VRIJE
UNIVERSITEIT
BRUSSEL

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Toegepaste Informatica

INDOOR POSITIONING BY VISIBLE LIGHT COMMUNICATION

RONALD MICHIELS

Academic year 2020–2021

Promoter: Prof. Dr. Beat Signer
Advisor: Maxim Van de Wynckel

Faculty of Sciences and Bio-Engineering Sciences

VRIJE
UNIVERSITEIT
BRUSSEL

Afstudeer eindwerk ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de Ingenieurswetenschappen: Toegepaste Computerwetenschappen

INDOOR POSITIONING BY VISIBLE LIGHT COMMUNICATION

RONALD MICHIELS

Academiejaar 2020–2021

Promoter: Prof. Dr. Beat Signer
Advisor: Maxim Van de Wynckel

Faculteit Wetenschappen en Bio-ingenieurswetenschappen

Abstract

Nowadays, many people rely on computers and smart devices for ordinary activities such as sending emails, calling, texting, reading the news and interacting with social media. One important use case in which technology supports us is outdoor navigation, also known as GPS (Global Positioning System). GPS technology allows us to navigate the outside world easily. Indoor navigation in a building or underground is a different story however. This thesis is aimed at investigating IPS (Indoor Positioning Systems) using VLC (Visible Light Communication).

IPS using different technologies exist, such as Wi-Fi, Bluetooth, Infrared and other technologies. The main advantage of using VLC is that this technique uses ordinary light as transmission medium and can potentially make use of already existing lighting infrastructure.

We discuss what VLC is and how we can use it to help us with indoor positioning. We investigate some practical use cases for such a system and the reason it can be useful. We look at the technical side of VLC: how are data transmitted and received, the kind of devices that are needed to use VLC and we discuss the positives and negatives of each device type. We also look at how data can be transmitted and decoded using VLC, as well as the positioning algorithms that exist and can be used in a VLC system.

In this work, we take a look at interesting related academic works that utilize a VLC system. We focus primarily on other related works that use image sensors (or cameras). This is the type of device that is used in this thesis as well, since it has some advantages over light sensors. One important reason for its usage is the popularity of the smart device. Almost every smart device is equipped with a camera, which can be used as a receiver in VLC applications.

Our proposed solution uses a proximity-based technique to determine the approximate location of the user. We discuss how we detect light sources, error correction, decoding of messages and more.

We evaluate the solution using two different types of tests: using controlled experiments and simulation tests. In our controlled experiment, we change one parameter at a time such as distance or frame rate to evaluate the effect on our solution's performance. Simulation tests on the other hand are uncontrolled and consist of a person walking from one room to another with their

smart device in order to test the performance of the implemented solution in a more realistic scenario.

Samenvatting

Tegenwoordig vertrouwen veel mensen op computers en slimme apparaten om gewone activiteiten uit te voeren zoals e-mails verzenden, telefoneren, sms'en, het nieuws lezen en interageren met allerlei sociale media. Een andere belangerijke use case waarin technologie ons ondersteunt is buiten-navigatie, ook bekend als GPS (Global Positioning System). GPS technologie laat ons toe om makkelijk buiten te navigeren. Binnen-navigatie in een gebouw of ondergronds is echter een ander verhaal. Deze thesis mikt erop om IPSs (Indoor Positioning Systems) te onderzoeken die gebruik maken van VLC (Visible Light Communication).

IPSs die andere technologieën bezigen, zoals Wi-Fi, Bluetooth, Infrarood of andere bestaan. Het primaire voordeel van VLC vergeleken met deze andere technologieën is het feit dat VLC gewoon licht gebruikt als transmissiemedium en de techniek mogelijks de bestaande verlichtings-infrastructuur kan hergebruiken.

We zullen bespreken wat VLC is en hoe het ons kan helpen met binnennavigatie. We onderzoeken enkele praktische gebruiksgevallen. We bekijken de technische kant van VLC: hoe worden de data doorgestuurd en ontvangen, de soorten toestellen die nodig zijn om VLC te gebruiken en de positieve en negatieve kanten van elk type toestel. We bekijken ook hoe de data kunnen gedecodeerd worden, alsook de positionerings-algoritmen die bestaan en benut worden in een VLC systeem.

In dit werk bekijken we ook interessant gerelateerde academische werken die een VLC systeem gebruiken. We focussen hierbij vooral op onderzoek dat gebruik maakt van afbeelding-sensoren (of camera's). Dit is het soort toestel dat we ook in deze thesis gebruiken, aangezien het enkele voordelen biedt over licht sensoren. Een belangrijke reden om camera's te gebruiken is de populariteit van het slimme apparaat. Bijna elk slim apparaat is uitgerust met een camera die kan gebruikt worden als ontvanger in een VLC applicatie.

Onze voorgestelde oplossing maakt gebruik van een nabijheid-gebaseerde techniek om bij benadering de positie van de gebruiker te bepalen. We

bespreken hoe we de lichtbronnen detecteren, fout-detectie doen, berichten decoderen en meer.

We evalueren de oplossing aan de hand van twee verschillende tests: gecontroleerde experimenten en simulatie experimenten. In onze gecontroleerde test veranderen we één parameter per keer, zoals de afstand of frame rate om het effect te evalueren op de performantie van onze oplossing. Simulatie tests aan de andere kant zijn ongecontroleerde tests die bestaan uit een persoon die wandelt van één kamer naar de andere, met zijn of haar slim toestel om de performantie van de geïmplementeerde oplossing te testen onder een realistischer scenario.

Acknowledgements

After finishing this Master thesis, I would like to express my enormous appreciation for my promoter, Prof. Dr. Beat Singer and my advisor Maxim Van de Wynckel. Thanks to their regular and always constructive feedback, I managed to realize this thesis in combination with a full time job as Application Developer.

In addition, I want to express my gratitude to all professors, assistants and lecturers of the VUB, who indirectly contributed to the realization of this thesis with their knowledge transfers.

I also thank my parents for their support during this Master study, both mentally and materially, and sometimes even from the content perspective.

Finally, I wish to mention that my studies as a Professional Bachelor in Applied Informatics have formed a good basis, but that this university study has enriched my knowledge both in breadth and depth. I now hope that these efforts make up the basis for an even more interesting professional career.

June 2021, Ronald Michiels

Contents

1

Introduction

In the technologically advanced and globally connected world of today, many people, companies and governments rely on Information Technology (IT) to operate and communicate [1]. Multiple technologies such as the modern computer, internet and smartphone did not exist as we know them prior to the 21st century, but are already integrated and crucial in the functioning of our society.

One impressive and arguably vital technology of our modern world is GPS, also known as the Global Positioning System [2]. GPS is a satellite-based radio-navigation technology that provides geolocation and time information. While originally made and deployed for military use, this technology quickly found its way into civil and commercial hands due to its usefulness.

Numerous companies and people rely on the technology to navigate the world safely and quickly without the hassle of studying a local map. All that is needed to take advantage of this positioning technology is some device capable of receiving GPS signals. Nowadays, users use their smartphone as

GPS for which handy apps are made like *Waze*[1] and *Google Maps*[2]. In short, the GPS system makes outdoor navigation reliable and easy.

Similarly to GPS, an *Indoor Positioning System* (IPS) is a positioning system for navigating an indoor or underground environment. The ability to indoor position can be extremely useful to many people, especially in large indoor or underground areas. Navigating hospitals, airports, malls or subways would be less daunting and easier to first time or infrequent visitors. Indoor positioning can potentially be utilized not only by humans, but also to navigate robots in a warehouse for instance.

Unlike GPS, no standard cost-effective IPS technique exists that reliably can be used for indoor navigation. GPS signals for example are not always available nor reliable for users inside buildings or underground structures [3]. The satellite signals are significantly degraded by various obstacles such as roofs and walls, so positioning indoors or building an IPS is more challenging [4]. In other words, the existing GPS infrastructure cannot be used in IPS.

However, multiple indoor positioning techniques exist, such as Wi-Fi based positioning (using Wi-Fi hotspots), Bluetooth, Infrared and others. *Visible Light Communication* (VLC) can also be used in an indoor positioning system. VLC uses visible light as medium to communicate positioning information to the user. Compared to the already mentioned IPS techniques, VLC does not use a special type of signal for its communication and can possibly make use of already existing infrastructure in a building (e.g. LED lights).

In this work we specifically focus on VLC using an image sensor or camera as receiver. One major advantage to using a camera as opposed to using a light sensor is the popularity of handheld smart devices such as smartphones. Nearly all smartphones are equipped with one or more cameras, including front-facing image sensors. In the current year, many people own, use and carry such smart devices on a daily basis. As a result, a VLC based positioning system that works using the camera of such smart device is more interesting since more people can potentially make use of it. A second advantage of the image sensor over a light sensor is its ability to capture context within a photograph of a light source. This context can be used in the positioning algorithm to increase accuracy or performance.

[1] https://www.waze.com
[2] https://www.google.com/maps

In this thesis we aim to investigate the VLC techniques and algorithms that exist in the literature and propose a basic VLC indoor positioning system of our own. The solution we present is based on approximately positioning the user utilizing unique identifiers coming from VLC enabled light sources in the environment. We discuss how we detect these light sources, perform error correction and decode the transmitted message. This system we propose is meant to be a working basis on which can be built and improved. The solution is highly scalable as additions can easily be made to improve positioning accuracy and performance.

2

Background Information

In this chapter we provide background information about Visible Light Communication (VLC). We provide a definition of VLC, associated technologies and available algorithms. On top of this, we go into detail about VLC data encoding, the types of receivers, multiplexing techniques and VLC-based positioning algorithms.

Visible Light Communication is a general term used to describe all kinds of communication that uses visible light, but is primarily used in the context of Indoor Positioning System (IPS). VLC-based IPS designs can provide localisation with the help of lamps acting as signal transmitters.

The visible light in VLC refers to light that is visible to human eyes. This is only a portion of the electromagnetic spectrum. Humans can see electromagnetic radiation with a wavelength between 380 nanometres to 740 nanometres [5] — which is called visible light, as illustrated in Figure 2.1.

A positioning system using Light Emitting Diodes (LEDs) as transmitters, rather than other light sources — as the LED lamp has multiple advan-

tages over more traditional lamp technologies. In addition, VLC allows to potentially re-use the existing LED lamp infrastructure of a building.

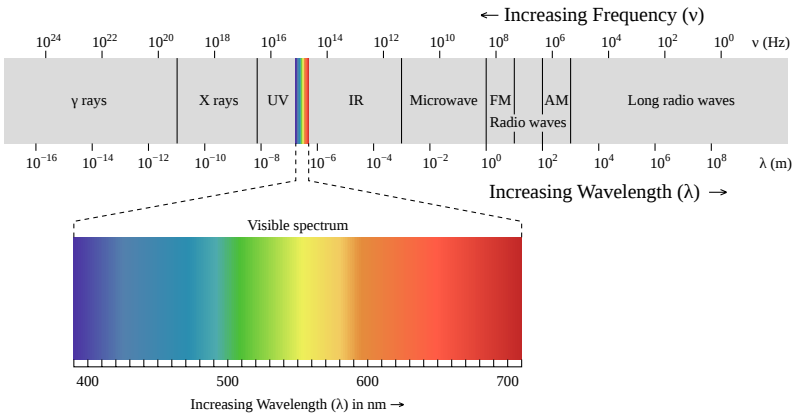

Figure 2.1: The electromagnetic spectrum [6]

A Light Emitting Diode [7] is a semi-conductor light source that emits light when current flows through it. In general, VLC uses LED (Light Emitting Diode) lights as a source. LED technology offers multiple advantages over other light source technologies, such as its ability to pulsate at high frequencies (above 200 Hz), better energy efficiency and longer life expectancy, cost effectiveness and security. LEDs do not cause side effects on the environment nor humans and can be used in restricted places such as hospitals and airplanes [8].

Generally, we assume the basic setup for a VLC application to be similar and contain some indoor environment in which visible light will be used for positioning. This environment is divided in so called *regions*. Each region is fitted with multiple LED lamps which serve the dual role of lighting and transmitter. The setup also contains the *receiver* — the device that will receive the transmitted information from the LED lamps and calculate its position in relation to the lamps based on the received data.

Some examples of real use cases in which our chosen positioning system can be used to navigate an indoor environment include [9]:

- Warehouse robot navigation: Robots can be used in a warehouse identify a tagged product, monitor assets or manage inventory. The robots operating in the warehouse could use VLC to navigate the workspace. Ceiling-mounted LED lamps would provide both light and information

for the positioning system. The robot itself will be fitted with a receiver, such as a Photodiode. With this basic setup, the robot can navigate the warehouse.

- Museum exhibit navigation (e.g. eduction or entertainment): Museum exhibitions can use a VLC based positioning system to help navigate its attendees through the exhibition. Throughout the museum, LED lamps have been installed that can be used in a visible light positioning system. The museum provides an app that can be installed on the users' smart device, using LED lamps for navigation. On top of this navigational feature, the app also determines what piece of art the user is currently looking at and provides the corresponding text and audio information.

- Hospital navigation (healthcare): Hospitals could leverage this technology primarily in navigation for patients or visitors throughout the hospital itself. Most hospitals are a maze where someone could get lost, or waste time finding the correct route to a certain department or room. A visible light-based positioning system could help alleviate this issue. VLC capable LED lamps, also used for lighting, can be used together with the users' smartphone in a hospital navigation app. The app can provide the correct route users should take to get to their destination within the hospital and track their position. This system could prevent confusion and potentially help elderly and visually impaired people better with extra audio cues during navigation.

- Store or shopping mall navigation (e.g. retail): Assuming that the store or shopping mall is equipped with the necessary VLC-capable LED lamps, users can use their smart devices to navigate through the shopping space. In addition, the shops in the mall can provide extra information or perks to the customers that walk through the space. One idea is proximity-based advertisements, such as special discounts or offers for users that are close to a certain product. Another use case is an in-store navigation to a certain product that the user is looking for. As example, if the user is searching for *product X*, the app will highlight or show a route where to find this product in the store.

The required accuracy of the VLC technique will vary from application to application. Depending on the application's requirements, more complex or simpler positioning algorithms can be utilised as will be explained later.

2.1 Data Encoding

The LED lamps, which serve as a transmitter in a VLC application, need to encode and transmit their own positional information (as where in the building the lamp is located) to the receiver. This encoding step cannot be done by an ordinary LED lamp and will require an extra component — such as a small processor — to encode the positioning data within the VLC signal. The technique that is used to generate this encoded signal is called *data modulation* [10, 11]. Data modulation techniques are used to convert digital data into optical signals. The receiver of the signal is assumed to have the necessary tools onboard to decode the received signal from the LED lamps.

We have explored several data modulation techniques: [9]. See also figure 2.2.

- On-Off Keying (OOK): Rapidly switching on and off LEDs to convey a signal.
- Intensity Modulation (IM): The luminous intensity of the source changes to convey a signal.
- Pulse Position Modulation (PPM): Data transferred in pulses of light in certain time intervals.

2.2 Receiver Types

Next to the transmitter, the receiver has a vital role within a VLC system. A key function (as already stated) is that the receiver must be able to decode the incoming encoded signal in the system — but this is not the whole story. In order to correctly determine the position of the receiver, plenty of additional useful data needs to be obtained, such as the received signal strength, the signal strength ratio, the time of arrival, other geometric information — such as the angle of arrival. This information might be required by the positioning algorithm of the VLC application, and must be captured by the receiver itself. The kind of information needed depends on the particular positioning algorithm used.

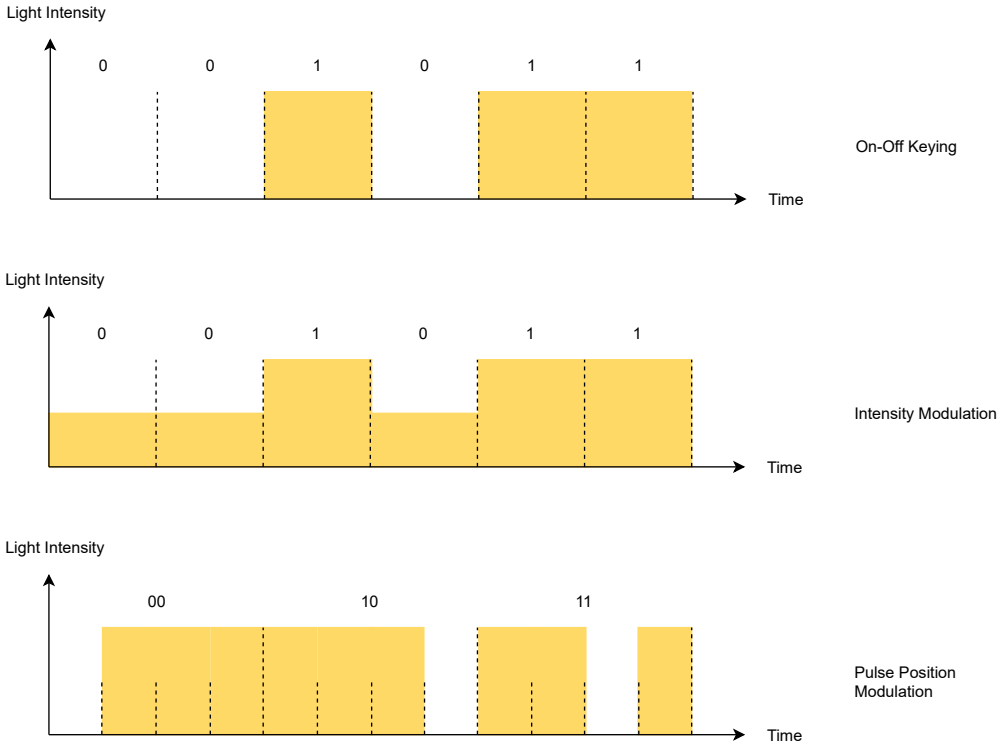

Figure 2.2: The discussed data modulation techniques

In short, we can distinguish three different types of receivers [12]: The Photodiode (PD), Image Sensor (IS) and Photodiode Array (PDA).

Photodiodes have already been widely used in visible light positioning systems, since they offer some advantages as opposed to the other receiver types. One such advantage is their relatively low price, as well as their ability to communicate at high data-rates with LED lamps. PDs can measure signal strength accurately due to their high dynamic ranges. On top of this, PDs have fast response times which means they can measure the time of signal arrival with a high degree of accuracy. This means PDs are suitable for positioning systems using the Time of Arrival (ToA), Time Difference of Arrival (TDoA) or Received Signal Strength (RSS) algorithms.

Image sensors can provide better information related to the position of the LED transmitters as opposed to the PD. Image sensors also have the ability to separate light sources and can deal with ambient light sources (such as sunlight) much better than PD. To retrieve the information received from the lamp's signal, a sequence of taken photographs is processed to find the

presence and absence of light sources. A disadvantage of the IS is that the achieved data rate is lower or equal to the camera frame rate. To achieve a better data rate, LED arrays can be used to transmit more information in a signal frame than a single lamp can.

Photodiode arrays can be used to obtain more information (as opposed to one PD) and at a higher data rate (compared to IS). A hybrid algorithm of fingerprinting, RSS, TOA, TDoA and Angle of Arrival (AOA) can be possible with a personal digital assistant.

2.3 Multiplexing Techniques

According to the book *Multiplexing*, "Multiplexing is the process of transmission of information from more than one source into a single signal over a shared medium" [13]. In the context of VLC-based positioning systems, multiplexing techniques can be used to transmit data when multiple LED base stations are close by and whose signals may otherwise interfere.

Multiple multiplexing techniques exist, as described in [12]. One such technique is time division multiplexing (TDM), a multiplexing technique where different LED lamps transmit their signals at different points in time. However, TDM requires the signals of the LED lamps to be synchronised, which is a disadvantage. TDM is generally easier to implement than other multiplexing techniques. Unfortunately, it takes a longer time for the receiver to receive enough signals required to calculate the position of the transmitter.

Frequency division multiplexing (FDM) modulates the light of different LED lamps to be transmitted at different frequencies, but introduces the complexity of de-multiplexing the signal. In general, FDM is regarded as having a greater positional accuracy than TDM [9]

Code Division Multiplexing is a more complex technique, but has been shown to be more robust in terms of being less susceptible to interference compared to FDM.

Colour division multiplexing (or wave length division multiplexing) uses different light colours for each LED lamp to transmit the signal. However, an issue in terms of interference of the signal with this technique, is the difficulty

to distinguish different light colours due to the white balance of the image sensor.

Finally, space division multiplexing is mostly used by VLC-based positioning systems that use an image sensor. The accuracy of the positioning can be affected by the optical quality of the lens of the receiver.

2.4 Positioning Algorithms

VLC-based positioning algorithms can be divided into five groups: proximity, fingerprinting, triangulation, vision analysis and hybrid algorithms (a combination of various techniques) as discussed in sections 2.4.1 through 2.4.5 respectively.

2.4.1 Proximity

Proximity is the simplest algorithm, but cannot give the absolute or relative position of the receiver as the name implies. Instead, it can only provide proximity location information on how close the receiver is to a transmitter. The proximity location of the mobile device is determined by using the signal of a single LED lamp. Since each LED lamp has its unique identifier (UID), the receiver can now retrieve this identifier with its associated location data in the application's database. This allows the receiver to approximately position itself in the environment.

2.4.2 Fingerprinting

Fingerprinting (or scene analysis) is a positioning technique that compares the receiver's live measured VLC data with the pre-recorded location-related VLC data stored in a database, and tries to find a close match. Due to the nature of taking measurements, the measured data will likely not match one-to-one with the pre-recorded information in the database, but the live and recorded data are matched as closely as possible. The fingerprinting algorithm can be divided in two phases: an online and an offline phase. During the offline phase, location data (such as signal strength) in the environment

is measured and collected to be stored in the database, while the online phase compares and attempts to match the live-recorded positioning data from the environment with the collected dataset. To match the data, multiple pattern matching techniques can be used as described below.

- **Probabilistic methods**: this method uses probability to locate the recorded location data. When we define $L_0, L_1, L_2, ...L_n$ as the LED lamp locations tested during the offline phase, and s as the online recorded signal strength, then we can calculate the probability $P(L_i|s)$ that the receiver is positioned under this LED lamp L_i, using the Bayes formula [14] $P(L_i|s) = \frac{P(s|L_i)P(L_i)}{P(s)}$. We determine the location of the receiver as the LED lamp for which $P(L_i|s)$ is highest.

- **K-Nearest Neighbours (k-NN)**: A relatively simple estimation algorithm. Assume $i = 1, 2, ...m$ and $j = 1, 2, ...n$. S_i is the online signal strength measured by the receiver. U_{ij} is the received signal strength from the i-th LED lamp measured at the j-th LED lamp during the offline phase. The distance D_j between the online measured signal strength vector $S = (S_1, S_2, S_3, ...S_m)$ and the offline measured signal strength vector measured at the j-th LED lamp $U_j = (U_{1j}, U_{2j}, U_{3j}, ...U_{mj})$ can be calculated using the formula $D_j = \sqrt{\sum_{i=1}^{m}(S_i - U_{ij})^2}, j = 1, 2, ...n$. After calculating the distances, the (x, y) coordinate of the receiver is calculated using k LED lamps which have the smallest distances using the formula $(x, y) = \frac{1}{k}\sum_{j=1}^{k}(x_{L_j}, y_{L_j})$ with L_j being the LED lamp at position j [14].

- **Correlation**: This technique uses the observation that the correlation of the mixed received signals from all LED lamps and the transmitted signal of a specific LED lamp is higher when the receiver is closer to that specific LED lamp.

- **Signal features**: The accuracy of a fingerprinting algorithm depends on the type of signal features used. The most basic signal feature we already discussed is signal strength from a LED lamp, as well as multiple signal strengths from multiple LED lamps. Fingerprinting can potentially also use different signal features, such as the impulse response time of LED lamps, a series of received signals in a period of time and others.

2.4.3 Triangulation

Triangulation aims to provide absolute positioning of the receiver using the geometric properties of triangles. In general, two different derivations exist - lateration and angulation. Lateration estimates the position of the receiver based on the measured distance from LED lamp source to receiver. Lateration techniques include time of arrival (TOA), time difference of arrival and received signal strength (RSS). The latter derivation, angulation, uses the measured angles of multiple LED lamps to find the receiver's position. The used angulation technique is angle of arrival (AOA)[12].

- **Time of arrival (TOA)**: TOA estimates the distance from the receiver to the LED lamp using the arrival time of the signal. The distance is calculated multiplying the propagation delay of the signal with the speed of light. To estimate the distance, signals of at least three different LED lamps must be captured.
- **Time difference of arrival (TDoA)**: TDoA is similar to TOA, but works with the calculated difference in time of a specific received signal, detected by 2 separate receivers — which have precisely synchronised time references. TDoA-based designs determine the phase delay information from the received signals.
- **Received signal strength (RSS)**: RSS is the measured optical signal strength by the receiver. RSS is assumed to be constant, and utilises direct detection (DD), a type of non coherent detection that determines the presence or absence of energy without recovering phase information. The receiver position can be estimated by using the intensities (power) of received signals together with trilateration.
- **Received signal strength ratio (RSSR)**: RSSR is similar to RSS, but instead of directly translating the received optical power to distance, RSSR translates the received power from multiple LED lamps to the ratio of distance, determining the receiver's position based on the distance ratio.
- **Angle of arrival (AOA)**: AOA is the direction (angle) from which the signal is received. AOA exploits the fact that the received power is also a function of the angles φ_i and θ_i in the picture. These angles are defined by the location coordinates of the LEDs and the receiver, which can be determined using trigonometric relations. A figure illustrating the setup can be found below 2.3.

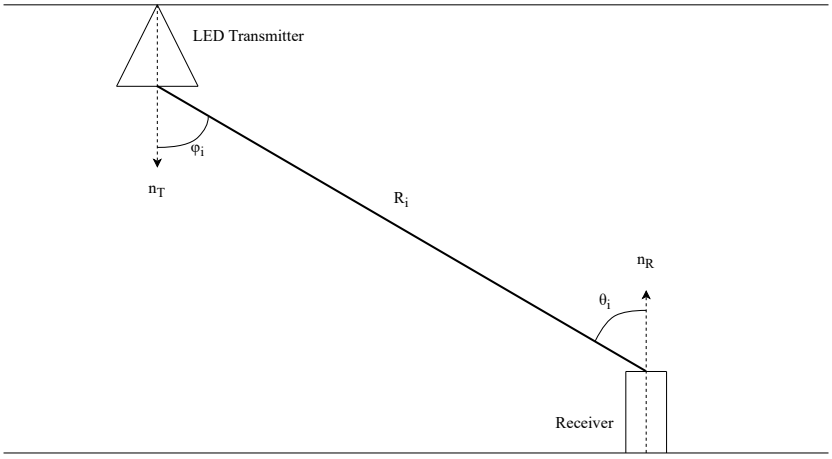

Figure 2.3: Depiction of the angle between transmitter and receiver [15]

2.4.4 Vision Analysis

Vision analysis considers the geometric relationships between 3D objects in reality with their 2D positions in the projections on an image sensor. The geometric relations are derived from the pinhole camera model. In this model, three coordinate systems exist: the 3D world coordinate system, the 3D camera coordinate system and the 2D image coordinate system as illustrated in Figure 2.4.

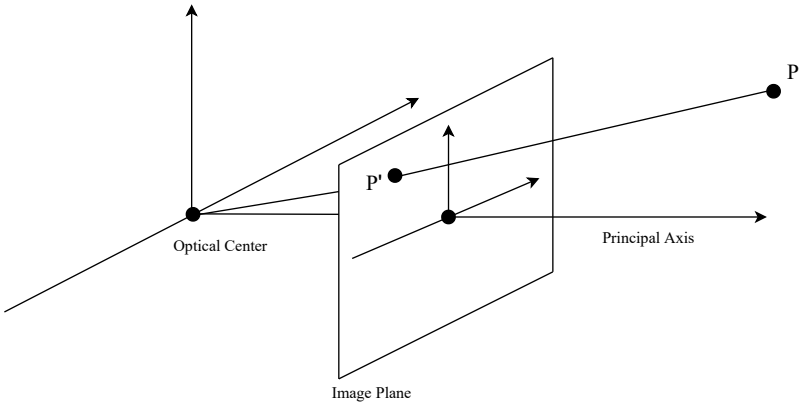

Figure 2.4: The pinhole camera model [16]

Suppose P is a point in 3D space, with a coordinate (x', y', z') in the camera coordinate system and (X, Y, Z) in the 3D world coordinate system. (X, Y, Z) can be transformed to (x', y', z') using the matrix [17]

$(x', y', z', 1)^T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} (X, Y, Z, 1)^T$ with R being a (3 x 3) rotation matrix and t a (3 x 1) translation vector and 0 = (0, 0, 0).

In the pinhole camera model, shown in figure 2.4, if the point P' is a projection of the point P with 2D coordinates (x, y), then

$(x, y, 1)^T = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} (x', y', z', 1)^T$ with f the focal length of the lens.

The geometric relationship between the coordinates of the 3D and 2D world of a point P can thus be described as [17]:

$$(x, y, 1)^T = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} (X, Y, Z, 1)^T$$

VLC-based positioning systems using image sensors can typically easily retrieve the reference information, since the LED lamps are assumed to be always available and their positions known. The 2D image coordinate can also be obtained using image processing, possibly helped through LED fast blinking. The 3D position of the LED lamps themselves can be obtained through the light signal.

Next to the basic vision analysis, two other techniques are discussed:

- **Single view geometry**: This method depicted in Figure 2.5 uses one camera to take the image of multiple LED lamps. Next, the geometric relationship between the 3D world coordinates and the 2D image coordinates are used to derive the position of the camera. The 3D world coordinates are typically retrieved with the information sent by the LED lamps and the 2D image coordinates are determined using image processing.
- **Vision Triangulation**: A method to find the 3D position of an object in the world using multiple cameras, as illustrated in Figure 2.6.

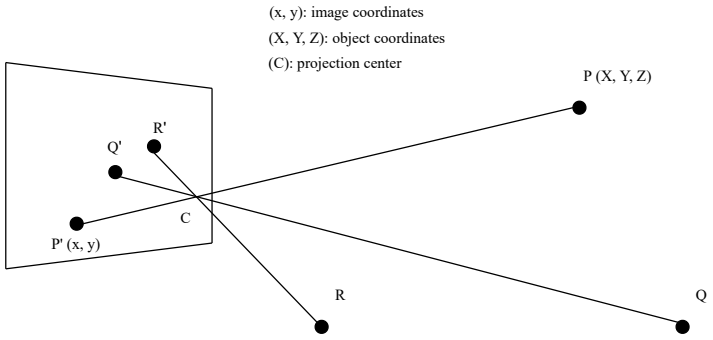

Figure 2.5: Single view geometry [18]

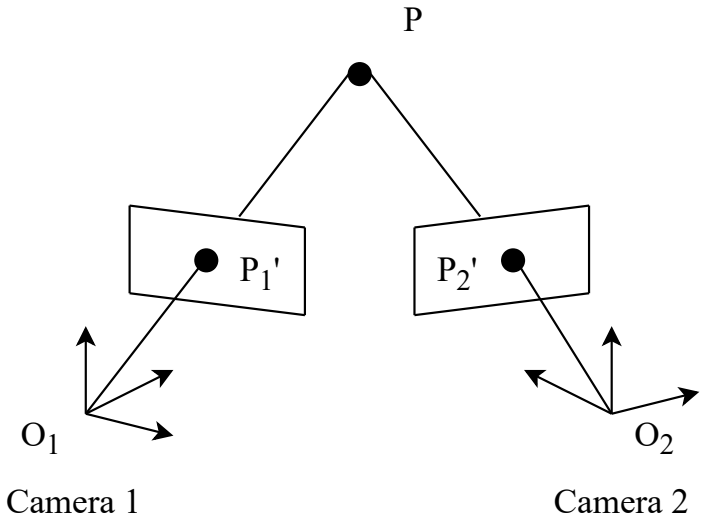

Figure 2.6: Two camera view model [19]

2.4.5 Hybrid Algorithms

Hybrid algorithms use multiple positioning algorithms together. We explore several positioning algorithms to improve a VLC technique. This combined approach has the potential added benefit of leveraging the advantages of two or more techniques together that complement each other, resulting in an overall system that is more robust and reliable compared to a system sticking with only one (type of) algorithm or VLC technique.

Various hybrid algorithm combinations are possible. To give an idea on how such a hybrid solution might work, we list some examples below.

- Combining the RSS and AOA algorithms together: These two trilateration methods can be combined to achieve a better positioning accuracy. In this example, we have the standard setup of an environment with VLC-capable LED lamps and a mobile PD receiver. During the first step of the hybrid algorithm, the rough position of the mobile device is determined using RSS. In the second step, AOA is used to determine the angle between the receiver and the LED lamp transmitter, utilizing the ratio between the RSS measured received power and the theoretical received power. The theoretical power is the maximum power that the device can receive from the LED lamps, just as if the receiver is placed directly underneath the LED lamp. Using the angle from the AOA algorithm, the distance of the receiver to the LED lamp is calculated. In a third step, a more exact position can be determined using quadratic programming. See paper [20] in which this technique is used.

- Combining proximity and RSS algorithms together: Assuming we use the standard VLC setup, being VLC capable LED lamps and a receiver, the hybrid algorithm works by first determining the general position of the receiver based on the received signal IDs using proximity. Next, the RSS algorithm can be used to find the more exact position of the device. To do so, the minimum mean square error (MMSE) algorithm can be used in conjunction with RSS to limit calculation work, since the possible position of the receiver is limited to the (proximity) area found in the first step as described in [21].

- Combining RSS and TDoA algorithms: Using the standard VLC setup once more, this hybrid technique utilizes RSS to determine the rough distance of the receiver from multiple LED lamps. Next, the difference in distance based on the RSS measurement is calculated. The TDoA is calculated as well using the local highest peaks of the received signal. The distance difference between TDoA measurements is calculated. The final distance differences are calculated based on the RSS differences and TDoA differences, using weighted sums. To obtain the actual position, the Newton method can be used. This technique is used in paper [22].

- Combining VLC and Dead Reckoning in an indoor tracking system: The setup contains a mobile device that we seek to position. In a first step, a VLC positioning technique is used to determine the position. Next, Dead reckoning is used in combination with a Kalman Filter and gyroscopic data from the mobile device itself. Dead Reckoning is a method for calculating the position of an object (mobile device) using

the last known location of the object, in combination with its estimated speed and direction. the speed and direction information can be derived from the onboard gyroscope and accelerometer hardware of the device. The Kalman Filter (also known as Linear Quadratic Estimation) is an algorithm that can be used to perform the Dead Reckoning with the gathered information in order to more accurately estimate the exact position of mobile device [23].

3

Related Work

Visible Light Communication is a technique that is already established in both academic research and the industry. In this chapter, we focus on literary work related to the subject and research of this thesis.

3.1 VLC-based IPS With Image Sensor

The related paper *VLC-based Positioning System for an Indoor Environment Using an image sensor and an Accelerometer Sensor* [24] proposes a visible light indoor positioning system using a handheld device, such as a smartphone, equipped with a camera (image sensor) and accelerometer.

The authors of this paper chose to use an image sensor, since most handheld devices are equipped with cameras rather than photo diodes. In addition, image sensors have the added benefit of not requiring multiplexing techniques since the camera can spatially separate light sources.

In addition, the paper notes how image sensor based positioning algorithms exist in various complexity. One difficulty with the Image Sensor is calculating the angle at which the camera receives the communicating target LED light, which makes the algorithm to determine the receiver's position much more complex.

To overcome the complexity and limitations of existing algorithms, the authors opted for a two-step approach or technique to determine the position of the handheld device in a VLC enabled environment. Their proposed algorithm determines the distance from LED to receiver using an image sensor in the first step, while utilizing the calculated distances in a set of equations that are solved to determine the position of the device. Solving the equations is done in conjunction with information gathered from the accelerometer sensor which is part of the device and provides useful information about angle or tilt of the receiver at that moment.

One additional difficulty when dealing with image sensors is sensor noise. Most algorithms opt to ignore this phenomenon, but the paper presents a technique to reduce this image sensor noise in order to improve accuracy of the system.

The proposed environment for this system to work in, consists of ceiling mounted LED panels and an image sensor receiver, such as one found on a smartphone. The ceiling lamp consists of multiple LEDs in array configuration, with a minimum of four lights per panel. One of four lights in the array are chosen to be transmitters of their position. The others are used for illumination purposes.

These LED lamps have a pre-defined known position in 3D space. This 3D position is transmitted by the lamp in its environment, and picked up by the image sensor which demodulates the signal. The IS can detect the LED lights without issues when in frame.

An image sensor is basically a matrix that consists of photosensitive elements, in which each element (pixel) acts like a photo sensor. Therefore, it can detect and demodulate multiple light signals at once during operation, such as the four selected LED lights in our environment.

The used algorithm assumes that the image sensor is facing the LED lamp directly, not taking into account angle or tilt. This makes the algorithm significantly easier, faster and more accurate. The angle or tilt will be measured

later using the accelerometer however, since it is a vital piece of information in the calculation.

To conclude, the authors tested the functionality and performance of their proposed VLC-based positioning system using simulations in Matlab. This allowed them to more easily manipulate the environment in which the tests were conducted, but they did not test the system with hardware yet.

3.2 IPS Using Camera

A second related paper, *High-resolution indoor positioning using light emitting diode visible light and camera image sensor* [25], presents a similar VLC based positioning system compared to the previous work. This paper focuses on the usage of an image sensor (or camera) for indoor positioning as well.

This paper investigates the viability of a VLC-based indoor positioning system using an image sensor or camera. The handheld device of choice is a smartphone with front-mounted camera, which is used as image sensor and receiver.

The authors in this paper opted to use two LED lights for positioning of the smartphone, mounted on the ceiling in test-environment. The positioning consists of 2 steps: retrieving LED coordinates using LED identification and position approximation of the smartphone with image processing and the extracted coordinates.

First, the camera captures an image under the LED lights. Using captured images, the authors retrieve the LED UID. A lookup on a database for the exact LED location is executed using the retrieved UID.

During this process, the algorithm determines coordinates of the camera (image sensor) using only the captured image. Next, the approximated coordinates of the camera and LEDs are used together in the next positioning calculation.

In order to perform an accurate calculation, the tilt and rotation of the device is taken into account. Tilt and rotation can be determined by the use of the accelerometer that is built in practically every commercially available smartphone. Tilt and rotation values are used as compensation for image

processing to more accurately position the image sensor relative to the LED-lights. Compensation is only possible when both LEDs are captured by the image, which typically translates to a rotation of less than 20 degrees.

After this final calculation, the end result is the best approximation of the smartphones position in the 3D space. Important to note is that this method requires the LED lamp positions to be known beforehand.

The authors in this paper tested their method using a regular Samsung smartphone and two LED lights in a controlled environment. The LED lights are mounted on the test-environment's ceiling and the smartphone is placed below the lights. Images are captured using the smartphone in different positions, rotations and pitches.

The paper concludes by stating that accurate indoor positioning using VLC is possible using their proposed technique, achieving an accuracy error of only a few centimetres.

However, it is not mentioned where exactly the calculations have taken place during the experiment, on the smartphone itself or a different device entirely. Execution time of the algorithm from image capture to final position estimation is also not mentioned.

3.3 Smartphone-based VLC

The paper *Smartphone Camera Based Visible Light Communication* [26] explores the feasibility of using a smartphone as both transmitter and receiver in a VLC system.

Contrary to the standard approach of using multiple LED lights as medium to transmit positional information using VLC to a receiver such as a smartphone, the authors of this paper focus on a different method of transmitting information using VLC.

This work investigates the use of a screen as transmitter instead of LED lights for example, while the receiver is a regular smartphone. The transmitter can be any screen, such as the display of another smartphone. The SURF (Speeded up robust features) algorithm is used to detect information on

captured images. In order to deal with image distortion and other undesired effects, projective transformations are applied. See figure 3.1.

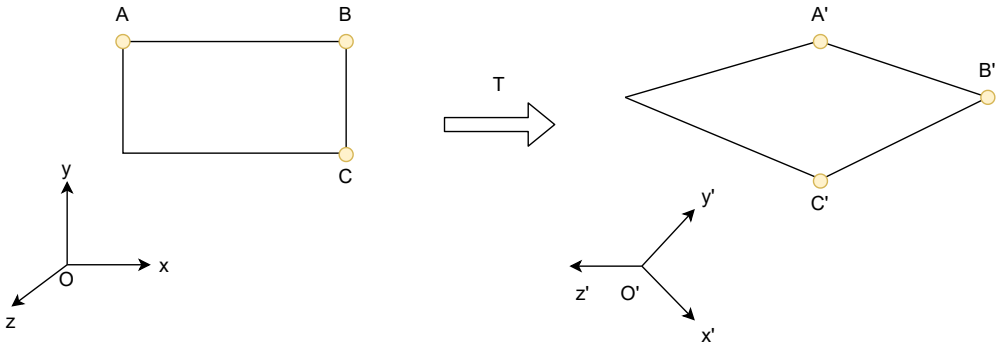

Figure 3.1: Projective Transformation

The transmitter (display or screen) receives the digital data to transmit. The display translates this data into an image, using binary stream conversions. Next, the pre-arranged detection image is shown by the transmitter. This image signals to the receiver that useful data transmission is about to commence. After showing this detection frame, successive data frames are shown until the required data is transmitted.

The receiver (camera) detects the pre-defined detection frame, and starts the capture of following data frames transmitted by the transmitter. The SURF algorithm is executed on both detection frames: the frame captured by the receiver, and the original detection frame known by both the transmitter and receiver. The algorithm detects, extracts and calculates key-points plus their descriptors on both frames. The result of both frame's analysis are compared by determining the correlation between both. Depending on the result, the algorithm determines if sufficient key-points are found, in which case data recovery can commence. If not, the process is stopped since the captured detection frame was not satisfactory to continue data recovery. The SURF algorithm is especially useful to determine whether the loss, due to environment noise such as ambient light and other sources, is acceptable or not.

In the next stage during data recovery, geometric transformation calculations are executed in order to counteract possible warping and distortions in the received or captured image versus the original transmitted data frame. This is done using the key-points from original pre-defined and captured detection frames to compute the projective transformation matrix, which is used to correct all received distorted images.

Binarisation is used to clean up the received image and reduce inter-symbol interference. The last step in the algorithm uses quantisation to reconstruct broken or incomplete cells received by the captured image. The receiver will scan through the frame, in order to reconstruct the cells using the following logic: if 50 percent or more of the cell is black, the entire cell is considered and coloured black. If on the contrary, less than 50 percent of the cell is black then the entire cell is considered and converted into white. After this step, the image processing is completed and the result can be compared to the originally transmitted data frame.

A potential benefit of this method is the high data transmission rate: when using black and white images, a data rate of around 112 kilobit per second can be achieved, which is substantially higher than what is possible with regular LED arrays. This figure can be tripled to about 300 kilobit per second transfer rate when using colour images.

This technique has two downsides: using dedicated screens or even entire smartphones as transmitter is much more expensive than using LED lights. In addition, the authors described the optimal distance of this method to be from 10 to 50 centimetres, which is substantially shorter than the traditional LED light approach.

3.4 VLC-improved Positioning Algorithm

The related work *Image Sensor Based Visible Light Positioning System With Improved Positioning Algorithm* [27] proposes a new positioning algorithm for VLC-based indoor positioning using smartphones.

The paper investigates a positioning algorithm, aimed to be faster and less error prone compared to the more widely used positioning algorithm: Levenberg-Marquardt (LM) [28]. However, this method is sensitive to initial guesses which results in longer response times of the positioning system which the new algorithm, named singular value decomposition (SVD) algorithm, aims to avoid.

The setup for the VLC-based indoor positioning system using the SVD algorithm is rather conventional to the other setups: the researchers use three LED lights as transmitters and a smartphone image sensor (camera) as receiver.

Each LED light in this setup has been assigned a unique LED ID, related to its physical mounting location and stored in a database. The LED light broadcasts its ID repeatedly using light intensity modulation, which is picked up by the image sensor of the receiver and used to demodulate the transmitted signal. The smartphones front camera is used for this purpose, and captures multiple images of the ceiling-mounted LED lights to determine both their IDs and the centre of the LED lights. Based on this information, the receiver's location and orientation can be calculated.

This system uses the pinhole camera projection model where the three LED lights are projected on the image sensor's image plane. In this system, the LED lights have 3D coordinates of the global coordinate system (GCS), while also being perceived by the image sensor (through pinhole camera projection) in the receiver coordinate system (RCS). The RCS is actually a 2D coordinate system, with the third dimension being the focal length of the camera lens, provided it is correctly focused.

The transformation from 3D LED light centre point in GCS to the corresponding image sensor centre point in RCS can be described by a combination of scaling, rotation and translation operations. The paper describes how we need to calculate scaling factor K, rotation matrix R and translation θ. This results in an equation with 6 unknowns, α, β and γ in the rotation matrix R and θ_x, θ_y and θ_z in θ which is why 3 LED lights are needed (at a minimum) to solve this problem.

The unknowns that we just described are found using the new proposed SVD-based positioning algorithm. This algorithm itself is quite technical, complex and requires a lot of matrix calculations. this can be found in the paper.

The authors of the paper tested their algorithm experimentally using three LED lights, a smartphone and laptop as server. They compared the performance of the SVD-based positioning algorithm against the LM-based positioning algorithm and ran both algorithm tests ten thousand times.

The paper concludes stating that they experimentally found their SVD-based positioning algorithm to be fifty to eighty times faster than the LM-based positioning algorithm. The average positioning errors are 6 centimetres and 14 centimetres for a test distance of 1.2 metres and 1.6 metres respectively.

Due to the speed of this algorithm, the response time of the entire solution is much improved compared to a more traditional LM-based positioning technique.

3.5 VLC Time of Arrival-based IPS With Smartphone

The paper *Improved Smartphone-Based Indoor Pedestrian Dead Reckoning Assisted by VisibleLight Positioning* [29] proposes a combination of pedestrian dead reckoning and visible light communication as indoor positioning system.

The described technique utilizes pedestrian dead reckoning (PDR) and visible light communication (VLC) techniques to create a low-cost, yet accurate indoor positioning system powered by LED lamps and the user's smartphone. The idea is to use VLC in order to periodically correct the position estimated by PDR, for which the positioning errors can accumulate a lot.

Pedestrian dead reckoning is used to calculate the step size and moving direction of the pedestrian (or user) using a PDR-enabled device such as the modern smartphone with built-in internal measurement unit (IMU sensor). The PDR algorithm uses the sensors in the smartphone to calculate steps taken by the user based on the data coming from the acceleration sensor in the device. Using the human motion step model, PDR can calculate the estimated distance travelled by the pedestrian using the device. PDR is powerful, in that it does not require any extra external devices, it has no extra costs, is autonomous and has relatively high short-term positioning accuracy. The huge downside of PDR, is that it suffers from accumulative error which results in huge errors for larger distances travelled.

The authors of the paper propose one improvement when using PDR to improve accuracy over distance in an attempt to limit accumulated errors, namely by creating heading angle groups in the collected accelerometer data. The angles of acceleration measured by the accelerometer can vary quite a bit at any given measured moment with outliers. In order to limit influence of the outliers, the heading angle group groups the data from small to large angles according to the described formula in the paper. For each created data-group, the averages are calculated. The average heading angle of the

best group is obtained and used in the PDR calculation, which gives superior results to random and mean angles.

Visible light communication or VLC is used to communicate the positions of pre-determined LED lamps in the environment. the LED lamps transmit their encoded position using modulation. The smartphone is utilized as receiver, using its camera to capture images of the transmitting LED lamps and decode the transmitted messages. The authors note that most smartphone image sensors work in roller shutting mode, which means each line of pixels is exposed to light at different time intervals, meaning the first and last row of pixels on the image sensor when taking a picture are exposed at different times. This property is used during VLC: LED lights have the ability to turn on and off at high frequencies, which can be captured on the image form the receiver (smartphone) itself. When the LED is turned on, the picture will contain one or multiple lines of light-coloured pixels. When the LED is turned off, the next lines of the image will be darker. This tendency will continue until the picture is finished being captured. This method uses the high line sampling rate to achieve signal transmission and reception. Due to the high blinking rate of the LEDs, human eyes cannot pick up on it.

To extract the information from the captured image, image processing techniques are used such as converting the image to greyscale and binarization according to a certain threshold. This allows the technique to determine whether the transmitted bit is a 1 or 0 (white or black). To improve accuracy further, a cyclic redundancy check (CRC) is executed on the decoded message. If the check fails, the threshold will be adjusted and the image is decoded again. If the check succeeds, a range constraint verification is performed. Based on the estimated step size of PDR, the range of LED lights that the user can reach is calculated. If the decoded message comes from a LED light in this range, the decoding is considered correct and absolute. If not, the decoded message is incorrect and the threshold is re-adjusted, the image analysis is re-executed and CRC performed again.

The researchers of the paper tested the solution experimentally using a test-trail of 94 metres long, with LED lamps mounted at the ceiling 8 metres apart. A Samsung Galaxy S7 smartphone was used for the experiment. The described method was compared to regular PDR, as well as a Wi-Fi based positioning system.

To conclude, the authors note that their proposed indoor positioning system greatly outperforms regular PDR in accuracy, while beating the Wi-Fi based

IPS system as well. Important to note is that the response time is similar to PDR.

4

Solution and Implementation

In this chapter, we will discuss our proposed VLC-enabled indoor positioning system. The system consists of a smartphone or smart-device with a (front facing) camera. In this section, we will regularly refer to our presented solution as *application* or *app*. Figure 4.1 below illustrates the setup we use.

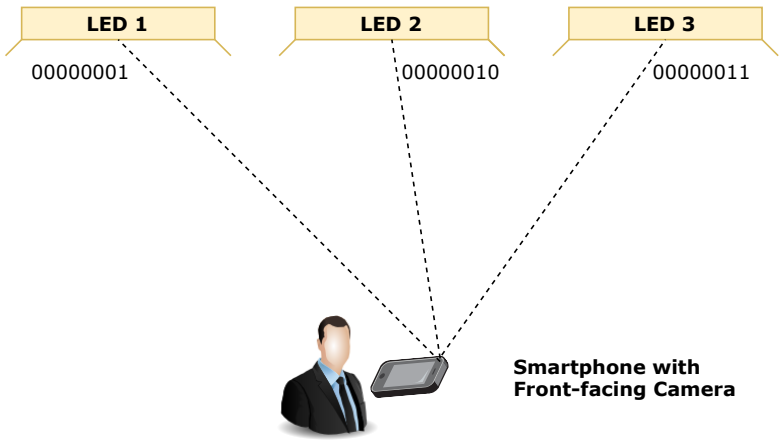

Figure 4.1: Illustration of the proposed positioning system

The idea of this setup is as follows. The user enters an unfamiliar indoor environment with their smartphone (or smart-device) and wishes the ability to position themselves in the building. With the proposed application operational, the smartphone activates its front-facing camera to have the ability to capture an image feed from the camera sensor. Assuming the user holds their smartphone in an ordinary fashion, the front-camera of said smart-device faces the ceiling. As depicted in 4.1, the ceiling contains multiple LED lamps (in our example 3 lamps). Since the front-camera is facing the ceiling on which the LED lamps are mounted, the camera captures the LED lights on video feed. The application is able to detect the lights or LED lamps in each image of the video feed from the camera using computer vision.

The LED lamps are able to communicate their respective identifiers using OOK or ON-OFF Keying described in Section 2.1. This means that the lamps quickly switch ON and OFF to transmit a binary message. In the context of this method, light ON equals a binary 1 while light OFF is 0. For example, if the light is ON, then OFF and then ON again, the lamp effectively transmitted `101` in binary which translates to 5 in decimal. In practice however, the size of the message or the amount of bits that will be transmitted is pre-defined and known by both the transmitter and receiver.

Our proposed solution is capable of recognising whether a LED lamp is ON or OFF on the given input image from the camera. Using this information and the history of previous images provided by the camera, the transmitted binary value 0 (light OFF) or 1 (light ON) can be determined and effectively the transmitted message can be decoded. Using the lamp's identifier, the application already has an estimation where in the indoor space it is located since the ID can be cross referenced with a central database containing the exact position of the lamp with that ID.

We start by explaining the light source detection in Section 4.2 where we use computer vision algorithms to extract the brightest source in the video. Next, in Section 4.3 we explain how light from this detected source is encoded and decoded.

The first phase of our practical implementation of the solution primarily focuses on capturing the information of a single LED lamp. This approach makes the development process more straight forward and testing less ambiguous.

4.1 Requirements

4.1.1 Functional Requirements

In this section, we discuss the functional requirements of the solution that we want to build. These requirements should be met by our application.

Our application should be able to take a video in the mp4 format with any frame rate and resolution above 480 x 270 pixels as input. The frame rate should not matter as every frame will be analysed individually. Resolution is also not very important as we are not interested in detail, but rather bright light signatures in the image.

The solution should have the ability to detect the light sources on a given image. This ability is vital, as without light detection we cannot receive and decode messages. It should be possible for the app to find these light sources in regularly lit rooms without problems, such as a room or office space with the lights turned on.

The application needs the ability to retrieve binary messages from an array of consecutive images. The logic is required to be able to retrieve the sent message by one or multiple transmitters in the video. After retrieving the binary message that was sent, the app should also be able to decode the sent data and convert it to a readable decimal number, which is the unique identifier of the light source.

If the solution receives an incomplete or faulty transmission in the video, it should not get stuck but instead reset itself and be able to continue decoding any correct transmissions that may follow.

It should be possible to decode multiple transmissions captured sequentially by one video file. It is however not required for the application to be able to decode multiple transmissions in parallel. For example, if two or more transmitters are captured in the same frame and simultaneously transmit a different message, the app is not required to be able to decode all of these transmissions at the same time.

After successfully decoding captured transmissions from the input video, the application should be able to output its results in a visible manner to the user, such as printing its findings to a console or file. The output should

contain the original received message as well as the decoding. If multiple messages were decoded, all of them should be presented.

4.1.2 Non-Functional Requirements

In this section, we discuss the non-functional requirements of the solution. These requirements are a bit more vague, and are nice-to-have rather than required.

The application should be responsive and have a limited processing time. Analysing the input video file is fast and the application stays responsive during this time.

The built solution should visually show what is happening to the user. Preferably, the current light source detection is drawn on the current video frame while processing, as well as the decoded messages to inform the user.

The application should be able to reliably decode messages correctly captured by a video file from 3 meters or less away.

4.2 Light Source Detection

Our first version of the application is built with Python using the *OpenCV*[1] framework. OpenCV is an open source computer vision library and defines multiple image processing procedures that are used in the application. On top of this, OpenCV makes it easier to manipulate images.

The first functionality of the app that we tackled is the ability to detect light sources on an input image. This light detection ability is useful since capturing and later decoding a transmission is based on the presence or absence of a distinct light source. In order to do this, the following steps are executed. For the code, please see appendix B.

The algorithm has a built-in *FrameObjectHistory* A. This is a custom data structure that keeps a certain amount of already processed frames (FrameObjects A) in its buffer and metadata as history. These buffered images are used

[1] https://opencv.org

during the algorithm to compare the current input frame with the history of previous frames. Using the result of this comparison, the algorithm can better determine whether and where a light is visible in the image.

When the app receives an image from the video feed, the image is first resized to 480 by 270 pixels, as this resolution is sufficient for detecting light sources. Next, this resized image is converted to greyscale using a conversion procedure from OpenCV. This is an important step as the greyscale image is much easier and more performant to work with than the original colour image. After converting the image to greyscale, a procedure performing Gaussian blur is executed on the greyscale image. This blurs the image using a Gaussian filter, in order to remove potential image noise. This technique is often used in image processing.

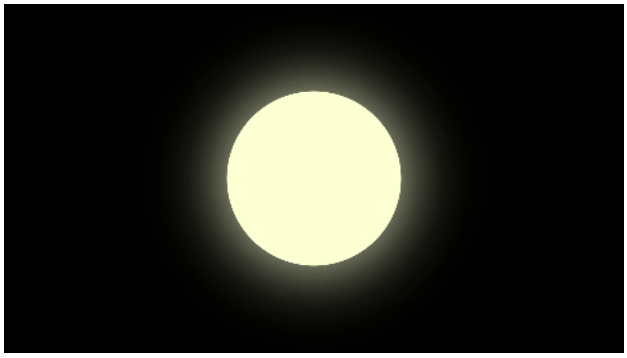

Figure 4.2: The original input frame

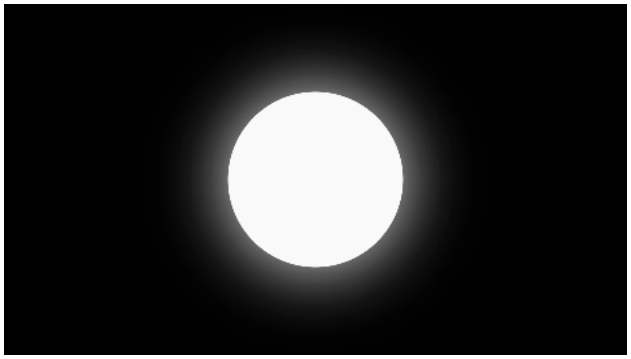

Figure 4.3: The greyscale version of the input frame

After applying the Gaussian blur on the greyscale image, OpenCV is used in order to find the darkest and whitest pixels as well as their positions on the image. This information is important later for next iterations of the algorithm.

Figure 4.4: The Gaussian blurred greyscale frame

In order to determine if a light is visible on the Gaussian blurred image, we make use of a thresholding function provided by OpenCV. This procedure takes a greyscale image together with a certain threshold value and returns a new greyscale image. The returned greyscale image has a fixed-level thresholding applied compared to the input image. This means that the thresholding function checks every pixel in the image and compares it to the given threshold value. If the current pixel's value is smaller/equal than/to the threshold value, the pixel is set to absolute black (value 0). Otherwise, the pixel is set to absolute white (value 255). This effectively creates a bitonal image where every pixel is either absolute black (0) or white (255).

The described operation results in a new image, in which any light sources should be coloured white (value 255) and dimmer (non-light source) objects should be black (value 0). The effectiveness of this procedure is dependent on the chosen threshold value. We choose said value as described below.

To determine the threshold value, the *SMA* or *Simple Moving Average* of x (for example 10), historic images are calculated using the before-mentioned FrameObjectHistory object. Since every processed frame is also stored in this history, it can be used in the next iteration. The calculated SMA is basically the average maximal pixel brightness value of the last x frames in the history, which is determined using *minMaxLoc* function for each frame. The reason to use this historic buffer is to smooth out any anomalies that can happen in the video feed. With x chosen large enough, bad input frames will have less of an impact on the detection rate of the app.

After calculating the SMA, a margin of y percent (for example 0.15) is applied to said SMA in order to allow variance in the perceived brightness of the input frames captured by the mobile device's front camera. With the threshold

value determined, which is $thresholdValue = sma * y$ where $0 < y < 1$, the result image of the threshold value is calculated.

The newly generated threshold image might contain some inconsistencies after these operations. To clean up the image, we run erode and dilate procedures on the threshold image. Erosion removes small-scale details and artefacts from the image, while dilation expands the shapes in the image. Running these processes in succession creates the effect in which speckles/small artefacts are removed from the image, while the image structure remains intact.

With the image cleaned, we can now use OpenCV in order to detect shapes on the generated bitonal frame. OpenCV is capable of finding countours given a bitonal image, and provides a list of found shapes on the image. Effectively, we interpret this list as a series of potential light sources found in the given frame.

In order to measure these potential light sources, we use OpenCV to draw the *minimal closing circle*. This is the smallest circle that fits around a given contours or shapes found in the previous step and returns its coordinates and radius on the image. We record this date in the FrameObjectHistory for later use, but this information can also be used in order to draw the detected light sources on the input image if there are any.

4.3 Data Encoding and Decoding

The application now has the capability to detect light sources on a given image or input stream. In order to add the capability to encode, send, receive and decode messages captured from VLC-enabled light sources, we use the following system.

As mentioned before, the VLC-enabled light sources or transmitters are responsible for transmitting their *UID* or *Unique Identifier*. The UID is an integer value corresponding to the unique position of the transmitter, configured beforehand.

Each light source sends its UID using the aforementioned OOK or On-Off-Keying method. However, the UID cannot be sent in its decimal form. In order to transmit the ID, the identifier is converted from decimal to binary,

a code which can be sent using OOK as a one (1) is encoded by light and a zero (0) is transmitted by shutting off the light momentarily for a short period of time.

Converting the decimal UID value to binary is a naïve approach however: most UID codes will consist of many successive one (1) or zero (0) signals. This can cause issues as identical successive symbols are confusing and will result in synchronization issues. For example, how do you distinguish a signal of 5 one (1) symbols 11111, which is 31 in decimal? Without any further information, the best way to count the symbols is correct synchronization between transmitter and receiver. This requires perfect synchronisation however and is quite error prone.

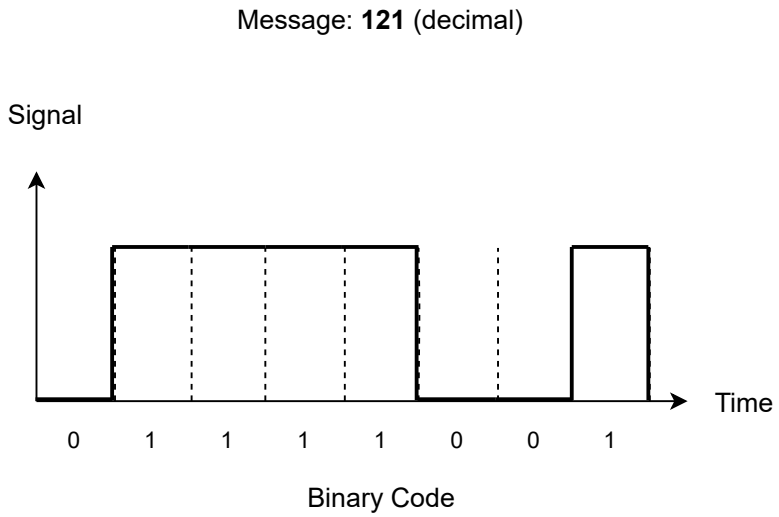

Figure 4.5: Decimal to binary conversion

4.3.1 Error Correcting Codes

To solve the issues with potential transmission errors, we use a linear error correction code (ECC). An ECC is used to control errors that might appear when data is sent over noisy communication channels. In order to combat errors that might slip in the data sent over the channel due to noise, which might appear anywhere in the transmitted message, redundancy is used. The transmitter of the code builds in redundancy that can be used to recover the message contains error(s) at reception. ECC allows to recover a sent message that suffers from an error caused by noise without having to retransmit.

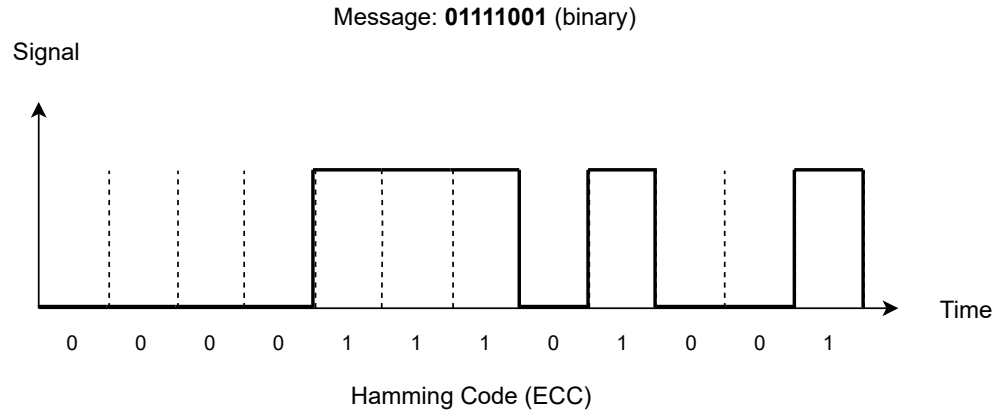

Figure 4.6: Binary to Hamming conversion

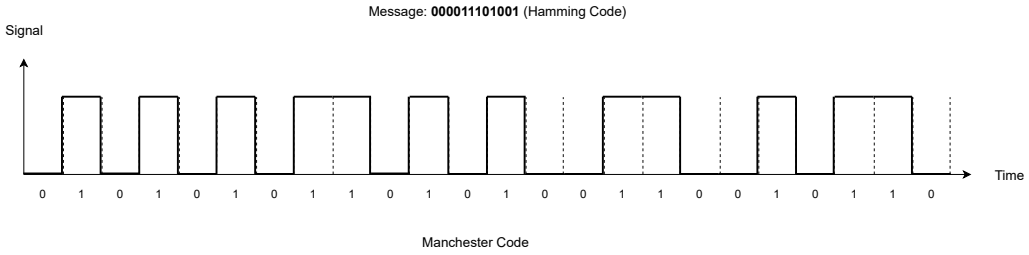

Figure 4.7: ECC to Manchester conversion

Multiple ECC techniques exist to achieve this goal such as Hamming [30], Reed-Solomon code [31] and turbo codes [32]. Hamming codes are linear error-correcting codes that can detect up to two erroneous bits or fix one-bit errors. Reed-Solomon Codes are linear error-correcting codes that can detect and correct multiple errors. Turbo codes are forward error correcting codes. We explain these ECC techniques in more detail below.

Considering the ECC options, we discuss the following error-correcting codes: Hamming code, Reed-Solomon codes and Turbo codes. Both Hamming and Reed-Solomon codes are linear ECCs, while Turbo code is a forward error correcting code (FEC).

Hamming codes [30] are named after Richard Hamming, who invented Hamming code in the 1940s while working on the Bell Model V computer. The basic principle of Hamming code is to build in a mechanism within the to-transmit data that allows the receiver of the data to check whether the message has suffered from an error or not. Take the example below, which is scalable.

Imagine we want to send 16 bits of data, represented by the 4 x 4 block shown in figure 4.8. Eleven of the sixteen bits will be actual data, while five bits in the block are reserved as redundancy bits. In this example, the bits on positions (0,1) and (0,2) as well as (1,0) and (2,0) are parity bits. The bit at position (0,1) checks the parity of columns 1 and 3 while the bit at position (0,2) checks the parity of columns 2 and 3. Similarly, the bit at position (1,0) checks the parity of rows 1 and 3 while the bit at position (2,0) checks the parity of the bits in rows 2 and 3. Checking the parity means checking whether there is an even or an odd number of ones in the columns or rows. If the amount is even, the parity bit is left at zero. Otherwise, it is overwritten to one. The special parity bit at position (0,0) is used as a parity bit for the whole block and allows us to detect two errors in the block, but we cannot fix them in that case. This example in practice carries 11 useful bits and is called a (15, 11) Hamming code.

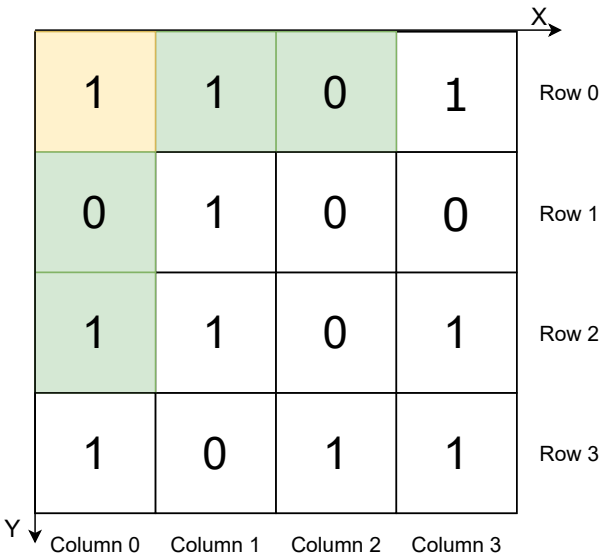

Figure 4.8: A (15, 11) Hamming code example

This solution scales well, since the parity bits are chosen to be put in positions that are powers of two. In this example, we used 4 parity bits in a block of 16 bits for error correction. This scales favourably however. Imagine a block of 256 bits (16 x 16). In this block, we only need 8 parity bits for error correction. You will be able to identify any single-bit error or detect that two errors have occurred. Hamming code works similarly for even larger blocks of data, and allows us to have error checking capabilities with a very small footprint.

Reed-Solomon codes [31] are error correcting codes invented by Irving Reed and Gustave Solomon in 1960, both MIT staff members. The ECC has many applications in modern products, including storage devices such as DVDs, as well as wireless communication, high speed modems and more. Like Hamming code, the basic idea of Reed-Solomon error correction is to add a block of redundant information (bits) to the transmitted message or data. When noise occurs during the transmission, the Reed-Solomon decoder will be able to detect said noise and restore the damage done, effectively recreating the original message.

Reed-Solomon codes are linear block codes. A Reed-Solomon code is specified as $R_S(n, k)$ with s-bit symbols. Notice that we talk about symbols, and not singular bits anymore. A symbol is comprised of multiple bits, with which Reed-Solomon works. The encoder takes k data symbols of s-bits each and adds parity symbols to make an n-symbol code word.

The data block of n symbols consists of k data-symbols and n-k parity symbols. A Reed-Solomon decoder can correct up to t symbols that contain errors in a codeword, with the equation $2t = n - k$.

For example, suppose each codeword contains 255 code word bytes and 8-bit symbols, so s = 8, n = 255. Of the 255 bytes, 223 bytes are data and 32 bytes are parity symbols. When plugging these values into the mentioned formula, we get k = 223 and 2t = 32. As a result, t = 16. This means that the Reed-Solomon ECC can automatically correct up to 16 bytes anywhere in the codeword. Reed-Solomon codes are well suited for correcting burst errors, where a series of consecutive bits in the codeword are erroneous.

We won't explain the exact mathematics behind Reed-Solomon codes. Moreover, during the decoding procedure, Reed-Solomon uses matrix inversion which is computationally heavy. This is the reason many applications for Reed-Solomon have custom hardware to make these computations much faster/easier for the device to execute.

Turbo codes [32] are forward error correcting codes invented in 1991 by Claude Berrou. Turbo codes are thus relatively new, but widely used thanks to their impressive performance. Turbo codes are used when encoding data to be sent over 3G and 4G networks, while NASA for example uses turbo code for all of its space probes sent to space since 2003.

Compared to Hamming and Reed-Solomon codes, turbo codes are more focused on integrating information redundancy in the sent message. Turbo

codes are based on the idea of sending the original message in three copies. The first copy of the message is unaltered. The second version is modified by encoding each bit of information using a certain algorithm shared with the receiver (decoder). A third version of the message is sent, but not the original message but rather a transformed version of the original. The receiver decodes the three received versions of the message and encodes them to find the original message.

The mathematics behind turbo codes is even more complex than Reed-Solomon and out of scope. However, interesting to note is that turbo codes are close to the theoretical limit for information transmitted with an error rate close to zero, and are thus highly efficient error correcting codes. Consider however that in most applications, turbo codes are hardware accelerated.

We chose to use Hamming codes, since it is an effective yet relatively simple way to protect our message from errors. Compared to Reed-Solomon and turbo codes, it does not require special hardware to decode messages quickly.

4.3.2 Manchester Code

In order to circumvent this issue of synchronisation and multiple successive identical symbols entirely, we use *Manchester encoding* when transmitting the hamming-encoded UIDs. Manchester encoding, also known as Phase encoding, is an encoding technique in which every binary symbol is translated into two binary symbols, but guarantees that the signal does not contain more than two identical successive symbols.

The main advantage of using Manchester code in the context of visible light communication is the effect the encoding has on the transmitted message. The Manchester-encoded message ensures that the light source transmitting the payload will not send more than two successive low signals. This means that the lamp will never be switched OFF for more than the time it takes to send two low signals.

Manchester encoding achieves this by translating the symbols as follows: 0 (binary) = 01 (Manchester) and 1 (binary) = 10 (Manchester) as defined by G.E. Thomas. The IEEE 802.3 standard defines Manchester encoding as follows: 0 (binary) = 10 (Manchester) and 1 (binary) = 01 (Manchester). In this thesis, we use the latter definition.

As you can see, by encoding the message in this way, no more than two identical successive symbols can exist. The downside of using Manchester code is that it effectively requires doubling the data to be sent in order to effectively transmit the same message, as each symbol is encoded by two symbols. We are willing to pay this price to have unambiguous messages.

In order to decode the sent Manchester code, the receiving-device has to keep track of two things when decoding: First, correctly deciphering the sent signal to either zero (0) or one (1). Second, keeping track of the received signals in order to reconstruct the transmitted Manchester signal. If we can reconstruct the message, the code can be deciphered and the UID retrieved.

Considering the first point, determining whether a signal is zero (0) or one (1) is done by detecting light source(s) on the received image. If a light source is found, the signal is interpreted as one (1). Otherwise, the signal is interpreted as zero (0), How a light source is detected on an input image is discussed in Section 4.2.

As mentioned in point two above, in order to reconstruct the Manchester code, every input frame must be evaluated and deciphered. Important to note is that not every input frame contains useful information, as in not every frame is part of the transmitted signal. We must determine which frames are useful to decipher and which ones are not part of the signal.

To help deciding which frames are part of the transmission and which ones are not, the transmitter itself is programmed to transmit the signal in a certain pattern. Typically, the transmitter's rest state is having it's light ON. We define an *interval time* as well as a *sampling time* known by both the transmitter and receiver. These values contain the time between transmissions and time between signal changes respectively. In other words, the message in the form of Manchester code, is transmitted every interval time (milliseconds). During transmission, each signal is sent for a duration of sampling time (milliseconds) before the signal changes again. After each interval period during which the light is ON, a low signal or light OFF is transmitted to signal the start of the new transmission.

In practice when decoding the message, we keep track of the start of the transmission, the already-received Manchester code and the history of received frames.

4.3.3 Decoding the Message

The transmitter sends the Manchester encoded message. To decode this message, every frame of the input video is analysed. Depending on the current status of the decoding progress of the message, this frame is either significant or not.

The algorithm distinguishes 4 transmission states that are used for decoding the message: Idle, Start, Decoding and Stop. These 4 transmission states as described in Appendix A describe the current state of the transmission and determine the next steps.

The algorithm keeps track of 2 histories: The frame object history and the transmission history, found in Appendix A, keep track of significant past frames and recorded transmissions respectively. The frame history is used to determine whether a signal starts and helps when recording the Manchester code. The transmission history is used primarily for the ability to receive multiple messages and store them.

The decoding algorithm seen in Appendix B works as follows. For each frame we receive as input, we check the current transmission status. By default this state is Idle. If the length of the until-now received Manchester code is 0 (zero) and the current binary value of the frame is also 0 (zero) — so no light source was detected on the image, we change the transmission-status from Idle to Start if and only if a certain amount of consecutive frames prior to this image in the frame history have a binary value of 1 (one). Detecting this zero signal is the start-sign of the transmission. We do not count this zero-signal to be part of the transmission however, as this merely notifies the start of the signal but is not part of it itself. Every consecutive frame after the zero signal is considered to be part of the message.

If the until-now recorded Manchester code's length equals the length of the sent image (known beforehand to both sender and receiver), the transmission status is set to Stop. The Start and Stop states are only used briefly for setup however. In Start's case, this state is used to clean the frame history and prepare to start decoding the next coming frames which contain the message signals. The transmission state is also set to Decoding after. In Stop's case, the received Manchester code is decoded and converted to decimal. This decimal number represents the transmitter's UID. The received as well as decoded messages are added to the transmission history. Apart from this, the frame history is cleared and the transmission state is set to Idle.

When the transmission status is set to decoding, this means that the frames we currently receive are part of the transmission and contain useful information. It is important to understand however that not every frame should be added to the received Manchester code. Depending on the input video, we probably have multiple frames per signal. This means that one signal sent by the transmitter, say a `1` (one) might actually be `111111` on video when we have 6 frames per signal. Clearly, not every frame's binary value should be added to the Manchester Code.

To determine when the signal is counted, we use the following logic. Since a history of the received frames is kept in the frame history, we can check the signal on the previous frame. If no previous frame exists, we know the current frame is the first frame of the new signal. In this case, we add the frame to the frame history and continue with the next frame. Idem when the previous frame does in fact exist, but its binary value equals the current frame's binary value. In this case, the current signal is still being transmitted.

More interesting is the case in which the current frame's signal does not equal the previous frame's binary value. In other words, we found a transition from either `1` to `0` or vice versa. In this case, we take a look at the frame history and count the number of consecutive identical frame signals. The count of these previous frames falls in two categories. Either we count *frames per signal* frames or we count two times *frames per signal* frames. We determine the binary value of the series of frames we just counted and add it's value to the Manchester Code. We add the binary value twice in the latter case, when the counted frames equals *frames per signal* times two.

However, a third case exists: consider the count of the previous consecutive x-frames which have the same binary value to exceed *frames per signal* times two. This situation is technically impossible since we work with Manchester Code. As already described in 4.3.2, the major reason to use Manchester code is precisely it's property that more than two consecutive identical characters cannot occur in a valid code. Therefore, we consider codes in this category to be faulty. In this case, we reset both the frame and transmission histories and set the transmission status to Idle. We effectively reset the decoding procedure and discard the decoded message so far in order to start over.

After running through all the frames of the video, the algorithm prints out all the received and decoded messages contained within the input.

5

Evaluation

Using our VLC implementation discussed in Chapter 4, we evaluated the system using a indoor positioning setup as proof of concept. In our evaluation we want to evaluate the following parameters:

- **Position accuracy**: The accuracy of our system to correctly decode received messages and position itself in the environment.
- **Distance**: Evaluating the impact on the positioning accuracy by increasing or decreasing the distance between sender and receiver.
- **Frame rate**: Evaluating the impact that the frame rate of the camera has on positioning accuracy.
- **Sample time**: Evaluating the impact of the sample time (the time to transmit a high or low signal) on positioning accuracy
- **Fault tolerance**: The way our system deals with less-than-ideal situations, such as bad capture angles, dimly or brightly lit rooms, incomplete transmissions and other factors.

5.1 General test setup

In order to test the solution proposed in section 4, we used the following setup.

We use screens such as a desktop computer monitor that acts as a transmitter in our solution, instead of using real lamps. The main reason for using a screen as sending device are twofold.

First, using screens allows us to skip the usage of hardware controllers such as Arduino or Raspberry Pi. Such hardware devices are necessary in order to program a regular light to be able to transmit a signal. Since the usage of hardware controllers is not the focus of this thesis, we decided to not use them in order to keep the development of the solution cheaper and simpler.

Second, screens are more easily programmable which allows for faster and more effective testing during the development phase. Switching from one to another transmission method is easy, quick and does not require the setup needed to use a real light source with hardware controller.

A smartphone is used as receiver in the solution. The phone films the transmitter using its camera in order to create a video of the transmission. Important to emphasize here is that we primarily used the rear camera and created a video file but did not stream the video feed live to the presented algorithm. The solution runs on PC such as a desktop or laptop, not on the phone itself.

After recording a video of the transmission made by the screen, the video file is fed to the algorithm which analyses the frames contained within to extract and decodes the message sent by the transmitter.

5.2 Controlled Experiments

To test the working of our proposed solution, we conducted some controlled experiments. Before sharing the results, we would like to give some context and define the symbols in the table.

For each entry in the table, the same message was sent by the transmitter (sent column). The sent data is always decimal `121` which translates to the

Figure 5.1: The detection of our light source at 1 meter distance

following in binary: `010101011010100110010110`. This binary value is not the direct translation from `121` to binary as you probably noticed. Instead, three operations take place before the signal is sent.

The first step in the conversion is to translate `121` to its binary form, which is `01111001`. Next, we encode the binary number using Hamming encoding. This results in `000011101001`. The last step requires to convert this ECC to Manchester code, which gives `010101011010100110010110`. The reason for the conversion from binary to ECC and from ECC to Manchester Code can be found in Section 4.3.

Interesting to note is that the frame rate of the video file fed to the algorithm is important. Frame rate, or frames per second (fps) of a video file refers to the number of frames (images) that the video files plays every second. In general, a higher frame rate is always better for the algorithm as the higher frame rate allows the camera to capture more events in frame than a lower frame rate.

Sample time is a characteristic of the transmitter and describes the time in milliseconds (ms) that it takes the transmitter to send either a high `1` or a low `0` signal. For example, a sample time of 100 ms means that transmitting a High or Low signal takes 0.1 seconds or 100 ms. Both the transmitter and receiver know the agreed upon sample time beforehand.

When combining the frame rate together with the sample time, we can also talk about frames per signal. This metric is calculated using the following

formula: $FramesPerSignal = ((Framerate/1000) * SampleTime)$. For example, when the framerate is `30` and the sample time `100`, the frames per signal is `3`.

We conducted several experiments using the frame rate, sample time and distance as variables. In most cases, the frame rate was set to 30 fps, while the sample time ranged from 200ms to 50ms and the distance ranged from 1 to 4 meters. The sent data was always the same.

Notice how the frame rate changes from 30 fps to 60 fps for a sample time of 50ms. The reason why we switched to a higher frame rate is simple: at 30 fps, a sample time of 50ms gives 1.5 frames per signal, or one frame per signal when rounded down (since half frames don't exist). While technically possible with a perfect system, having only one frame per signal in reality causes issues due to imperfections in video capture. On top of this, 30 fps is too low to capture signals transmitted at lower sample times than 50ms. We chose to use 60 fps here to show that while possible, lower sample times require higher frame rates to capture properly.

The following table 5.1 presents an overview of our findings. For each experiment, we recorded multiple data points as shown below. Each record specifies the frame rate or frames per second (Fps) of the video that we captured. The sample time (St) is measured in milliseconds and denotes the time to transmit either a high or low signal. The higher the sample time, the longer it takes to send a signal. This table also shows the frames per signal (F/Sig), a calculated value based on the frame rate and sample time. For each of the records we also specified the distance (Dist in meters) at which the video was captured.

In each of the experiments, the same message was transmitted: `121` in decimal which translates to `010101011010100110010110` when converted to binary with Hamming ECC applied and translated to Manchester code.

The message decoded by the application is shown in the received column, while the status column displays whether the decoded message was correct or not: OK for correct, NOK for incorrect.

Nr	Fps	St	F/Sig	Dist	Received	Status
1	30	200	6	1	010101011010100110010110	OK
2	30	200	6	2	010101011010100110010110	OK
3	30	200	6	3	010101011010100110010110	OK
4	30	200	6	4	010101011010100110010110	OK
5	30	150	4	1	010101011010100110010110	OK
6	30	150	4	2	010101011010100110010110	OK
7	30	150	4	3	010101011010100110010110	OK
8	30	150	4	4	010101011010100110010110	OK
9	30	100	3	1	010101011010100110010110	OK
10	30	100	3	2	010101011010100110010110	OK
11	30	100	3	3	010101011010100110010110	OK
12	30	100	3	4	011010101101010011001101 10	NOK
13	30	50	1	1	n/a	NOK
14	30	50	1	2	n/a	NOK
15	30	50	1	3	n/a	NOK
16	30	50	1	4	n/a	NOK
13	60	50	3	1	010101011010100110010110	OK
14	60	50	3	2	010101011010100110010110	OK
15	60	50	3	3	010101011010100110010110	OK
16	60	50	3	4	n/a	NOK

Table 5.1: The Controlled Experiment Results

5.3 Simulation Experiment

The previous experiments in section 5.2 focused on the results for a single transmission captured at a certain distance for a specific sample rate. In this experiment, we want to simulate the usage of this application when the input video is not captured at a certain distance for only one iteration, but rather a dynamic test where the user films multiple light sources at different distances by walking from room to room.

For this experiment, we set up 3 different monitors in three distinct rooms. Monitor A is set up in one room, while monitors B and C are setup in two other distinct rooms. Each monitor acts as a transmitter in the experiment while we use a smartphone as capture device or receiver. Every monitor A, B and C transmits its own signal, namely their ID. Monitor A transmits its ID `121` while monitors B and C transmit ID `122` and `123` respectively. Important to note is that we chose a sample time of 200ms for this experiment.

The goal of this experiment is to capture all the transmitted messages by monitor A, B and C and decode these messages correctly to match the original sent identifiers. Practically, we do this by filming Monitor A in the first room, walking to the second room to film monitor B and finally continue to light source C in the last room as illustrated by 5.2. In each of the simulation experiments, we filmed the transmitter at a distance of approximately 3 meters.

After filming this continuous video, the application analyses the entire video, captures and decodes the embedded signals. After processing the entire input, the decoded messages are printed. Now we can compare the decoded results to its transmitted counterparts.

We performed several simulation experiments and found that the app can decode the transmitted signals correctly most of the time. In other words, for the first room the result was `121` decoded while the second and third rooms respectively yielded `122` and `123`.

We must notice however that this was not always the case. We encountered that sometimes the signal could not be correctly decoded due to bright ambient light interfering with the transmission. We found that 80% of the time all three decoded messages are correct. The other 20% of cases, one or more messages could not be decoded.

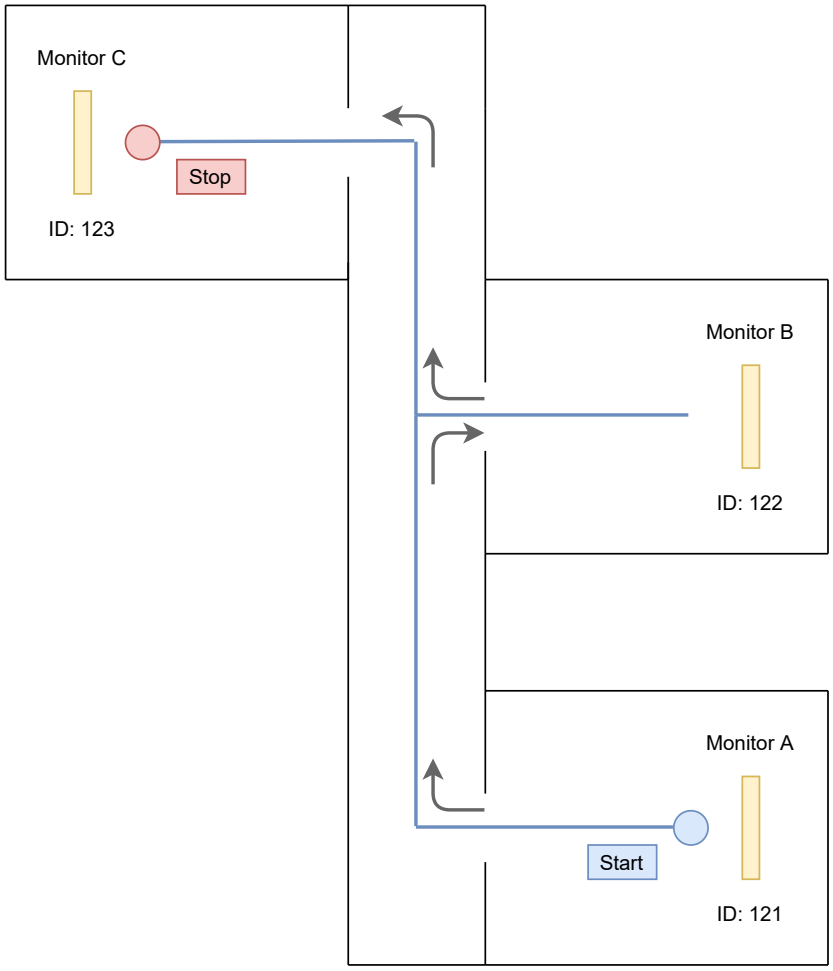

Figure 5.2: Simulation Experiment Path Diagram

6

Conclusion and Future Work

6.1 Conclusion

In this thesis we investigated indoor positioning techniques using visible light communication. We discussed the interest and motivation for such indoor positioning systems and why using visible light for this purpose is not only possible but also has many advantages over other indoor positioning techniques.

We discussed existing VLC positioning techniques such as fingerprinting (finding a close match between live data and pre-recorded measurements), triangulation (finding the absolute position using the geometric properties of triangles), vision analysis (using the geometric relationships between 3D objects and their 2D positions on an image) and proximity (approximating the receiver's position using the unique identifier's (UID) transmitted by VLC enabled light sources). The latter is the technique we proposed and used in our solution.

Our solution uses the proximity technique to determine the relative position of the receiver. In this solution, the transmitters broadcast their UID using On-Off Keying (OOK). The receiver records this transmission using its camera. The recording is analysed and the captured transmissions are read and decoded. A decoded transmission outputs a UID, which is checked against a known database. If a match is found, we know the approximate position of the capture device.

Evaluation of this solution is based on two different experiments: controlled situations and simulations. The first type of test investigates the performance of the solution when important properties of the transmitter and receiver change. We test the impact of distance from receiver to transmitter, the sample time of the transmission itself and the frame rate at which the receiver captures its footage.

The controlled experiments show that, for any sample time or frame rate, a distance of 1 to 3 meters is preferred. When testing at 4 (or more) meters distance, the algorithm often cannot correctly decode the captured transmission. The main reason for this phenomenon is the perspective in the individual frames themselves. Due to the nature of capturing video at 4 meters distance, the light source itself is smaller and more of the surroundings are in frame, creating a chance for interference. It is our experience that often the analysis of a signal at this distance will fail due to interfering bright light sources such as reflections from the sun. This can be picked up as a new light source, throwing off the results.

Frame rate of the recording and the sample time of the transmitted signal have a significant impact on the success of decoding a transmission. In general, a higher frame rate is always better as the receiver can capture more events in the same time. This success is also tied to the sample time however. A higher sample time means that the transmitter spends more time when sending a high or low signal, which is beneficial to the receiver as it has more time to capture the signal. In essence, the higher the sample time, the lower frame rate can be, but the longer the transmission of the message will take. The inverse is also true: If the sample time is lower, the frame rate needs to be higher to keep up, but the transmission of the signal will be faster.

We express this relationship between frame rate and sample time with *frames per signal*. This metric defines for how many frames (on average) in a capture a signal will be transmitted. For 30 fps and a sample time of 200, the frames per signal is 6. This means that every signal has on average 6 frames in the

recording that this signal is being sent. However, when the frame rate is 30 and the signal time is 50, the frames per signal is only 1 frame. This results in errors during decoding as seen in the results: none of the transmissions could be decoded in those recordings.

This is an issue we discovered when testing the algorithm: at 30 frames per second, the maximum signal time is approximately 100ms. When lowering the signal time any further, we risk consistent decoding failures during processing because in essence the receiver could not capture enough frames of each transmission signal for decoding to be effective. Improving the frame rate helps to resolve this issue. It must be noted however, that a sample time of 100ms is visible to the human eye and will definitely be noticed.

In the simulation experiment, we test the scenario in which someone walks from room to room, capturing the transmissions broadcasted by the transmitter in each room. Unlike the controlled experiments, this test consists of 3 different transmitters in 3 different locations who transmit a UID. Filming was done generally at 2 to 3 meters and after capturing one transmission, we walked to the next. This test aims to evaluate the solutions ability to pick up on transmissions when we arrive in a room where the current transmission is not finished yet. We chose to use a frame rate of 30 fps and a 200ms sample time.

In general, the algorithm performed well under these conditions. 80% of the time, all 3 UID's were correctly decoded in the test. In 20% of cases in which one or more decoded messages were incorrect was due to bright interfering lights from reflections or other light sources.

6.2 Future Work

We believe that the solution we presented is a good basis, but improvements can be made. Two prominent issues that we encountered and already discussed, are bright ambient light interference and frame rate bottleneck. An improvement to the light source detection in our algorithm, which can deal with additional interfering bright light sources can nullify this concern. One avenue to explore might be using the coordinates and radius of the detected light sources on the image, and using these values to filter out unwanted lights.

The frame rate bottleneck can be solved by using a camera with a substantially higher frame than the traditional 30 fps most cameras use to record footage. This might become more viable in the future. An alternative is to use the rolling shutter effect of smartphone cameras. Since most smart phone camera's use a rolling shutter camera, this should be possible. The related work 3.5 uses this property as well. One benefit of using the rolling shutter effect is the higher speed at which the light source can now switch. With this technique, it is possible to make the transmission of messages undetectable by humans due to the fast switch speed.

Some other improvements include the usage of real hardware instead of computer screens. This would include a LED lamp together with a small computer device (like an Arduino) to program and control the lamp. Currently we don't run the algorithm on the smart phone (receiver) itself but rather on a separate computer. We could either run the algorithm on the device itself or create a web app which is run in the browser. A server could run the algorithm in this case and output the results to the smart device. In this context, direct video streaming might be preferable over local video recording as we do now.

A

Appendix A: Constants and Objects

In this appendix we document the constants and objects that I designed for the proposed VLC positioning system, using Python. The procedures described in appendix B make use of these objects.

```
TARGET_FPS = 30 # Frames per second
TARGET_FPMS = (TARGET_FPS / 1000)  # Frames per milli-second
SAMPLING_TIME_MS = 200 # Time to transmit a high or low signal in milli-
    seconds
FRAMES_PER_SIGNAL = TARGET_FPMS * SAMPLING_TIME_MS
SIGNAL_SELECTION_RANGE_PERCENTAGE = 0.25 # Allowed deviaton/margin on the
    FRAMES_PER_SIGNAL in percentage
FRAMES_PER_SIGNAL_MIN = math.floor(FRAMES_PER_SIGNAL * ( 1 -
    SIGNAL_SELECTION_RANGE_PERCENTAGE))
FRAMES_PER_SIGNAL_MAX = math.ceil(FRAMES_PER_SIGNAL * ( 1 +
    SIGNAL_SELECTION_RANGE_PERCENTAGE))
FRAMES_PER_SIGNAL_TIMES_TWO_MIN =  math.floor((FRAMES_PER_SIGNAL * 2) * ( 1
    - SIGNAL_SELECTION_RANGE_PERCENTAGE))
FRAMES_PER_SIGNAL_TIMES_TWO_MAX = math.ceil((FRAMES_PER_SIGNAL * 2) * ( 1 +
    SIGNAL_SELECTION_RANGE_PERCENTAGE))
BINARY_MESSAGE_BIT_SIZE = 12
BINARY_MANCHESTER_MESSAGE_BIT_SIZE = (BINARY_MESSAGE_BIT_SIZE * 2)
```

Listing A.1: Global constants

```
class TransmissionStatus(enum.Enum):
 IdleTransmission = 1
 StartTransmission = 2
 DecodingTransmission = 3
 StopTransmission = 4
```

Listing A.2: Transmission states

```
class FrameObject(object):
 def __init__(self, width, height, originalImage = None, grayImage = None,
     coordinates = None, minMaxVal = None):
  self.width = width
  self.height = height
  if originalImage is None:
   self.originalImage = np.zeros((height, width, 1), np.uint8) # original
       color image
  else:
   self.originalImage = originalImage # original color image
  if grayImage is None:
   self.grayImage = np.zeros((height, width, 1), np.uint8) # grayscale image
  else:
   self.grayImage = grayImage # grayscale image
  if coordinates is None:
   self.coordinates = []
  else:
   self.coordinates = coordinates
  if minMaxVal is None:
   self.minMaxVal = (0, 0, 0, 0)
  else:
   self.minMaxVal = minMaxVal

 def getBinaryValue(self):
  if len(self.coordinates) > 0:
   return "1"
  else:
   return "0"
```

Listing A.3: FrameObject definition

```
class FrameObjectHistory(object):
 def __init__(self, maxKeptFrameObjects, frameObjectsInSMACalculation):
  self.maxKeptFrameObjects = maxKeptFrameObjects
  self.frameObjectsInSMACalculation = frameObjectsInSMACalculation
  self.frameObjects = []
  self.staticSMA = -1 # SMA calculated before the transmission starts
  self.signalSelectionBuffer = []
  self.startTransmissionOffset = math.ceil(FRAMES_PER_SIGNAL)

 ...

 # returns negative number if simple moving average (sma) cannot be
     calculated due to having no or too little frames the history.
 def calculateCurrentSimpleMovingAverage(self):
  sma_sum = 0
  sma_count = 0
  if len(self.frameObjects) >= self.frameObjectsInSMACalculation:
   startIndex = (len(self.frameObjects) - self.frameObjectsInSMACalculation)
   frameObjectsInCalculation = self.frameObjects[startIndex:]
   for frameObject in frameObjectsInCalculation:
    (minVal, maxVal, minLoc, maxLoc) = frameObject.minMaxVal
```

```
    sma_sum = sma_sum + maxVal
    sma_count = sma_count + 1
  elif len(self.frameObjects) > 0:
   return self.staticSMA
  if sma_count == 0:
   return -1
  else:
   sma = (sma_sum / sma_count)
   return sma

 def lightSourceDetectedOnAllPreviousXFrames(self, xframes):
  result = True
  if len(self.frameObjects) > xframes:
   startIndex = ((len(self.frameObjects) - xframes) - 1)
   stopIndex = (len(self.frameObjects) - 1)
   frameObjectsInCalculation = self.frameObjects[startIndex:stopIndex]
   for frameObject in frameObjectsInCalculation:
    if len(frameObject.coordinates) == 0:
    result = False
    break
  else:
   result = False # Not enough frames in buffer. Amount of frames needed =
       xframes + 1 (current frame)
  return result
```

Listing A.4: FrameObjectHistory definition

```
class TransmissionHistory (object):
 def __init__(self, transmissionStatus, binaryManchesterString,
     binaryMessageString, decimalMessageString) -> None:
  self.transmissionStatus: TransmissionStatus = transmissionStatus
  self.binaryManchesterString: str = binaryManchesterString
  self.binaryMessageString: str = binaryMessageString
  self.decimalMessageString: str = decimalMessageString
  self.transmissionHistory = []

 ...

 def addCurrentTransmissionToHistory(self):
  entry = (len(self.transmissionHistory) + 1, self.binaryManchesterString,
      self.binaryMessageString, self.decimalMessageString)
  self.transmissionHistory.append(entry)

 def printTransmissionHistory(self):
  print("Recorded␣transmissions:␣")
  for entry in self.transmissionHistory:
   string = str(entry[0]) + "␣||␣" + entry[1] + "␣|␣" + entry[2] + "␣|␣" +
       entry[3]
   print(string)
  return self
```

Listing A.5: TransmissionHistory definition

B

Appendix B: Procedures

In this appendix we report the procedures that I developed for the VLC positioning system, using Python. The procedures described here make use of the constants and objects as presented in appendix A.

```
def MainApplication():
 colour = BLUE
 thickness = 2
 frameCounter = 0

 video = "data/captured_video.mp4"
 cap = cv2.VideoCapture(video)

 bufferSize = SAMPLING_TIME_MS * TARGET_FPMS *
     BINARY_MANCHESTER_MESSAGE_BIT_SIZE
 frameObjectHistory = FrameObjectHistory(bufferSize, 15)

 transmissionStatus = TransmissionStatus.IdleTransmission
 binaryManchesterMessageString = ""
 binaryMessageString = ""
 decimalMessageString = ""
 transmissionHistory = TransmissionHistory(transmissionStatus,
     binaryManchesterMessageString, binaryMessageString,
     decimalMessageString)

 while(True):
  frameCounter = frameCounter + 1
  # Detect the light sources on the image
```

```
  (coordinates, processedFrame, newFrameObj) = processFrame(
      frameObjectHistory, frame, colour, thickness)
  # Show the image and its detected light sources
  cv2.imshow("Processed Frame", processedFrame)
  frameObjectHistory.addFrameObject(newFrameObj)
  (decFrameObjectHistory, decTransmissionHistory) = decodeFrameMessage(
      frameCounter, newFrameObj, frameObjectHistory, transmissionHistory)
  frameObjectHistory = decFrameObjectHistory
  transmissionHistory = decTransmissionHistory
  if cv2.waitKey(1) & 0xFF == ord('q'):
   break
 cap.release()
 cv2.destroyAllWindows()
 transmissionHistory.printTransmissionHistory()
```

Listing B.1: Main application

```
def processFrame(frameObjectHistory, frame, colour, thickness):
 resizedFrame = cv2.resize(frame, (480, 270))
 cleanFrame = resizedFrame.copy()
 # coordinates = [((x, y), radius)]
 (coordinates, newFrameObj) = getLightSourceCoordinates(frameObjectHistory,
     resizedFrame)
 processedFrame = drawLightSourceCoordinates(cleanFrame, coordinates, colour
     , thickness)
 return (coordinates, processedFrame, newFrameObj)
```

Listing B.2: The processFrame procedure

```
def getLightSourceCoordinates(frameObjectHistory, inputFrame):
 width = 480
 height = 270
 cleanImage = inputFrame.copy()
 originalImageMem = inputFrame.copy()
 coordList = []
 # get the grayscale version of the given frame
 grayImage = cv2.cvtColor(inputFrame, cv2.COLOR_BGR2GRAY)
 grayImageMem = grayImage.copy()
 # Use Gaussian blur to remove noise. Important to note: uses uneven numbers
 gaussBlur = cv2.GaussianBlur(grayImage, (31, 31), 0)
 # use frameObjectHistory to get SMA (Simple Moving Average) of brightest
     pixels
 if frameObjectHistory.staticSMA > 0:
  current_sma = frameObjectHistory.staticSMA
 else:
  current_sma = frameObjectHistory.calculateCurrentSimpleMovingAverage()
 # calculate the darkest and brightest pixels on the current image
 (minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(gaussBlur)
 minMaxVal = (minVal, maxVal, minLoc, maxLoc)
 # calculate range of pixel brightness
 percentage = 0.15 # percentage of pixels counted as part of light vs the
     brightest spot on the image
 rangeValue = (maxVal - minVal) * percentage
 thresholdValue = (maxVal - rangeValue)
 if current_sma == -1:
  current_sma = thresholdValue
 # calculate the threshold image using the calculated threshold and max
     values
 current_sma = current_sma - (current_sma * percentage)
 thresh = cv2.threshold(grayImage, current_sma, maxVal, cv2.THRESH_BINARY)
     [1]
```

```
# clean up spikkles (erode and dialate small artifacts)
kernal = np.ones((3,3),np.uint8)
thresh = cv2.erode(thresh, kernal, iterations=5)
thresh = cv2.dilate(thresh, kernal, iterations=2)
# find the contours of light sources on the image
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
# Draw the contours
contourImage = cv2.drawContours(inputFrame, contours, -1, (0, 255, 0), 2)
contourcounter = 0
for contour in contours:
 contourcounter = contourcounter + 1
 (x, y), radius = cv2.minEnclosingCircle(contour)
 center = (int(x), int(y))
 radius = int(radius)
 detectedCircleImage = cv2.circle(cleanImage, center, radius, (0, 255, 0),
     2)
 coordList.append((center, radius))
# sort and limit the coordlist
limitedCoordList = limitCoordList(coordList, 3)
# create a new Frame Object for the next iteration
newFrameObj = FrameObject(width, height, originalImageMem, grayImageMem,
    limitedCoordList, minMaxVal)
# return list of coordiantes = [((x, y), radius)] for each frame as well as
     the frame object for memory storage (later use)
return (limitedCoordList, newFrameObj)
```

Listing B.3: The getLightSourceCoordinates procedure

```
def decodeFrameMessage(frameCounter: Number, frameObject: FrameObject,
    frameObjectHistory: FrameObjectHistory, transmissionHistory:
    TransmissionHistory):
 global FRAME_COUNTER_OFFSET
 global BINARY_MESSAGE_BIT_SIZE
 # Determine whether a new transmission starts or stops. If not, continue
     with current transmission
 if len(transmissionHistory.binaryManchesterString) == 0:
  if frameObject.getBinaryValue() == "0":
   xframes = int(3 * SAMPLING_TIME_MS * TARGET_FPMS) # xframes = 2 * Sample
       Time frames (ms) * aantal frames per ms
   if frameObjectHistory.lightSourceDetectedOnAllPreviousXFrames(xframes):
    transmissionHistory.transmissionStatus = TransmissionStatus.
        StartTransmission
    FRAME_COUNTER_OFFSET = frameCounter + FRAMES_PER_SIGNAL
 elif len(transmissionHistory.binaryManchesterString) ==
     BINARY_MANCHESTER_MESSAGE_BIT_SIZE:
  transmissionHistory.transmissionStatus = TransmissionStatus.
      StopTransmission
 if transmissionHistory.transmissionStatus == TransmissionStatus.
     StartTransmission:
  transmissionHistory.binaryManchesterString = ""
  static_sma = frameObjectHistory.calculateCurrentSimpleMovingAverage()
  frameObjectHistory.staticSMA = static_sma
  frameObjectHistory.clearAllExceptLastFrameObjects()
  frameObjectHistory.clearSignalSelectionBuffer()
  # Set the transmission state to decoding when the frameHistory is prepared
  transmissionHistory.transmissionStatus = TransmissionStatus.
      DecodingTransmission
 elif transmissionHistory.transmissionStatus == TransmissionStatus.
     StopTransmission:
```

```
    binaryMessageString = convertManchesterStringToBinaryString(
        transmissionHistory.binaryManchesterString)
    binaryHammingString = convertHammingStringToBinaryString(
        binaryMessageString, BINARY_MESSAGE_BIT_SIZE)
    # Decode the complete received Hamming message
    decimalMessageString = convertBinaryStringToDecimalString(
        binaryHammingString)
    transmissionHistory.setBinaryMessageString(binaryMessageString)
    transmissionHistory.setDecimalMessageString(decimalMessageString)
    frameObjectHistory.clearSignalSelectionBuffer()
    frameObjectHistory = frameObjectHistory.clearAllExceptLastFrameObjects()
    transmissionHistory.addCurrentTransmissionToHistory()
    transmissionHistory.transmissionStatus = TransmissionStatus.
        IdleTransmission
    transmissionHistory.clearMessageStrings()
    frameObjectHistory.resetTransmissionOffset()
    return (frameObjectHistory, transmissionHistory)
if transmissionHistory.transmissionStatus == TransmissionStatus.
    DecodingTransmission:
    currentSignal = frameObject.getBinaryValue()
    previousSignal = frameObjectHistory.getLastSignalFromSignalSelectionBuffer
        ()
    if previousSignal is not None:
     if previousSignal == currentSignal:
      frameObjectHistory.addSignalToSignalSelectionBuffer(currentSignal)
     else:
      # A different signal compared to previous frames is received.
      framesInBuffer = frameObjectHistory.getSignalSelectionBufferCount()
      # Check how many consecutive frames in the buffer had the same signal
          and compare the cound to the pre-defined range.
      if FRAMES_PER_SIGNAL_MIN <= framesInBuffer and framesInBuffer <=
          FRAMES_PER_SIGNAL_MAX:
       transmissionHistory.binaryManchesterString = transmissionHistory.
           binaryManchesterString + previousSignal
       frameObjectHistory.clearSignalSelectionBuffer()
       frameObjectHistory.addSignalToSignalSelectionBuffer(currentSignal)
      # Two identical consecutive signals are possible. Check if this is the
          case.
      elif FRAMES_PER_SIGNAL_TIMES_TWO_MIN <= framesInBuffer and
          framesInBuffer <= FRAMES_PER_SIGNAL_TIMES_TWO_MAX:
       transmissionHistory.binaryManchesterString = transmissionHistory.
           binaryManchesterString + previousSignal + previousSignal
       frameObjectHistory.clearSignalSelectionBuffer()
       frameObjectHistory.addSignalToSignalSelectionBuffer(currentSignal)
      else:
       # More than two identical consecutive signals are not possible when
           using Hamming Code. An error occured while decoding the message.
           Reset and try again.
       transmissionHistory.binaryManchesterString = ""
       frameObjectHistory.clearSignalSelectionBuffer()
       frameObjectHistory.clearAllFrameObjects()
       transmissionHistory.transmissionStatus = TransmissionStatus.
           IdleTransmission
       frameObjectHistory.resetTransmissionOffset()
    else:
     # if the offset is not 0, we are currently receiving the start signal,
         which should not be counted.
     if frameObjectHistory.startTransmissionOffset == 0:
      # this is the first real signal, add to buffer
      frameObjectHistory.clearSignalSelectionBuffer()
      frameObjectHistory.addSignalToSignalSelectionBuffer(currentSignal)
     else:
```

```
    # subtract 1 frame from the startTransmissionOffset.
    frameObjectHistory.subtractFrameFromOffset()
return (frameObjectHistory, transmissionHistory)
```

Listing B.4: The decodeFrameMessage procedure

Bibliography

[1] William J Martin. *The Global Information Society*. Taylor & Francis, 2017. ISBN 978-0-566-07812-5.

[2] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and James Collins. *Global Positioning System: Theory and Practice*. Springer Science & Business Media, 2012. ISBN 978-3-211-82839-7. doi: 10.1007/978-3-7091-3297-5. URL http://dx.doi.org/10.1007/978-3-7091-3297-5.

[3] Haiyu Lan, Chunyang Yu, Yuan Zhuang, You Li, and Naser El-Sheimy. A Novel Kalman Filter with State Constraint Approach for the Integration of Multiple Pedestrian Navigation Systems. *Micromachines*, 6(7): 926–952, 2015. doi: 10.3390/mi6070926. URL http://dx.doi.org/10.3390/mi6070926.

[4] Hyun-Seung Kim, Deok-Rae Kim, Se-Hoon Yang, Yong-Hwan Son, and Sang-Kook Han. An Indoor Visible Light Communication Positioning System Using a RF Carrier Allocation Technique. *Journal of lightwave technology*, 31(1):134–144, 2012. doi: 10.1109/JLT.2012.2225826. URL http://dx.doi.org/10.1109/JLT.2012.2225826.

[5] Shlomi Arnon. *Visible Light Communication*. Cambridge University Press, 2015. ISBN 978-1-107-06155-2.

[6] Wikipedia, the free encyclopedia. Em spectrum, 2020. URL https://commons.wikimedia.org/wiki/File:EM_spectrum.svg. [Online; accessed 06/12/2020].

[7] Hiroaki Murata. Light Emitting Diode Lamp, June 1990.

[8] Rajan Sagotra and Reena Aggarwal. Visible Light Communication. *International Journal of Computer Trends and Technology (IJCTT) volume4 Issue4*, pages 906–910, 2013.

[9] U Nadeem, NU Hassan, MA Pasha, and C Yuen. Indoor Positioning System Designs Using Visible LED Lights: Performance Comparison of TDM and FDM Protocols. *Electronics Letters*, 51(1):72–74, 2015. doi: 10.1049/el.2014.1668. URL `http://dx.doi.org/10.1049/el.2014.1668`.

[10] Se-Hoon Yang, Eun-Mi Jung, and Sang-Kook Han. Indoor Location Estimation Based on LED Visible Light Communication Using Multiple Optical Receivers. *IEEE Communications Letters*, 17(9):1834–1837, 2013. doi: 10.1109/LCOMM.2013.070913.131120. URL `http://dx.doi.org/10.1109/LCOMM.2013.070913.131120`.

[11] Se-Hoon Yang, Deok-Rae Kim, Hyun-Seung Kim, Yong-Hwan Son, and Sang-Kook Han. Visible Light Based High Accuracy Indoor Localization Using the Extinction Ratio Distributions of Light Signals. *Microwave and Optical Technology Letters*, 55(6):1385–1389, 2013. doi: 10.1002/mop.27575. URL `http://dx.doi.org/10.1002/mop.27575`.

[12] Trong-Hop Do and Myungsik Yoo. An in-Depth Survey of Visible Light Communication Based Positioning Systems. *Sensors*, 16(5):678, 2016. doi: 10.3390/s16050678. URL `http://dx.doi.org/10.3390/s16050678`.

[13] Somayeh Mohammady. *Multiplexing*. BoD–Books on Demand, 2019. ISBN 978-1-78984-569-3. doi: 10.5772/intechopen.78513. URL `http://dx.doi.org/10.5772/intechopen.78513`.

[14] Tsung-Nan Lin and Po-Chiang Lin. Performance Comparison Of Indoor Positioning Techniques Based on Location Fingerprinting in Wireless Networks. In *2005 international conference on wireless networks, communications and mobile computing*, volume 2, pages 1569–1574. IEEE, 2005. doi: 10.1109/WIRLES.2005.1549647.

[15] Naveed Ul Hassan, Aqsa Naeem, Muhammad Adeel Pasha, Tariq Jadoon, and Chau Yuen. Angles Phi and Theta to compute channel impulse response where nt and nr are transmitter and receiver orientation vectors, respectively. Indoor Positioning Using Visible LED Lights: A Survey, 2016.

[16] Trong-Hop Do and Myungsik Yoo. Pinhole Camera Model, An In-depth Survey of Visible Light Communication Based Positioning Systems, 2016.

[17] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003. ISBN 0-521-54051-8.

[18] Trong-Hop Do and Myungsik Yoo. Collinearity Condition, An In-depth Survey of Visible Light Communication Based Positioning Systems, 2016.

[19] Trong-Hop Do and Myungsik Yoo. Two View Geometry, An In-depth Survey of Visible Light Communication Based Positioning Systems, 2016.

[20] Gregary B Prince and Thomas DC Little. A Two Phase Hybrid RSS/AoA Algorithm for Indoor Device Localization Using Visible Light. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 3347–3352. IEEE, 2012. doi: 10.1109/GLOCOM.2012.6503631. URL `http://dx.doi.org/10.1109/GLOCOM.2012.6503631`.

[21] Jia Ziyan. A Visible Light Communication Based Hybrid Positioning Method for Wireless Sensor Networks. In *2012 Second International Conference on Intelligent System Design and Engineering Application*, pages 1367–1370. IEEE, 2012. doi: 10.1109/ISdea.2012.411. URL `http://dx.doi.org/10.1109/ISdea.2012.411`.

[22] Mauro Biagi, Stefano Pergoloni, and Anna Maria Vegni. Last: A Framework to Localize, Access, Schedule, and Transmit in Indoor VLC Systems. *Journal of Lightwave Technology*, 33(9):1872–1887, 2015. doi: 10.1109/JLT.2015.2405674. URL `http://dx.doi.org/10.1109/JLT.2015.2405674`.

[23] Zhice Yang, Zeyu Wang, Jiansong Zhang, Chenyu Huang, and Qian Zhang. Wearables Can Afford: Light-weight Indoor Positioning with Visible Light. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 317–330, 2015. doi: 10.1145/2742647.2742648. URL `http://dx.doi.org/10.1145/2742647.2742648`.

[24] Phat Huynh and Myungsik Yoo. Vlc-based Positioning System for an Indoor Environment Using an Image Sensor and an Accelerometer Sensor. *Sensors*, 16(6):783, 2016. doi: 10.3390/s16060783. URL `http://dx.doi.org/10.3390/s16060783`.

[25] Jae-Yoon Kim, Se-Hoon Yang, Yong-Hwan Son, and Sang-Kook Han. High-resolution Indoor Positioning Using Light Emitting Diode Visible

Light and Camera Image Sensor. *Iet Optoelectronics*, 10(5):184–192, 2016. doi: 10.1049/iet-opt.2015.0073. URL http://dx.doi.org/10.1049/iet-opt.2015.0073.

[26] Rayana Boubezari, Hoa Le Minh, Zabih Ghassemlooy, and Ahmed Bouridane. Smartphone Camera Based Visible Light Communication. *Journal of Lightwave Technology*, 34(17):4121–4127, 2016. doi: 10.1109/JLT.2016.2590880. URL http://dx.doi.org/10.1109/JLT.2016.2590880.

[27] Ran Zhang, Wen-De Zhong, Kemao Qian, and Dehao Wu. Image Sensor Based Visible Light PositioningSystem With Improved Positioning Algorithm. *IEEE Access*, 5:6087–6094, 2017. doi: 10.1109/ACCESS.2017.2693299. URL http://dx.doi.org/10.1109/ACCESS.2017.2693299.

[28] Kenneth Levenberg. A Method For The Solution of Certain Non-linear Problems in Least Squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[29] Yang Wang and Hongdong Zhao. Improved Smartphone-based Indoor Pedestrian Dead Reckoning Assisted by Visible Light Positioning. *IEEE Sensors Journal*, 19(8):2902–2908, 2018. doi: 10.1109/JSEN.2018.2888493. URL http://dx.doi.org/10.1109/JSEN.2018.2888493.

[30] Chi-Shiang Chan and Chin-Chen Chang. An Efficient Image Authentication Method Based on Hamming Code. *Pattern Recognition*, 40 (2):681–690, 2007. doi: 10.1016/j.patcog.2006.05.018. URL http://dx.doi.org/10.1016/j.patcog.2006.05.018.

[31] Wael Halbawi, Navid Azizan, Fariborz Salehi, and Babak Hassibi. Improving Distributed Gradient Descent Using Reed-Solomon Codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2027–2031. IEEE, 2018. doi: 10.1109/ISIT.2018.8437467. URL http://dx.doi.org/10.1109/ISIT.2018.8437467.

[32] Yihan Jiang, Sreeram Kannan, Hyeji Kim, Sewoong Oh, Himanshu Asnani, and Pramod Viswanath. Deepturbo: Deep Turbo Decoder. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2019. doi: 10.1109/SPAWC.2019.8815400. URL http://dx.doi.org/10.1109/SPAWC.2019.8815400.