y) = \frac{1}{2\pi\sigma} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \tag{6.27}$$

where σ represents the standard deviation.[5][43] The discrete form of the Gaussian kernel is obtained by sampling the Gaussian function. According to [43], the sufficient number of samples is determined as the first odd number greater than 6σ. They empirically showed that a higher number of samples do not deliver significantly better precision of Gaussian function concerning additional computations. Conversely, a smaller number of samples would lead to omitting essential details.[43] Figure 6.7 shows a sampled Gaussian function with $\sigma = 3$ and the number of samples $n = 19$.

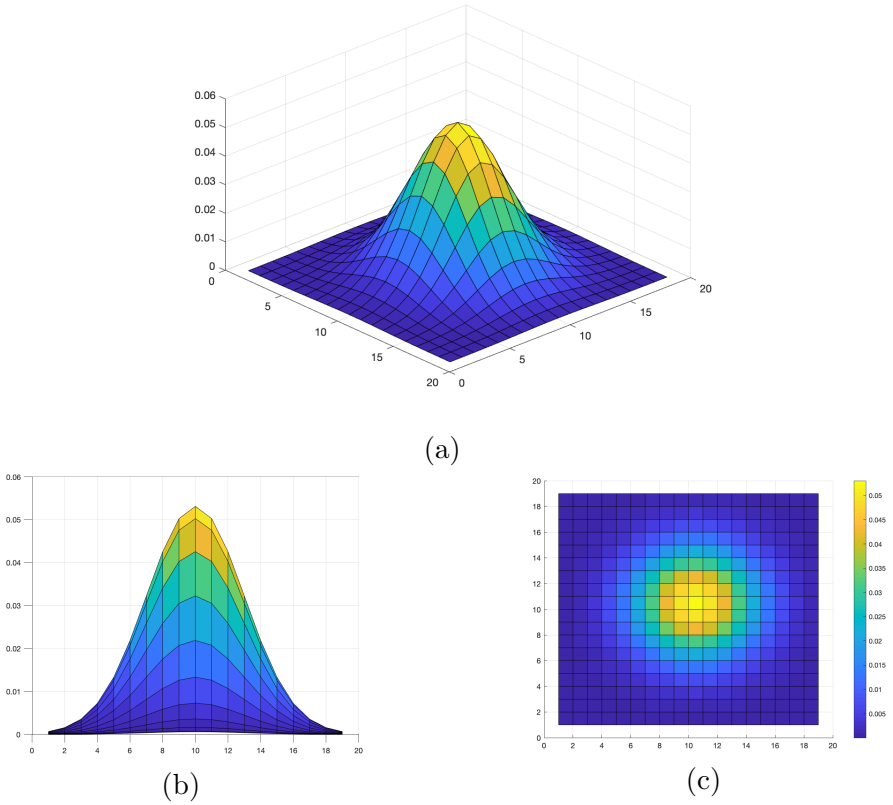


Figure 6.7 Sampled Gaussian function with $\sigma = 3$ and $n = 19$; a) 3D representation, b) 2D representation, and c) heat-map.

Since we defined the Gaussian kernel, let us derive the LoG operator. We will use the convolution's and Laplacian operator's linearity to acquire the LoG. The linearity allows changing computation order without any mathematical violation. Eq. (6.28) illustrates linearity and defines the LoG operator.

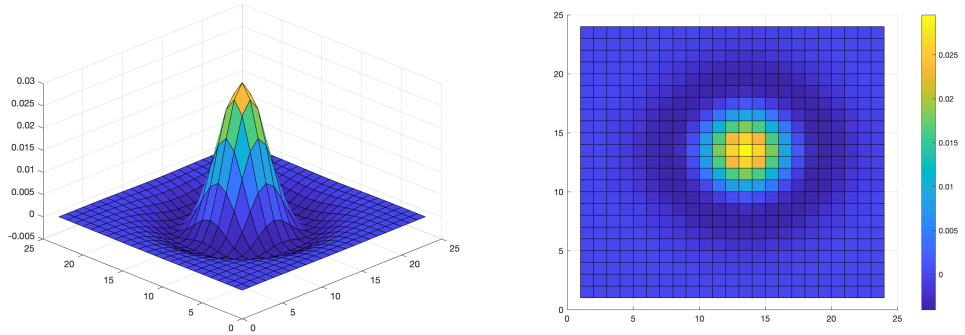
$$\nabla^2[g(x,y) * f(x,y)] = [\nabla^2[g(x,y)] * f(x,y)] = LoG(x,y) * f(x,y) \tag{6.28}$$

The different computation order brings significant benefit. Since the LoG operator is image independent, it can be prepared in advance and reduce the number of Laplacian calculations. IIt could be calculated only once instead of determining the Laplacian for every pixel.[5]

$$\begin{aligned} LoG(x,y) = \nabla^2 g(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2} = \\ = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{x^2+y^2-2\sigma^2}{\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \end{aligned} \tag{6.29}$$

Eq. (6.29) presents the LoG's analytical solution acquired by substituting the Laplacian with the LoG operator. Figure 6.8 illustrates the discrete LoG kernel sampled from analytical representation (6.29). The LoG operator has the shape of a *sombrero* which is a similar profile found in the receptive field of biological vision.[5] Since the shape character is wider, the filter size must be increased to prevent a bias from truncation.[5][45]

The LoG kernel size is commonly determined as three times the distance between the zero-crossings in the spatial domain ($\approx 8.5\sigma$).[45] In other words, three times the width of the central lobe. If the mask size increases substantially, it is more efficient to work in the frequency domain and exclude the convolution in particular cases. It would transfer the image and the mask to the frequency domain using discrete Fourier transform or Fast Fourier transform. Then, the frequency domains would be multiplied together. Lastly, the multiplication result would be transformed back to the spatial domain where edges would be expressed.[5]



(a) (b)

Figure 6.8 Sampled LoG operator with $\sigma = 3$ and $n = 25$; a) $-LoG(x,y) = -\nabla^2 g(x,y)$, and b) $-LoG(x,y)$ heat-map.

6.2.3 Canny detector

The Canny detector considerably enhances the edge detection performance of previously discussed methods.[48] It has three main goals; to optimize the detection errors, localize edges precisely and obtain only a single response per edge. The algorithm is as follows;

Firstly, it smooths the image with a Gaussian filter, reducing the details and noise. The Gaussian filter design was discussed in the previous section. Secondly, the gradient magnitude and direction are determined in each pixel according to (6.5) and (6.6).[49] The Sobel detector often approximates the partial derivatives (Section 6.2.1). Lastly, the algorithm applies a unique thresholding technique based on hysteresis called non-maxima suppression. The edge representation is defined by contours passed through the suppression. Since the non-maxima suppression uses two-level thresholding, it improves the edge contour construction and decreases the sensitivity to the threshold value.[50] The thresholding technique is crucial in the algorithm. Thus, let us focus on the suppression in broader detail.

The hysteresis non-maxima suppression defines a high T_h and low T_l threshold. Any pixel surpassing T_h is immediately accounted as an edge pixel. Pixels with gradient magnitude below T_l are excluded and clustered to the background. However, the advantage of the approach comes at magnitudes between the thresholds. The algorithm searches for pixels with this gradient magnitude and tests them to be part of the strong edge contours (magnitude above T_h). Pixels that fulfill the requirement are added to the edge contour. In other words, the algorithm finds pixels on the T_h and searches for weaker edges that would be linked to the contours below the high threshold. Weaker edges not connected to the strong edges are discarded, and connected edges are appended. The search continues until the T_l is reached and the algorithm carries out the edge representation.[50]

6.3 Implementation

The detection algorithm exploits the techniques mentioned above to fulfill its objective. The algorithm's objective is to detect nodes from an image which is captured by the mobile robot [13]. This section introduces the node_detection module, detection algorithm and results from the conducted testing.

6.3.1 Detection module

The Detection module provides image processing and detection functionality. We developed the module in Python environment utilizing the OpenCV[3] and Numpy[4] data structures.

The module comprises a static detection and processing class, an analysis script, and test helper functions. Test helper functions implement data collection handling and provide image and execution time-saving handlers. The analysis script allows tuning the detection parameters on collected images by employing the OpenCV and MatplotLib python modules. The script plots the processing and detection results to visualize the algorithm steps. Furthermore, it provides multi-image processing to accelerate the data analysis and

[3] https://pypi.org/project/opencv-python/4.5.5.64/
[4] https://pypi.org/project/numpy/1.22.3/

a user-friendly tuning of the detection algorithm. The multi-image processing handles up to nine images per run. Lastly, the node_detection module provides two fundamental classes; processing and detection class.

The `DetectionProcessing` class implements image processing that is performed before applying the detection method. It provides image loading, blurring, binarization, morphological techniques, and gradient and threshold calculation methods.

The `DetectionAlgorithms` class provides the detection method based on Circle Hough Transform. The method returns detected objects and the image with bounding boxes. The class is designed to be easily scaled with other detection algorithms by implementing a new static method within the class.

The core implementation employs the static architecture for both classes to take advantage of methods being rather bound to the class than to the class object. This approach allows calling methods without object initialization.

6.3.2 Detection algorithm

The detection is performed in several steps, as illustrated in Algorithm 1. Firstly, the image is converted into grayscale and blurred by a box filter of size (20px, 20px). The blurring reduces the image details and suppresses the noise resulting in better edge detection. Figure 6.9b illustrates the blurred image. Secondly, the blurred image is binarized to emphasize the saturated regions that represent our system nodes (Figure 6.9c). Thirdly, the image gradient is calculated to determine the thresholds dynamically. This procedure adaptively reacts to different light environments to which the system is exposed. Fourthly, the Canny edge detector determines the edge representation. The algorithm exploits the Canny detector because it carries the best results from the detectors mentioned above. Finally, the Circle Hough Transform function takes the edge representation and detects the nodes. The function returns circle objects that hold the circle center positions and their radius. Furthermore, it draws the bounding boxes to the original image, as shown in Figure 6.9d.

Algorithm 1: Pseudo-code of the node detection algorithm.

Input : image
Output: (circles, image with bounding boxes)
1 Convert RGB → Grayscale image;
2 Blur the image by box filter (20px, 20px);
3 Binarize the image with threshold (245);
4 Calculate image gradient;
5 Adaptively determine thresholds based on the gradient;
6 Canny edge detection with hysteresis non-maxima suppression;
7 Run Circle Hough Transform-based detection on the processed image;

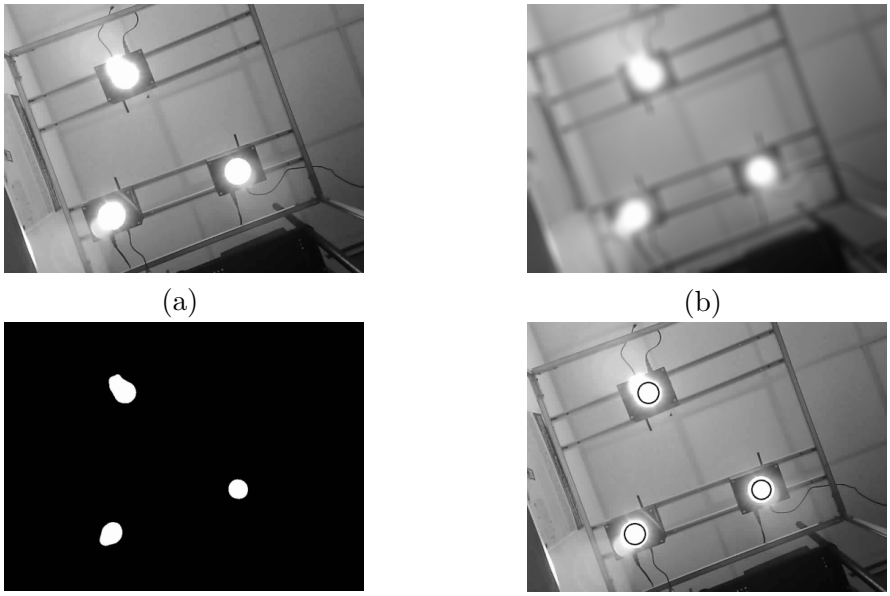


Figure 6.9 Algorithm steps illustration; a) original image in grayscale, b) blurred image with box filter, c) binarized image, and d) processed image with bounding boxes.

6.3.3 Test environment

We designed a performance test to verify the algorithm's functionality and weaknesses. The test was performed on our testbed and had the following scenario.

Firstly, the testbed area was divided into a grid to cover the test space and prevent the position bias. Grid points of the (75 cm × 75 cm) area were spaced 10 cm apart, as shown by red lines in Figure 6.10. Secondly, we defined a series of system configurations and robot positioning strategies to verify the algorithm's sensitivity to data augmentation. Furthermore, it allowed ruling out the model overfitting to certain environment setups. Lastly, we analyzed the collected data and evaluated the algorithm performance with appropriate metrics. The collected data comprise 351 images captured in various settings. Section 6.3.4 presents the evaluation.

System		**Robot Position**		**Images**
Freq. Range [Hz]	**Duty Cycle** [%]	**Phase**	**Grid Shift**	
100–300	20	aligned	(0, 0)	49
100–300	50	aligned	(0, 0)	49
100–300	80	aligned	(0, 0)	49
100–300	20	perp.	(0, 0.5)	49
100–300	50	perp.	(0.5, 0.5)	49
1000–3000	80	perp.	(0.5, 0)	49
1000–3000	80	random	(0.5, 0)	57
Total number of images				**351**

Table 6.1 Environment settings where Phase is relative to the nodes orientation and Grid Shift is relative to the main grid.

Table 6.1 lists the environment settings. The first and second column of the table defines the system configuration. The Phase column denotes the robot position with respect to the node's orientation. If the "aligned" Phase, the robot travels in the direction of the longer node side. On the other hand, "perp." Phase denotes traveling direction perpendicular to "aligned" and "random" Phase represents a random robot alignment. The fourth column represents a shift of the coordinate system regarding the main grid. For instance, the Grid Shift (0.5, 0.5) defines horizontal and vertical shift of the main grid by 0.5 step size (5cm). Figure 6.10 illustrates this shifting scenario with a red and blue grid where red lines denote the main grid and blue lines the shifted grid. The last column of the table presents the number of captured images in the session.

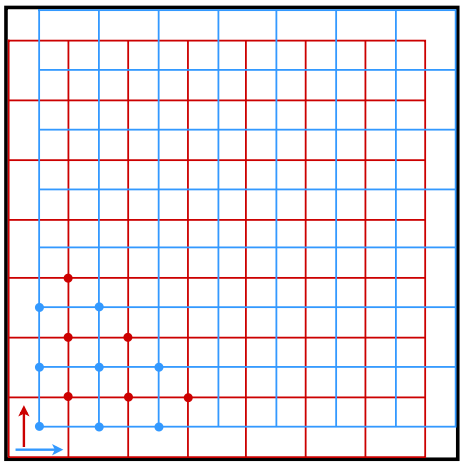


Figure 6.10 The grid layout in the testbed area; red dots in the bottom-left corner depict an example of capture positions if the Grid Shift is (0,0) and blue dots for Grid Shift (0.5,0.5); the red arrow denotes the travel direction if "aligned" Phase and blue arrow if "perp." Phase.

6.3.4 Algorithm evaluation

Our evaluation employs three commonly used object detection metrics: Precision, Recall, and F1 score. Precision delivers the ability of the model to detect only relevant objects, and is defined in Eq. (6.30). Recall denotes the ability to detect all ground truths, and is defined in Eq. (6.31).[51]

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \tag{6.30}$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}} \tag{6.31}$$

Finally, F1 score measures the balance between the precision and recall. The closer the F1 score to one, the more balanced the metrics are. Eq. (6.32) defines the F1 score calculation, an average of the above metrics.

$$F1 = 2\frac{P \cdot R}{P + R} \tag{6.32}$$

It is important to note that recall plays a superior role in our project. Detecting a non-existent object is unfortunate, but it can be discarded during fingerprint detection. On the other hand, data of an undetected node is lost immediately, and the navigation can be compromised due to a low number of detected nodes. Thus, our algorithm must detect all nodes despite a slight precision decrease.

Variables that need to be defined and acquired to calculate the above metrics, are True Positive (TP), False Positive (FP), and False Negative (FN). They have the following meaning in our project:[51]

- True Positive (TP) – a node was successfully detected in the image;
- False Positive (FP) – an incorrect detection of non-existent node or misplaced detection of the object in the image;
- False Negative (FN) – an undetected node in the image.

The test dataset comprises 351 images. The algorithm detected 868 nodes, missed 16, 26 were misplaced, and five were non-existent. Test results are listed in Table 6.2. The algorithm precision is 96.6 %, the recall is 98.2 %, and the average execution time[5] is 18.4 ms. As we claimed before, we aimed to acquire a high recall result. We believe 98.2 % accomplishes our goal. Furthermore, a great precision×recall balance was obtained, as the F1 score confirms. Thus, we assume that our algorithm is functional.

Recall [%]	**Precision** [%]	**F1 score**	$\mathbf{t_{avg}}$ [ms]
98.2	96.6	0.974	18.4

Table 6.2 Algorithm evaluation results.

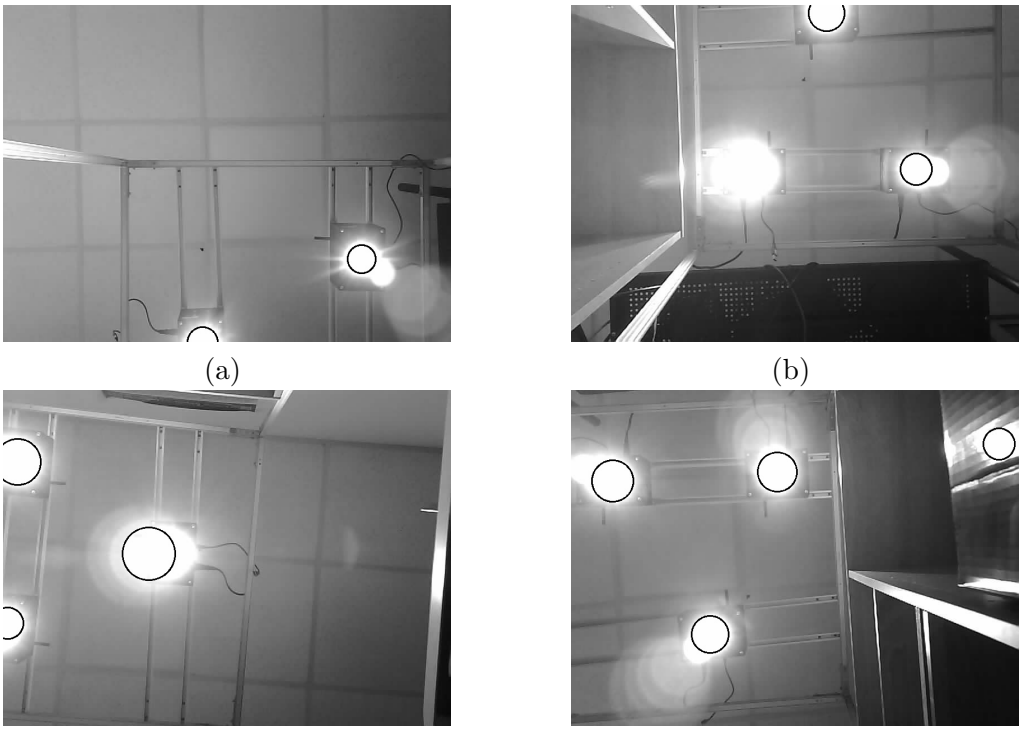
Strengths & Weaknesses

We observed the algorithm behavior on different stimuli during the test execution. The following was concluded. The algorithm performed very well in detecting partial circles lying near the edges of the image. Figures 6.11a and 6.11c present two examples. A slight move in direction revealing the node resolved the detection if the node missed near the edge.

The rest of the missed nodes were caused by the overwhelming saturation of the CMOS sensor around the node. Figure 6.11b shows the saturated region around the node borders, causing the classification error. Furthermore, the saturation caused a bounding box misplacement in several cases, as shown in Figure 6.11c. We believe the saturation side effect can be resolved by a proper configuration of the CMOS camera.

Lastly, the only false detected non-existent object was caused by a reflection. Figure 6.11d illustrates the reflection issue. We believe this effect can be suppressed again by the camera configuration and remaining errors can be discarded during the fingerprint detection.

[5]Performed on Macbook Pro Early 2015, 2.7 GHz Intel Core i5, 8 GB 1867 MHz DDR3.



(a) (b)

(c) (d)

Figure 6.11 Examples of detection strengths and weaknesses.

Chapter 7

Conclusions

This thesis proposed a functional VLP infrastructure that allows a user-friendly development of indoor navigation algorithms. Chapter 2 introduced our architecture exploiting modern communication protocols to acquire different VLP environments and settings. In addition, the thesis proposed a custom firmware that drives the VLP transmitters, a system application, and a node detection algorithm.

The firmware employs the LoRaWAN® stack for acquiring the configuration messages from the system gateway. Furthermore, it implements several fingerprinting techniques that could be used for initial positioning development. The firmware was successfully tested by multiple scenarios discussed in Section 4.4.

The system application's design is proposed in Chapter 5. It exploits SPA architecture allowing fast rendering and a native app experience. The application comprises a front end implemented in conventional web languages and a back end powered by Python. It supports the MQTT communication protocol that enables interaction with the system gateway. Moreover, the application allows single and multi-node configurations that set nodes with a unique fingerprint. This functionality was verified in the firmware testing mentioned above. Lastly, the application implements a system node management that allows creating, adding, and deleting nodes from the VLP environment.

Finally, the thesis proposed a node detection algorithm based on Circle Hough Transform. The algorithm achieves a high recall and precision, which was demonstrated by a relevant experiment presented in sections 6.3.3 and 6.3.4. The experiment considered 351 test images and carried out the following results. The algorithm's recall was 98.2 %, we achieved a precision of 96.6 %, and the precision×recall balance was verified by the F1 score of 0.974. Moreover, the algorithm achieved a sufficient real-time execution duration of 18.4 ms. We conclude from the above results that our algorithm was effectively verified and functionally sufficient. Lastly, we noticed the following characteristics during the evaluation. The algorithm performed well in detecting partial circles near the image edges. On the other hand, it struggled with some saturated regions and reflections, which caused several classification errors. However, we believe these errors can be corrected with a proper CMOS camera configuration.

7.1 Future work

Our project's vision comprises the following:

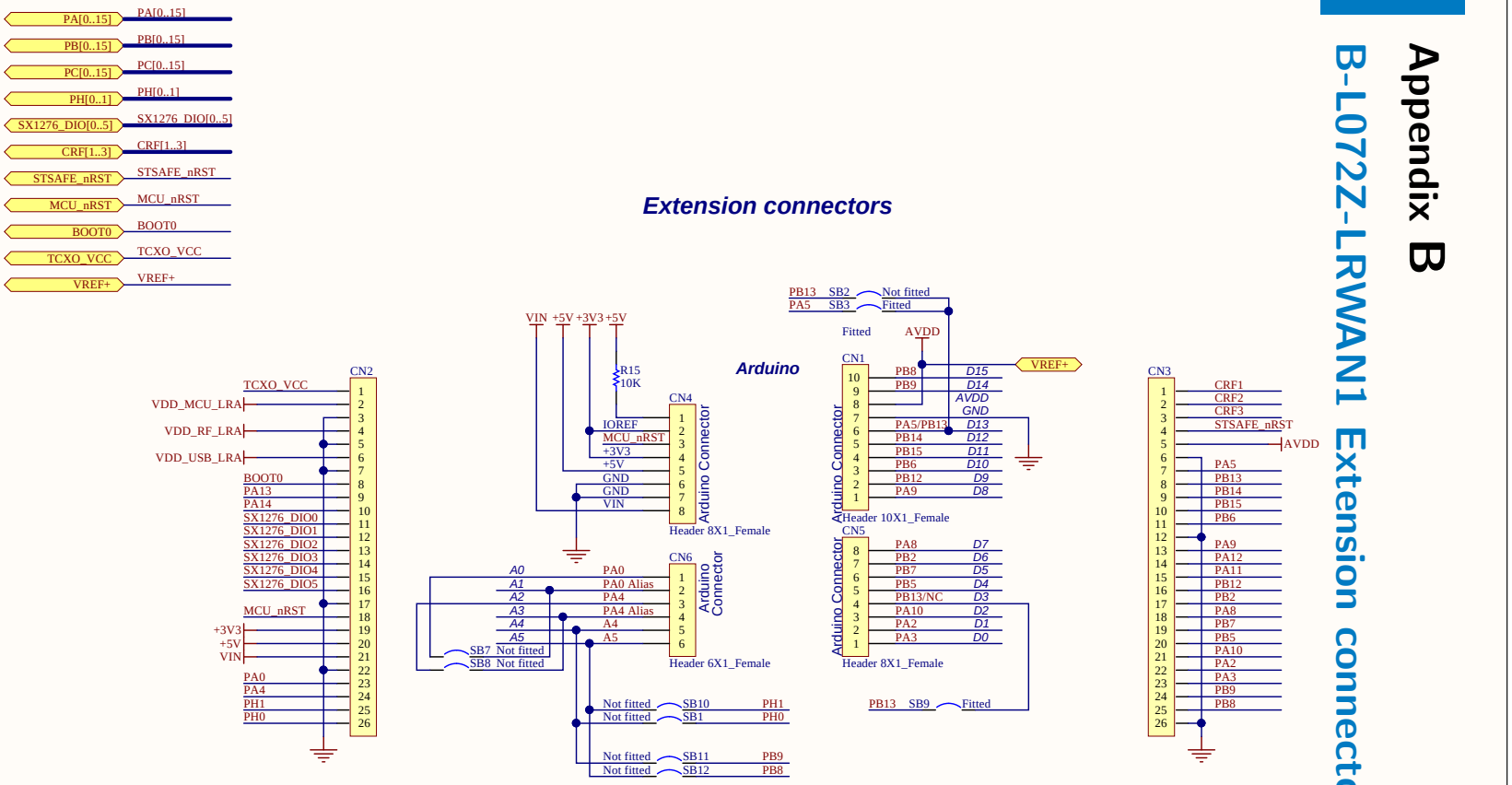
- We want to expand the fingerprinting methods implemented in the system.
- We aim to analyze possible options for enhancing the workflow performance for indoor navigation development and embrace the analysis results.
- We wish to implement more complex configuration methods allowing fingerprinting combinations and embracing data transmission over VLC channel.
- We want to propose a fingerprint detection algorithm that would enable the testing of object localization using the CMOS-based method.
- We want to present a photodiode receiver to enhance the number of testable positioning methods utilizing our system.
- We want to implement a native mobile application that would form an ideal combination with the developed application in this thesis.

Appendix A

List of Abbreviations

4PPM	Four-Pulse Position Modulation.
ADC	Analog-to-Digital Converter.
AES	Advanced Encryption Standard.
ALS	Ambient Light Sensor.
API	Application Programming Interface.
CDM	Code Division Multiplexing.
CMOS	Complementary Metal-Oxide-Semiconductor.
CSS	Cascading Style Sheets.
DAC	Digital-to-Analog Converter.
DMA	Direct Memory Access.
EMI	Electromagnetic Interference.
FDM	Frequency Division Multiplexing.
GCC	GNU Compiler Collection.
GPIO	General-Purpose Input/Output.
GPS	Global Positioning System.
HAL	Hardware abstraction layer.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
IDE	Integrated Development Environment.
ITF	Interleaved Two of Five.
JSON	JavaScript Object Notation.
LED	Light Emitting Diode.
LoG	Laplacian of Gaussian.
LoRa®	Long Range.
LoRaWAN®	Long Range Wide Area Network.
MAC	Medium Access Control.

MCU	Microcontroller Unit.
MQTT	Message Queuing Telemetry Transport.
OOK	On-Off Keying.
OTAA	Over-the-Air Activation.
PWA	Progressive Web Application.
PWM	Pulse Width Modulation.
RF	Radio Frequency.
RSE	Rolling Shutter Effect.
RTC	Real-Time Clock.
RTOS	Real-Time Operating System.
SEO	Search Engine Optimization.
SPA	Single-Page Application.
SPI	Serial Peripheral Interface.
TDM	Time Division Multiplex.
UART	Universal Asynchronous Receiver-Transmitter.
VLC	Visible Light Communication.
VLP	Visible Light Positioning.
VPPM	ariable-Pulse Position Modulation.
XML	Extensible Markup Language.



Title: **Connectors**		
Project: **STM32 LoRa Discovery**		
Size: A4	Reference: MB1296	Revision: C-01
Date: 25/11/2016		Sheet: 6 of 6

life.augmented

Appendix C

Bibliography

[1] H. Aoyama and M. Oshima, "Visible light communication using a conventional image sensor," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015, pp. 103–108.

[2] J. Fang, Z. Yang, S. Long, Z. Wu, X. Zhao, F. Liang, Z. L. Jiang, and Z. Chen, "High-speed indoor navigation system based on visible light and mobile phone," *IEEE Photonics Journal*, vol. 9, no. 2, pp. 1–11, 2017.

[3] C. Xie, W. Guan, Y. Wu, L. Fang, and Y. Cai, "The led-id detection and recognition method based on visible light positioning using proximity method," *IEEE Photonics Journal*, vol. 10, no. 2, pp. 1–16, 2018.

[4] STMicroelectronics, "Stm32 lorawan® expansion package for stm32cube," USA, 2021. [Online]. Available: https://www.st.com/resource/en/user_manual/um2073-stm32-lorawan-expansion-package-for-stm32cube-stmicroelectronics.pdf

[5] P. A. Mlsna and J. J. Rodríguez, "Chapter 19 - gradient and laplacian edge detection," in *The Essential Guide to Image Processing*, A. Bovik, Ed. Boston: Academic Press, 2009, pp. 495–524. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123744579000196

[6] P. Chen, M. Pang, D. Che, Y. Yin, D. Hu, and S. Gao, "A survey on visible light positioning from software algorithms to hardware," *Wireless Communications and Mobile Computing*, vol. 2021, 2021.

[7] Z. Zhou, M. Kavehrad, and P. Deng, "Indoor positioning algorithm using light-emitting diode visible light communications," *Optical Engineering*, vol. 51, no. 8, pp. 1 – 7, 2012. [Online]. Available: https://doi.org/10.1117/1.OE.51.8.085009

[8] Y.-S. Kuo, P. Pannuto, K.-J. Hsiao, and P. Dutta, "Luxapose: Indoor positioning with mobile phones and visible light," in *Proceedings of the 20th annual international conference on Mobile computing and networking*, 2014, pp. 447–458.

[9] R. Zhang, W.-D. Zhong, K. Qian, S. Zhang, and P. Du, "A reversed visible light multitarget localization system via sparse matrix reconstruction," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 4223–4230, 2018.

[10] Z. Zhang, H. Chen, X. Hong, and J. Chen, "Accuracy enhancement of indoor visible light positioning using point-wise reinforcement learning," in *Optical Fiber Communication Conference (OFC) 2019*. Optica Publishing Group, 2019, p. Th3I.3. [Online]. Available: http://opg.optica.org/abstract.cfm?URI=OFC-2019-Th3I.3

[11] M. Suda, "Sw for indoor visible light positioning testbed," Bachelor's Thesis, CTU FEE, Technická 2, 5 2020.

[12] S. Bosak, "Hw for indoor visible light positioning testbed," Bachelor's Thesis, CTU FEE, Technická 2, 5 2020.

[13] ——, "Mobile-robot and platform for vlc indoor navigation," Master's Thesis, CTU FEE, Technická 2, 5 2022.

[14] W. Guan, Y. Wu, C. Xie, L. Fang, X. Liu, and Y. Chen, "Performance analysis and enhancement for visible light communication using cmos sensors," *Optics Communications*, vol. 410, pp. 531–551, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030401817309458

[15] W. Guan, X. Zhang, Y. Wu, Z. Xie, J. Li, and J. Zheng, "High precision indoor visible light positioning algorithm based on double leds using cmos image sensor," *Applied Sciences*, vol. 9, no. 6, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/6/1238

[16] S.-H. Song, D.-C. Lin, Y.-H. Chang, Y.-S. Lin, C.-W. Chow, Y. Liu, C.-H. Yeh, K.-H. Lin, Y.-C. Wang, and Y.-Y. Chen, "Using dialux and regression-based machine learning algorithm for designing indoor visible light positioning (vlp) and reducing training data collection," in *Optical Fiber Communication Conference (OFC) 2021*. Optica Publishing Group, 2021, p. Tu5E.3. [Online]. Available: http://opg.optica.org/abstract.cfm?URI=OFC-2021-Tu5E.3

[17] J. Jin, L. Feng, J. Wang, D. Chen, and H. Lu, "Signature codes in visible light positioning," *IEEE Wireless Communications*, vol. 28, no. 5, pp. 178–184, 2021.

[18] A. F. Hussein, H. Elgala, and T. D. C. Little, "Visible light communications: Toward multi-service waveforms," in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2018, pp. 1–6.

[19] M. Saadi, T. Ahmad, Y. Zhao, and L. Wuttisttikulkij, "An led based indoor localization system using k-means clustering," in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2016, pp. 246–252.

[20] U. Nadeem, N. Hassan, M. Pasha, and C. Yuen, "Highly accurate 3d wireless indoor positioning system using white led lights," *Electronics Letters*, vol. 50, no. 11, pp. 828–830, 2014.

[21] S.-Y. Jung, S. Hann, and C.-S. Park, "Tdoa-based optical wireless indoor localization using led ceiling lamps," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1592–1597, 2011.

[22] T. Sato, S. Shimada, H. Murakami, H. Watanabe, H. Hashizume, and M. Sugimoto, "Alisa: A visible-light positioning system using the ambient light sensor assembly in a smartphone," *IEEE Sensors Journal*, vol. 22, no. 6, pp. 4989–5000, 2022.

[23] K. Abe, T. Sato, H. Watanabe, H. Hashizume, and M. Sugimoto, "Smartphone positioning using an ambient light sensor and reflected visible light," in *2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2021, pp. 1–8.

[24] H. Dhaduk, "An ultimate guide to web application architecture," https://www.simform.com/blog/web-application-architecture/, 2021, accessed: 2022-01-21.

[25] Spaceotechnologies, "9 different types of web applications (examples + use cases)," https://www.spaceotechnologies.com/types-of-web-applications/, 2021, accessed: 2022-01-21.

[26] Staticapps, "Defining static web apps," https://www.staticapps.org/articles/defining-static-web-apps/, 2014, accessed: 2022-01-21.

[27] Adobe, "Understand web applications," https://helpx.adobe.com/gr_en/dreamweaver/using/web-applications.html#processing_static_web_pages, 2021, accessed: 2022-01-21.

[28] Pluralsight, "The differences between static and dynamic websites," https://www.pluralsight.com/blog/creative-professional/static-dynamic-websites-theres-difference, 2020, accessed: 2022-01-21.

[29] Mozilla and individual contributors, "Spa (single-page application)," https://developer.mozilla.org/en-US/docs/Glossary/SPA, 2021, accessed: 2022-01-21.

[30] Y. Luchaninov, "Web application architecture in 2021: Moving in the right direction," https://mobidev.biz/blog/web-application-architecture-types, 2021, accessed: 2022-01-21.

[31] Asperbrothers, "Single page application (spa) vs multi page application (mpa) — two development approaches," https://asperbrothers.com/blog/spa-vs-mpa/, 2019, accessed: 2022-01-21.

[32] Mozilla and individual contributors, "Webassembly," https://developer.mozilla.org/en-US/docs/WebAssembly, 2021, accessed: 2022-01-21.

[33] S. Richard and P. LePage, "What are progressive web apps?" https://web.dev/what-are-pwas/, 2020, accessed: 2022-01-21.

[34] C.-T. Ho and L.-H. Chen, "A fast ellipse/circle detector using geometric symmetry," *Pattern Recognition*, vol. 28, no. 1, pp. 117–124, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/003132039400077Y

[35] Z. Yao and W. Yi, "Curvature aided hough transform for circle detection," *Expert Systems with Applications*, vol. 51, pp. 26–33, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417415008210

[36] W. Gander, G. H. Golub, and R. Strebel, "Least-squares fitting of circles and ellipses," *BIT Numerical Mathematics*, vol. 34, no. 4, pp. 558–578, 1994.

[37] T.-C. Chen and K.-L. Chung, "An efficient randomized algorithm for detecting circles," *Computer Vision and Image Understanding*, vol. 83, no. 2, pp. 172–191, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314201909233

[38] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 1, pp. 87–116, 1988. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0734189X88800331

[39] T. D'Orazio, C. Guaragnella, M. Leo, and A. Distante, "A new algorithm for ball recognition using circle hough transform and neural classifier," *Pattern Recognition*, vol. 37, no. 3, pp. 393–408, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320303002280

[40] S.-M. Hou, C.-L. Jia, Y.-B. Wanga, and M. Brown, "A review of the edge detection technology," *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing*, vol. 1, no. 2, p. 26—37, Oct. 2021. [Online]. Available: http://www.sparklinglightpublisher.com/index.php/slp/article/view/16

[41] Z. Su, W. Liu, Z. Yu, D. Hu, Q. Liao, Q. Tian, M. Pietikäinen, and L. Liu, "Pixel difference networks for efficient edge detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 5117–5127.

[42] R. Sun, T. Lei, Q. Chen, Z. Wang, X. Du, W. Zhao, and A. Nandi, "Survey of image edge detection," *Frontiers in Signal Processing*, vol. 2, p. 826967, 03 2022.

[43] H. Spontón and J. Cardelino, "A Review of Classic Edge Detectors," *Image Processing On Line*, vol. 5, pp. 90–123, 2015, https://doi.org/10.5201/ipol.2015.35.

[44] "Coins.png," USA, 2014. [Online]. Available: https://www.bogotobogo.com/Matlab/images/MATLAB_DEMO_IMAGES/coins.png

[45] S. R. Gunn, "On the discrete representation of the laplacian of gaussian," *Pattern Recognition*, vol. 32, no. 8, pp. 1463–1472, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320398001630

[46] G. Shrivakshan and C. Chandrasekar, "A comparison of various edge detection techniques used in image processing," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 269, 2012.

[47] X. Wang, "Laplacian operator-based edge detectors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 886–890, 2007.

[48] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[49] L. Ding and A. Goshtasby, "On the canny edge detector," *Pattern Recognition*, vol. 34, no. 3, pp. 721–725, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320300000236

[50] G. Shrivakshan and C. Chandrasekar, "A comparison of various edge detection techniques used in image processing," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 269, 2012.

[51] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242.