King's College London

Enhancing Spaces for VLC Communication: Optimal Room Design

Submitted by:

Ege Aybek
K23063206

Project Report Submitted in Partial Fulfillment of the Requirements for the M.Sc. Project Module

Engineering Department

Supervised by:

Prof. Rym M'Hallah

London
August / 2024

Abstract

The importance of optimizing visible light communication (VLC) systems in indoor environments lies in enhancing energy efficiency and communication quality, especially as smart lighting technologies become increasingly integral to modern infrastructure. This project sought to develop an algorithmic solution to optimize LED light placement and intensity, addressing challenges such as energy consumption, light uniformity, and object interference in various room setups. To tackle these issues, a Mixed Integer Linear Program (MILP) model was employed, incorporating constraints related to illuminance, power usage, and spatial configuration. The research involved practical experiments comparing the algorithm's performance with manual light placement, demonstrating that the algorithm consistently achieved better results in terms of uniform light distribution and energy efficiency. Key findings indicate that the algorithm not only improves VLC system performance but also offers significant potential for energy savings. However, the study did not account for real-time adjustments or advanced lighting features like dimmability, suggesting areas for future research to further enhance the adaptability and practical application of the proposed system.

Keywords: VLC, Python, CPLEX, MILP, Optimization, KMeans,

Table of Contents

List of Tables

List of Acronyms and Notation

Acronyms

VLC	Visible Light Communication
LED	Light Emitting Diode
LOS	Line of Sight
MILP	Mixed Integer Linear Programming

Notation

Table 1: Notation

Symbol	Description
B	LED types
$\alpha_i^{'}$	Power consumption of LED $i \in B$ during daytime
$\alpha_i^{''}$	Power consumption of LED $i \in B$ during night
$\beta_i^{'}$	Lumen of LED $i \in B$ during daytime
$\beta_i^{''}$	Lumen of LED $i \in B$ during night
$\tau^{'}$	Minutes in daytime period
$\tau^{''}$	Minutes in night period
c	Price per unit of energy consumption per minute
l	Room length
w	Room width.
$\varphi^{'}$	Minimum lux requirement during daytime.
$\varphi^{''}$	Minimum lux requirement during night.
x_i	Decision variable specifying number of LED's to be used from each LED type $i \in B$.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Rym M'Hallah, for her invaluable guidance, support, and encouragement throughout this project. Her insights and expertise have been instrumental in the completion of this work.

I would also like to thank my family for their unwavering support and encouragement. Their belief in me has been a constant source of motivation and strength.

Chapter 1

Introduction

Energy consumption is a key factor when it comes to reducing the cost. In 2022 the UK has consumed 154 Mtoe(million tonnes of oil equivalent) of energy and the %13 of this consumption is generated by the usage of electricity [1]. In terms of kWh (kilowatt-hour), 232,832.6 kWh of energy has been used just in the UK from electricity. Also, keeping in mind that the average percentage of energy consumption for lighting in commercial buildings is estimated to be between 25-40% of the total energy consumption [3], optimizing the lighting in cities can save an important amount of energy. Thus, optimizing the lighting in a building can lead to a big save in energy consumption, so does the cost.

In the context of optimization, it is crucial to consider the specific requirements to achieve desirable outcomes. Each system configuration may come with unique objectives and constraints. For instance, while both a house and an office fall within the category of indoor environments, the lighting needs for these spaces are different; offices often require enhanced lighting to create a productive working environment while the lighting requirement of a house will depend on the needs of residents. Technological advancements create new needs for indoor lighting of buildings. Visible Lighting Communication (VLC) is one of those advancements that brings new objectives and constraints to indoor lighting decisions.

VLC, as the name implies, is a wireless communication technology that uses visible light to send or receive data. It is predicted to have a very high variety of possible use cases in the future such as vehicle-to-vehicle communications and indoor positioning. For example, the vehicles are predicted to communicate by their headlights with the help of VLC and people will be able to find their way in multi-floor indoor environments such as shopping malls and airports. Thus, designing indoor environments according to the requirements of the VLC technology is a reasonable decision to be ready for the needs of

future.

VLC brings various types of requirements including a certain illuminance level within a space (a minimum of 500 lux as stipulated by the European Lighting Standard EN12464-1 [2]), the homogeneity of light to ensure consistent room illumination, the light sources to be light emitting diodes (LED) and a clear Line of Sight (LOS) for the receiver which is a definite advantage because the signal will be stronger [4]. So, having objects in the room is a constraining factor since they will block the direct LOS. Thus, the more objects in the room the more complex the problem is. While developing an optimization algorithm for indoor lighting, all these requirements of VLC technology and constraints must be considered.

In this study, we developed a Mixed Integer Linear Program (MILP) aiming to determine the required number of LEDs of each type within a room, by ensuring the average illuminance level in the room will be above the required level while minimizing costs. The obtained solution from MILP is fed into a simulation model that finds the optimal positions of the LEDs in a room containing various rectangular-shaped objects. This model is designed to optimize the average level of illumination throughout the room while ensuring the homogeneity of light distribution. The proposed model is fast and effective even for very large sized rooms with multiple objects and it will enhance technological infrastructure for VLC systems in buildings.

Outline

This project report is organized as follows. Chapter 2 reviews the literature on visible light communication (VLC) systems, LED optimization, and related computational methods, providing a foundation for the research. Chapter 3 defines the problem by outlining the challenges in optimizing LED placement and intensity for VLC, and models it using a Mixed Integer Linear Programming (MILP) framework. Chapter 4 describes the solution approach and methodology, detailing the use of MILP, KMeans clustering, and the imple-

mentation of the optimization algorithm in Python. Chapter 5 presents the experimental setup and results, showcasing the effectiveness of the proposed solution through various test scenarios and comparative analyses. Chapter 6 discusses the broader engineering issues and ethical considerations related to the deployment of VLC systems, including compliance with standards and the impact on energy consumption and sustainability. Finally, Chapter 7 summarizes the research findings, reflects on the contributions to the field, and provides potential future extensions, including real-time system adaptability and the integration of advanced lighting technologies.

Chapter 2

Literature Review

Sections 2.0.1 surveys the literature on VLC related problems and introduces existing some solutions on the specific studies. Section 2.0.2 reviews optimization applications for illumination in the indoor environments.

2.0.1 VLC Based

There are few studies that exists based on the optimization on VLC based systems however all of these studies focuses on different objectives. [6] and [13] are focused on a similar manner which is the LED arrangement on the ceiling while making the room support VLC. [6] is mostly does the lamp arrangement to create [11] a better uniformity across the room, on the other hand, [13] focuses on power consumption. [6] creates an office room scenario that has a fixed size of $7.8 \times 6 \times 3\ m^3$. A light reciever is placed in the center of the room on a desk that has a fixed height of 0.8m and there are total of 6 LED's in the scenario which each of them having a total luminous flux of 3600lm. As the result, the simulation results show that compared with the original arrangement, the uniformity of the illumination can be improved from 0.55 to 0.86, which ensures almost equal lighting effects for all users at different locations in the room [6]. Thus, since the uniformity is dispersed better, the quality of VLC increases accordingly. [13] has a similar approach with [6] to the problem. [13] assumes a room with dimensions 7.5 x 5 x 3 m^3. In addition to [6], [13] also considers the communication constraints in the problem while minimizing the power consumption by rearranging the LED's on the ceiling of the room. At the end, [13] compares their result with the traditional grid based LED placement on the ceiling and they result in a 14% power efficiency in general. By combining these two studies, it can be stated that the LED arrangement on the ceiling have remarkable impact on creating

homogeneity of lights in the room and decreasing power consumption. However, compared to our work, they do not include any objects in the room, thus no shadow affect on the floor is taken into account.

[12] and [10] both focuses on general power consumption in VLC systems. [12] studies the joint factor of VLC which is VLC is being capable of illumination and data transmission. This is by far the best feature of VLC. However, [12] finds that by doing both tasks, the power consumption is also increases. In the Joint Illumination and Communication Systems (JIC), the light that is emitted from the LED's is modulated with a higher rate then the human eye can perceive. Thus, it is possible to send data through light while humans see no difference in lighting. However, [12] has two main and interesting finding from the studies. The first finding is: The power consumption required to enable data transmission on top of illumination from any LED source is always higher compared to illumination only[12]. This is an interesting outcome, because compared to the energy consumption that is created from an LED emits light with a constant rate of μ always consumes less energy than an LED's that emits light with a modulation. The extra power consumption in here is being called as P_{extra}. The second finding of the study is focused around this P_{extra}. They result that the amount of P_{extra} in a JIC system is affected by the required illumination pattern[12]. These two findings shows the important of our project in a way that the VLC systems must be optimized, because, if not optimized, they will always consume more energy than regular lighting in a room. If we look at the other side, [10] focuses on the effect of illumination intensity on VLC sytems. Saying that, how the Signal to Noise Ratio (SNR) is affected on the receiver side when the illuminance of a single LED is increased. The finding out of this research is also interesting and is also a proof of why our study is important. [10] finds out that when the illuminance level of an LED is increased on the transmitter side, the SNR on the receiver side decreases accordingly so the average power loss increases. The test shows that, when the illuminance of LED increased from 400 lx to 7000 lx, the system SNR decreased significantly from 19.4 dB to 6.3 dB. Meanwhile, the average power

loss increases from 1.9 dB to 11.1 dB respectively [10]. This interesting outcome occurs because of the non-linearity of the photodetectors (receiver). Because we know that after a certain point, increasing the light intensity might not result in a proportional increase in detected signal power due to the saturation of the receiver or non-linear response curves of the photo-detectors. This interesting result is also a proof that optimization is an important factor on VLC systems, because increasing the illuminance is not a simple way of making VLC systems better. Thus, what makes the VLC system the best is to optimize the system in a way that works just as perfect depending on the needed constraints.

Apart from all of these studies, [9] focuses on creating a general understanding towards VLC and give answers to some key questions in the VLC. First of all [9] explains a very critical issue which is the usage of VLC in dark environments. For instance poeple also use their smartphones when they are in their bed, right before they go to sleep. However, when usually this is the case the lights in the room are switched off. Thus, no visible light in the room is present which creates a non feasible environment for VLC. However, [9] quotes from [11] that, the VLC is also usable in dark environments. The main idea of [11] is to encode the data into very short pulses of light at a very high frequency. This way, the light wave is imperceptible to human eyes but it can still be detected by photo-diodes. When this being the case, a huge problem in VLC is being solved. Apart from this [9] also shows the importance of the indoor positioning systems using VLC and discusses the methods on how to make the indoor positioning systems possible. [9] states that there are three main ways to make indoor positioning systems possible which are calculating the Receiver Signal Strength (RSS), calculating the Time of Arrival (TOA) and calculating the Angle of Arrival (AoA). However, using RSS and TOA is not feasible for VLC because RSS decreases when the signal collides with an object or when the transmitter becomes far away from the receiver, in which both of them are highly possible to occur in VLC systems. TOA is also not feasible because in ToA, the system needs rigidly synchronized signals between transmitter and receiver which makes the system more complex and expensive. However,

the AoA works when the receiver is in the line of sight with the transmitter and this is the one of the main requirements of VLC. Thus, [9] states that AoA is a feasible approach to create indoor positioning systems using VLC. Even tough the content that is discussed both in [9] and [11] are not directly related to our project, they create insights about why our project is doable and has so much potential for the future.

2.0.2 Optimization Based

[8] and [5] both focuses on the optimization of indoor lighting. Even tough both of these studies have nothing to do with VLC, when it comes to the optimization part, help us gain better understanding of the optimization of indoor lighting.

First of all, even tough no VLC constraint is applied in [5]'s work, the study in general, is a similar one to ours. [5] tries to find the locations of LED's on the ceiling that creates the best illuminance level in the room. It does not make any selection on how many LED must be used, it only focuses on the locations of the LED's. However, it differs from our project in several factors. First of all, no object is present in the room. Thus, there is no contribution from the shadows to the average illuminance in the room. In addition to that, presumably, since no objects in the room are present, the locations of the LED's on the ceiling does not depend on the objects. Second of all, there are no VLC constraints present in the study. Even tough, because of these factors, it differs from our project, it is a good example of showing that LED arrangement on the ceiling is an important factor when it comes to optimization of illuminance in an indoor environment.

Secondly, [8] creates a test case where the dimensions of the room are constant an it is $8 \times 6 \times 2.8\ m^3$. They create a multi-objective genetic algorithm to find the dim ratio on each of the LED on the ceiling to create the enough illuminance in the room. The number of LED's are also constant and it is set to 15. However, the locations of the LED's does not change. Also, shadow effects of the objects are not included in the work. At the end, it calculates how much each LED must be saved and they result in a total energy saving of

18%. Dimmability function is not applied to our project, however, it creates a good idea for the future of our work.

Table 2.1: Comparison of VLC Optimization Aspects in Various Studies

	Our Project	[6]	[12]	[13]	[9]	[10]	[8]	[5]
Dynamic Room Size	Yes	No	-	No	-	-	No	-
LEDs Number Decision	Yes	No	-	-	-	-	No	-
Objects in Room	Yes	-	-	-	-	-	-	No
LEDs not Dimmable	Yes	-	-	-	-	-	Yes	-
Calculates Shadow Effect	Yes	-	-	-	-	-	-	No
Finds Light Positions	Yes	Yes	-	Yes	-	-	-	Yes
Static Room Size	-	Yes	-	Yes	-	-	Yes	-
Communication Constraints	-	-	-	Yes	-	-	-	-
Dark VLC	-	-	-	-	Yes	-	-	-
Power Consumption	-	-	Yes	-	-	Yes	-	-
SNR with Illuminance	-	-	-	-	-	Yes	-	-
Dimmable LEDs	-	-	-	-	-	-	Yes	-
No Object Presence	-	-	-	-	-	-	-	Yes
No Shadow Effect Calculated	-	-	-	-	-	-	-	Yes
Includes Nothing about VLC	-	-	-	-	-	-	Yes	Yes

Note:

A "Yes" indicates that the aspect is present in the project or article.

A "No" indicates the aspect is specifically mentioned as not present or is a direct contrast to your project's capabilities.

A "-" (dash) is used where the aspect is not mentioned or is not relevant to that particular article, implying neither a direct match nor a contrast.

Chapter 3

Problem Statement

3.1 Problem Definition

As previously stated, the problem comes from the fact that the VLC technology has it's own rules and constraints. Since the VLC is a communication technology based on lighting, supplying the room with lots of LED's may seem like an option and a solution. In fact, it is a solution indeed. However, it is not efficient and it creates excessive cost because of the energy consumption. Apart from that, it is not sustainable at all.

Thus, in our project we wanted to create a model that makes the room VLC available in the most efficient way possible. So, our project aims on finding the minimum amount of LED's placed in the room to make the room satisfy the needs of VLC. By doing that, it is possible the make the room VLC available by consuming the minimum amount of energy.

In the setting, there are key parameters to go over. These are as follows:

- Number Of LED's required
- Value of illuminance required in the room
- Types of LED's (assuming each LED has a different lux level)
- Shadow effect
- Placement of lights on the ceiling

Since change of all these parameters directly affect the VLC quality, all of these parameters are a problem of creating an efficient VLC available environment. Thus, while creating the model, we had the take these into account and create the solution on top of that.

3.2 Model

As stated before, our project is based on a MILP optimization problem. It has one objective function, two main constraints as well as the non-zero constraints.

The objective function minimizes the price by computing the the day and night consumption of each LED, and the constraints help us to pick the correct LED's to ensure a minimum level of illuminance level within the room during day and night.

The objective function computes the price for an LED during the daytime and night and sums these to find the total price of an LED throughout the day. Thus, in general, it minimizes the cost in the model.

The two constraints basically multiplies the number of LED's with the illuminance values and divides it to the room size. By doing that, we can create a constraint for the average lux value for daytime and night respectively.

Table 3.1: Decision Variables

Symbol	Description
B	LED types
$\alpha_i^{'}$	Power consumption of LED $i \in B$ during daytime
$\alpha_i^{''}$	Power consumption of LED $i \in B$ during night
$\beta_i^{'}$	Lumen of LED $i \in B$ during daytime
$\beta_i^{''}$	Lumen of LED $i \in B$ during night
$\tau^{'}$	Minutes in daytime period
$\tau^{''}$	Minutes in night period
c	Price per unit of energy consumption per minute
l	Room length
w	Room width.
$\varphi^{'}$	Minimum lux requirement during daytime.
$\varphi^{''}$	Minimum lux requirement during night.
x_i	Decision variable specifying number of LED's to be used from each LED type $i \in B$.

Using the above notation, MILP model for lighting optimization problem is written as follows.

$$\min z = \sum_{i \in B} x_i (\alpha_i^{'} \tau^{'} c + \alpha_i^{''} \tau^{''} c) \tag{3.1}$$

$$\frac{\sum_{i \in B} x_i \beta_i^{'}}{lw} \quad \geq \varphi^{'} \tag{3.2}$$

$$\frac{\sum_{i \in B} x_i \beta_i^{''}}{lw} \quad \geq \varphi^{''} \tag{3.3}$$

$$x_i \quad \in \{0, 1, 2...\} \tag{3.4}$$

Objective function (3.1) minimizes cost of total power consumption. Constraints (3.2) and (3.3) ensure that average lux in the room will be above the minimum requirement for daytime and night respectively. Constraint (3.4) is the boundary constraint.

Chapter 4

Methodology

4.1 Introduction

The methodology consists of two main parts. The first part is solving the optimization problem. The second part is using the algorithm with machine learning algorithms to define the light locations and illustrate the results.

4.2 Used Methodology

As stated before, the project is done in two main parts, the optimization problem and the location finder algorithm to find the pest positions of LED's on the ceiling.

First of all, all the project is coded in Python and the optimization problem is solved with CPLEX solver. CPLEX is a tool for solving, first of all, for linear optimization problems [7]. The software is created by IBM. So, the path of the CPLEX solver has to be added to the Python Kernel Environment on your local computer so that the system knows which solver to use. After doing that, all the components of the optimization problem (constraints, objective function, etc.) are coded in Python. All other data, such as LED types and lumen values of the LEDs, are saved in separate .csv files in the same path. This makes the system dynamic so that the user can change these with what they have in their hands and use the algorithm easily. These data are taken out from the code and placed in separate .csv files to make it more user-friendly. Since these data are not embedded in the code, a non-technical user can easily change these data and run the algorithm. The .csv files contain:

- Number of LEDs that can be used

- The Lumen Value for each LED for daytime
- The Lumen Value for each LED for night
- The power consumption (Watt per minute) of each LED for daytime
- The power consumption (Watt per minute) of each LED for night

Thus, there are five .csv files in total that the code takes its data from.

After gathering these datas from the .csv files, the CPLEX solver solves the optimization problem and returns an array telling us how many LEDs should be used and which LEDs should be used for the given room (the room dimensions should be set within the code), which leads us to the end of the first part of the project. Also, to have a better understanding of the optimization algorithm, please check Algorithm 1.

The second part of the project is all about finding the exact locations of LEDs on the ceiling that generate the maximum average illuminance whilst keeping the standard deviation as low as possible. By doing so, we can ensure that the homogeneity of lighting in the room is also satisfied.

Finding the best LED locations on the ceiling is done via comparing experimental results. To do that, the ceiling and the floor of the room are divided into grid points. In the ceiling, all possible combinations of light positionings are tried and the illuminance level of each grid point in each case is recorded as can be seen from the Figure 4.1. At the end, the LED positions with the best illuminance performance on the floor are picked by the algorithm. However, a small assumption is made here. Normally, it is known that light travels in rays. However, one light source can reflect from walls or an object and affect other points in the room. But, in this project, the reflection of light is not considered. This is done intentionally because, in VLC, the direct Line of Sight (LOS) is important to hold the data between the transmitter and the receiver. Therefore, there is no need to consider the reflection of light.

Algorithm 1 Optimization Problem Pseudocode

Input:
Room dimensions: *length*, *width*, *height*
Spacing: *spacing*
Minimum illuminance: *min_lux_Day*, *min_lux_Night*
Time intervals: *minutes_day*, *minutes_night*
Energy cost: *price*
Bulb data: *Bulbs*, *PowerDay*, *PowerNight*, *LumenDay*, *LumenNight*
Objects in the room: *objects*
Initialization:
Create grids for ceiling and floor
Iterative Steps:
for each bulb i in *Bulbs* **do**
    Calculate grid points: $grid\_points \leftarrow (room\_width+1)*(room\_length+1)*spacing$
    Define optimization model:
        $model \leftarrow$ AbstractModel()
        Define sets, parameters, and variables
        Define constraints:
            ConstrainGridCapacity: $\sum model.X[i] \leq grid\_points$
            ConstrainMinLuxDay: $\frac{\sum model.X[i] \cdot model.LumenDay[i]}{room\_length \cdot room\_width} \geq min\_lux\_Day$
            ConstrainMinLuxNight: $\frac{\sum model.X[i] \cdot model.LumenNight[i]}{room\_length \cdot room\_width} \geq min\_lux\_Night$
        Define objective function: $\sum model.X[i] \cdot (model.PowerDay[i] \cdot minutes\_day \cdot price + model.PowerNight[i] \cdot minutes\_night \cdot price)$
        Solve the model using CPLEX solver
        Extract results: *num_lights*, *light_intensities*
end for
Optimization:
Apply KMeans clustering to ceiling grid points
Optimize light positions within clusters considering shadows
Output:
Optimal light positions and intensities, average illuminance, standard deviation

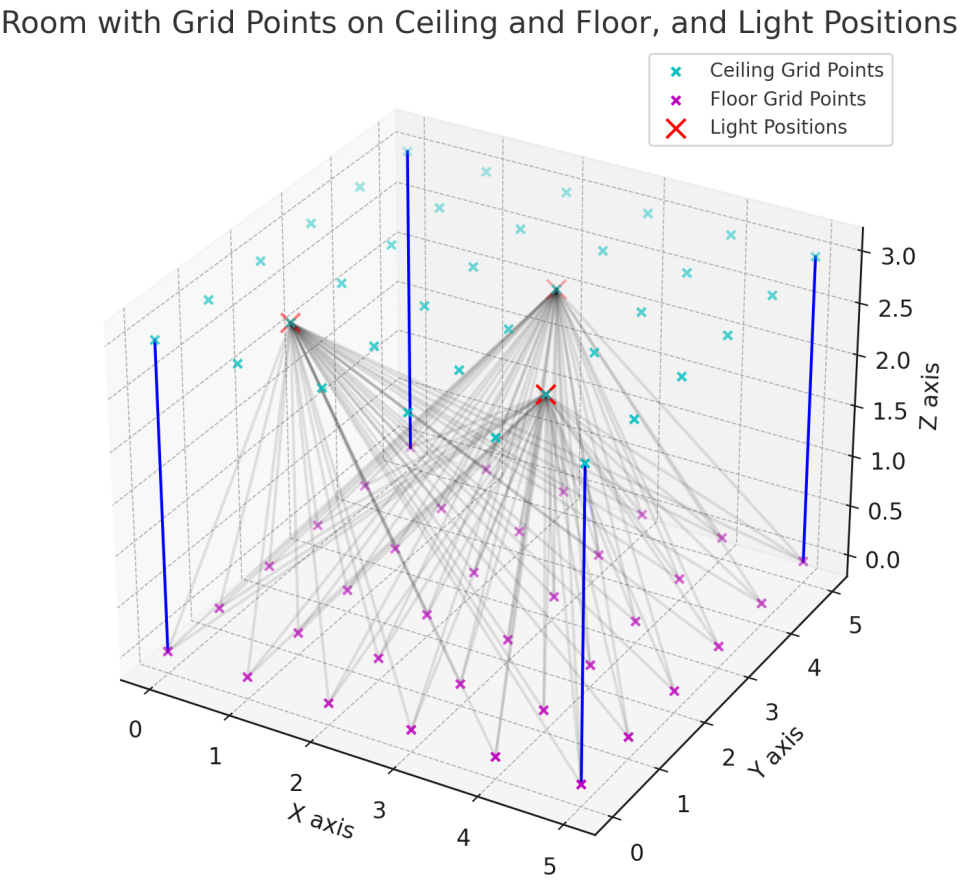


Figure 4.1: Room with grid points on the ceiling and floor, showing the light positions determined by trying all possible combinations. The ceiling and floor grids are represented by small dots, and the light positions are indicated with red markers. Light beams from the lights to the floor grid points are shown in black.

The illuminance E at a point on the floor is calculated by considering the contribution from each light source. The contribution of a light source to a floor point is inversely proportional to the square of the distance between the light source and the floor point:

$$E = \frac{I}{d^2} \tag{4.1}$$

where I is the intensity of the light source, and d is the distance between the light source and the floor point.

The total illuminance at each floor grid point is computed by summing the contributions from all light sources:

$$E_{\text{total}} = \sum_{i=1}^{n} E_i \tag{4.2}$$

where E_i is the illuminance contribution from the i-th light source and n is the total number of light sources.

The optimization problem aims to maximize the average illuminance while minimizing the standard deviation to ensure uniform lighting. The average illuminance is given by:

$$\bar{E} = \frac{1}{N} \sum_{j=1}^{N} E_j \tag{4.3}$$

where N is the total number of grid points on the floor, and E_j is the illuminance at the j-th grid point.

The standard deviation of the illuminance is calculated as follows:

$$\sigma_E = \sqrt{\frac{1}{N} \sum_{j=1}^{N} (E_j - \bar{E})^2} \tag{4.4}$$

By minimizing σ_E, the algorithm ensures that the lighting is as homogeneous as possible across the floor area.

However, another issue that we had to consider was the existence of an object within

the room since they are able to block the light so that the lights can not pass through the objects and can not reach to the points that lies on the back of the object as stated in the figure 4.2 with green rays. Thus, we had to come up with an algorithm that is responsible for detecting object collisions and calculates the shadow effects.

4.2.1 Collision Detection and Shadow Effect Calculation

In this project, the primary goal is to optimize the placement of lights in a room while accounting for the presence of objects that may cast shadows. The logic behind collision detection and shadow effect calculation is critical to achieving accurate lighting optimization. This section explains the methodology used for these calculations.

Collision Detection

Collision detection is essential to determine whether a light beam from a light source to a point on the floor intersects with any objects in the room. The process involves the following steps:

1. **Object Representation**: Each object in the room is represented as a rectangular prism defined by its lower corner coordinates, length, width, and height.
2. **Plane Creation**: The surfaces of the object are divided into planes. Each plane is defined by four vertices, which represent the corners of the rectangular face.
3. **Line-Plane Intersection**: For each light beam, represented as a line segment from the light source to a floor grid point, we check for intersection with each plane of the object. The intersection point is calculated using the following equation:

$$\mathbf{p} = \mathbf{p}_1 + t(\mathbf{p}_2 - \mathbf{p}_1) \tag{4.5}$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the endpoints of the line segment, and t is a parameter that determines the point of intersection.

4. **Point-in-Plane Check**: After calculating the intersection point, it is verified whether this point lies within the bounds of the plane. This process involves the following steps:

   (a) **Calculate the Normal Vector**: The normal vector $\mathbf{n}$ of the plane is calculated using the cross product of two vectors lying on the plane. Given three vertices $\mathbf{v}_0$, $\mathbf{v}_1$, and $\mathbf{v}_2$ of the plane, the normal vector is:

$$\mathbf{n} = (\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0) \tag{4.6}$$

   (b) **Check if the Point Lies on the Plane**: The intersection point $\mathbf{p}$ should satisfy the plane equation. This is done by ensuring the dot product of the normal vector with the vector from a point on the plane to the intersection point is zero:

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{v}_0) = 0 \tag{4.7}$$

   where $\mathbf{p}$ is the intersection point and $\mathbf{v}_0$ is a point on the plane.

   (c) **Bounded Plane Check**: To verify that the point $\mathbf{p}$ lies within the bounds of the finite plane, the following steps are taken:

   - **Parametric Representation**: Express the intersection point $\mathbf{p}$ in terms of the plane's basis vectors $\mathbf{u}$ and $\mathbf{v}$:

$$\mathbf{p} = \mathbf{v}_0 + s(\mathbf{v}_1 - \mathbf{v}_0) + t(\mathbf{v}_2 - \mathbf{v}_0) \tag{4.8}$$

     where $\mathbf{u} = \mathbf{v}_1 - \mathbf{v}_0$ and $\mathbf{v} = \mathbf{v}_2 - \mathbf{v}_0$, and s and t are scalar parameters.

   - **Range of Parameters**: The point $\mathbf{p}$ lies within the bounded plane if the scalar parameters s and t satisfy:

$$0 \leq s \leq 1 \quad \text{and} \quad 0 \leq t \leq 1 \tag{4.9}$$

If both the conditions (the point lying on the infinite plane and within the bounds of the plane) are satisfied, the intersection point **p** is considered to lie within the plane, and the light beam is considered to intersect with the object.

Shadow Effect Calculation

The shadow effect calculation quantifies the impact of shadows cast by objects on the overall illuminance at various points on the floor. The process involves the following steps:

1. **Illuminance Calculation**: The illuminance at each floor grid point is calculated by considering the contribution of each light source. The contribution of a light source to a floor point is inversely proportional to the square of the distance between the light source and the floor point:

$$E = \frac{I}{d^2} \tag{4.10}$$

   where E is the illuminance, I is the intensity of the light source, and d is the distance between the light source and the floor point.

2. **Shadow Check**: For each light source and floor grid point pair, a shadow check is performed to determine if the floor point is in the shadow of any objects. If the point is in shadow, the contribution of that light source to the illuminance at that point is set to zero.

3. **Aggregated Illuminance**: The total illuminance at each floor grid point is computed by summing the contributions from all light sources, accounting for shadows:

$$E_{\text{total}} = \sum_{i=1}^{n} E_i \tag{4.11}$$

   where E_i is the illuminance contribution from the i-th light source and n is the total number of light sources.

This methodology ensures that the placement of lights in the room is optimized, taking

into account the shadows cast by objects and providing accurate illuminance values at various points on the floor.

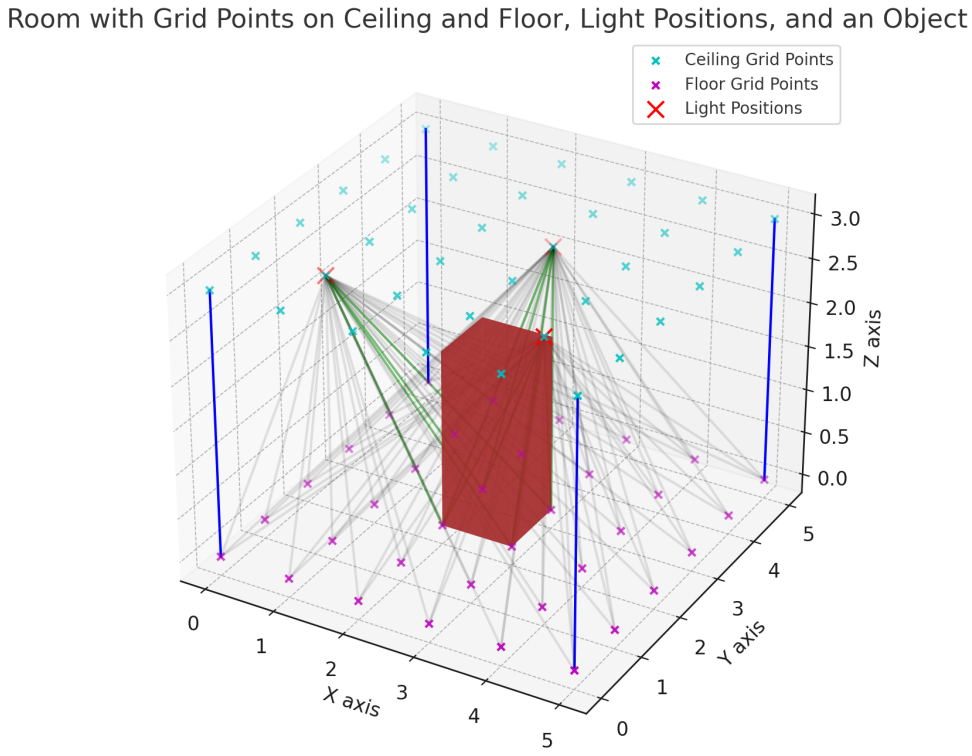


Figure 4.2: Room with grid points on the ceiling and floor, showing the light positions determined by trying all possible combinations, and an object in the room. The ceiling and floor grids are represented by small dots, the light positions are indicated with red markers, and an object is shown in dark red. Light beams from the lights to the floor grid points that collide with the object are highlighted in green, demonstrating that light cannot travel through the object, while non-colliding beams are shown in black.

4.3 Adjusting the Parameters For Different Use Cases

In order to test and apply the lighting optimization algorithm for different scenarios, several parameters can be adjusted in the code. These parameters include room dimensions, grid spacing, properties of the LED lights, and the presence of objects within the room. The flexibility of the system allows users to configure these parameters to suit various use cases.

Room Dimensions

The dimensions of the room can be easily adjusted by modifying the following parameters in the code:

- **Room Length (length)**: Defines the length of the room in meters.
- **Room Width (width)**: Defines the width of the room in meters.
- **Room Height (height)**: Defines the height of the room in meters.

By changing these parameters, the algorithm can be tested in rooms of different sizes, allowing for versatile application in various environments.

Grid Spacing

The grid spacing parameter determines the distance between grid points on the ceiling and floor. This parameter can be adjusted to control the resolution of the grid:

- **Spacing (spacing)**: Defines the distance between adjacent grid points in meters.

A smaller spacing value results in a finer grid with more points, which can lead to more precise optimization at the cost of increased computational complexity.

LED Properties

The properties of the LEDs, such as their lumen values and power consumption, are stored in separate .csv files. These files can be modified to reflect different types of LEDs or to update their characteristics. The .csv files used by the algorithm include:

- **Number of LEDs**: Specifies the number of different types of LEDs that can be used.

- **Lumen Value for Each LED (Daytime)**: Specifies the luminous flux (in lumens) for each type of LED during the day.
- **Lumen Value for Each LED (Nighttime)**: Specifies the luminous flux (in lumens) for each type of LED during the night.
- **Power Consumption (Watt per Minute) for Each LED (Daytime)**: Specifies the power consumption for each type of LED during the day.
- **Power Consumption (Watt per Minute) for Each LED (Nighttime)**: Specifies the power consumption for each type of LED during the night.

These .csv files can be edited to test the algorithm with different LED types and properties. For instance, if a user wants to test the performance of a new LED model, they can update the relevant .csv file with the new LED's specifications. This modularity makes the system highly adaptable and user-friendly.

Objects in the Room

The presence and properties of objects within the room can also be adjusted to test different scenarios. Objects are represented as rectangular prisms with specified dimensions and positions. The parameters defining the objects include:

- **Lower Corner Coordinates (lower_corner)**: Defines the (x, y, z) coordinates of the lower corner of the object.
- **Length (length)**: Defines the length of the object in meters.
- **Width (width)**: Defines the width of the object in meters.
- **Height (height)**: Defines the height of the object in meters.

By modifying these parameters, users can simulate different configurations and arrangements of objects within the room. This is crucial for evaluating how the presence of obstacles affects the lighting optimization and shadow calculations. The algorithm will take these objects into account when determining the optimal lighting positions, ensuring that the effects of shadows and blocked light paths are accurately modeled. Also, there is no such limitation about the number of objects exists in the room. You can have as many objects as you want inside the room even tough this will increase the computation complexity.

4.4 Using KMeans Clustering to Increase Computation Speed

After doing all these, we have faced with a speed problem. Even tough the algorithm worked well, the computation time was very long. Thus we had to come up with a solution.

To enhance the computation speed of the lighting optimization algorithm, KMeans clustering is employed. KMeans clustering is a method that partitions the ceiling grid points into a predefined number of clusters. This reduces the number of combinations that need to be evaluated, thereby significantly speeding up the optimization process.

KMeans clustering is used to provide a systematic approach to distribute the light positions across the ceiling. Instead of evaluating all possible positions for each light, KMeans helps in grouping the grid points into clusters, with the cluster centers representing potential light positions. This reduces the computational complexity from evaluating every possible combination to evaluating a smaller, more manageable set of cluster centers.

4.4.1 KMeans Clustering Algorithm

The KMeans clustering algorithm partitions the data into k clusters, where each cluster is represented by its centroid. The steps involved in KMeans clustering are:

1. **Initialization**: Randomly initialize k cluster centroids.

2. **Assignment**: Assign each data point to the nearest cluster centroid based on the Euclidean distance.
3. **Update**: Recalculate the centroids as the mean of all data points assigned to the cluster.
4. **Iterate**: Repeat the assignment and update steps until convergence (i.e., the centroids no longer change significantly).

Mathematically, the objective is to minimize the within-cluster sum of squares (WCSS), defined as:

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mu_i\|^2 \tag{4.12}$$

where k is the number of clusters, $\mathbf{x}$ is a data point, C_i is the i-th cluster, and μ_i is the centroid of the i-th cluster.

4.4.2 Application of KMeans in Lighting Optimization

In the context of the lighting optimization problem, the ceiling grid points are treated as the data points to be clustered. By applying KMeans clustering, the ceiling grid points are grouped into clusters, and the cluster centers are used as the initial positions for the lights. This reduces the number of combinations that needs to be evaluated, significantly speeding up the optimization process.

Figure 4.3 illustrates the ceiling grid points before and after applying KMeans clustering. In the left subplot, we see the initial grid points. In the right subplot, the grid points are grouped into clusters, with each cluster represented by a different color and the cluster centers indicated by larger markers.

Thus, after applying KMeans to the project a significant increase in the computation time have been observed. Thus, we were able to get results from more complex scenarios in shorter time periods.

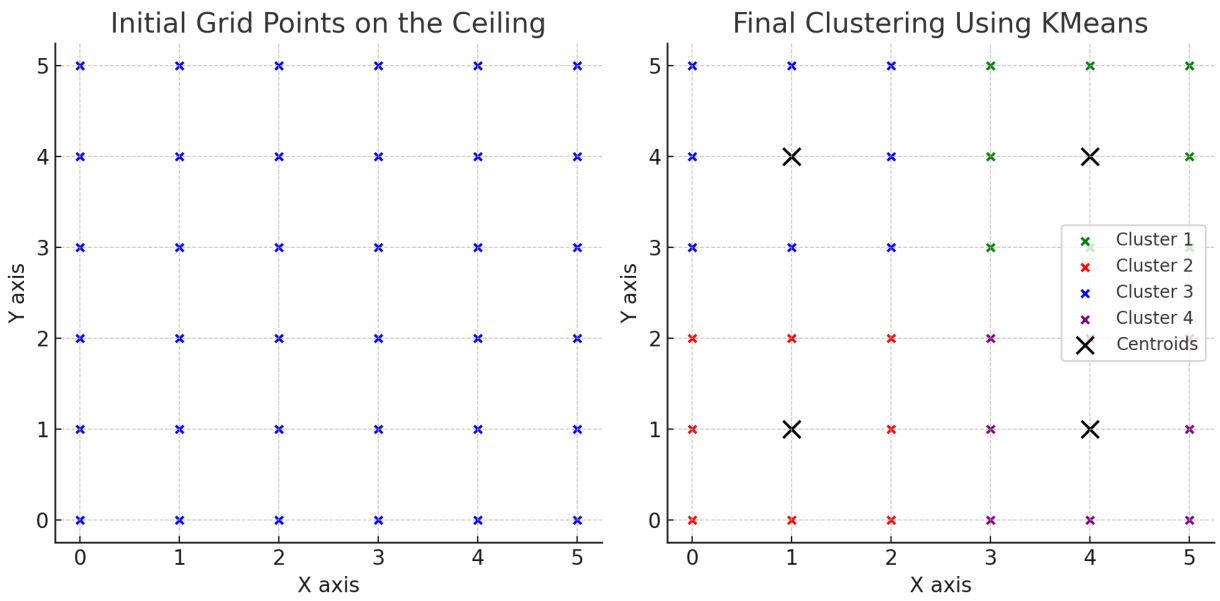


Figure 4.3: First and Final States of Clustering.

4.5 Complexity Analysis

This section provides an analysis of the runtime complexity of the lighting optimization project. The project involves several key steps, each contributing to the overall computational complexity. These steps include grid creation, KMeans clustering, illuminance calculation, shadow calculation, and optimization.

4.5.1 Grid Creation

The grid creation step involves generating a grid of points on the ceiling and floor based on the room dimensions. The time complexity of this step is:

$$O(n \times m)$$

where n is the length and m is the width of the room.

4.5.2 KMeans Clustering

The KMeans clustering step groups the ceiling grid points into clusters. The time complexity of the KMeans algorithm is:

$$O(t \times k \times n \times d)$$

where t is the number of iterations, k is the number of clusters, n is the number of data points, and d is the dimensionality of the data.

4.5.3 Illuminance Calculation

The illuminance calculation step computes the illuminance at each point on the floor grid. The time complexity of this step is:

$$O(l \times f)$$

where l is the number of lights and f is the number of floor grid points.

4.5.4 Shadow Calculation

The shadow calculation step determines whether a point on the floor is in the shadow of any objects. The time complexity of this step is:

$$O(o \times p)$$

where o is the number of objects and p is the number of planes per object.

4.5.5 Optimization

The optimization step involves trying all possible permutations of light intensities and optimizing the position of each light within its cluster. The time complexity of this step is:

$$O(p \times (l \times c + f \times l \times o \times p))$$

where p is the number of permutations of light intensities, l is the number of lights, c is the number of ceiling grid points, f is the number of floor grid points, o is the number of objects, and p is the number of planes per object.

4.5.6 Overall Time Complexity

Combining the individual complexities, the overall time complexity of the project is primarily driven by the optimization step, which is the most computationally intensive part. The overall time complexity is given by:

$$O\left(t \times k \times n \times d + p \times (l \times c + f \times l \times o \times p)\right)$$

where:

- t is the number of iterations for KMeans clustering.
- k is the number of clusters.
- n is the number of ceiling grid points.
- d is the dimensionality of the data.
- p is the number of permutations of light intensities.
- l is the number of lights.

- c is the number of ceiling grid points.
- f is the number of floor grid points.
- o is the number of objects.
- p is the number of planes per object.

4.5.7 Simplified Analysis

Assuming typical values and simplifying:

- $n \approx m$ (grid size is roughly square).
- o and p are constants (number of objects and planes per object are fixed).

The complexity can be simplified to:

$$O\left(t \times k \times n^2 + p \times (l \times n^2 + f \times l \times n^2)\right)$$

Given that $f \approx n^2$, the complexity can be further simplified to:

$$O\left(t \times k \times n^2 + p \times l \times n^4\right)$$

In the worst case, the overall time complexity is dominated by the $O(p \times l \times n^4)$ term, especially for larger grid sizes and numbers of lights.

Chapter 5

Expected Results

5.1 Introduction

In my project, I have tried some cases to get the results and understand the behaviour of the algorithm. Section 5.2 presents my experimental setup. Section 5.3 presents the results and how they tie together. Section 5.4 discusses the results.

5.2 Experimental Setup

While testing my project, I have created a room with dimensions of $3 \times 3 \times 3\,\mathrm{m}^3$. All the dimension data and the other relevant data in the .csv files are supplied to the Python code provided in the section A1 in Appendix A. The test has been run for 6 consecutive times. In the first try there are no objects present in the room. In the consecutive other 3 trials, there is one object in the room placed on the further side, middle and on the closer side of the room. In the 5^{th} trial there are 2 objects in the room, one placed in the closer side and one is placed on the further side of the room. In the 6^{th} trial there are three objects in the room one in the closer side one in the middle (but shifted by 1m) and one the further side of the room. So that, I can track the behivour of light positioning and the average illumination in the room. More details can be found in Table 5.1. However, it is important the note that the project is not limited to what I have tried. Every parameter in the project can be adjusted. The room can be bigger, there may be different illuminance value requirements, there may be different sized objects and so on. These can easily be adjusted and run. Furthermore, it is important to note that, as stated in Section 4.5, adding more objects and making the room bigger increases the computation time.

To validate my results, I have created a Python file from scratch. It basically works

Table 5.1: Experiment Setup

Setup	Room Length (m)	Room Width (m)	Room Height (m)	Spacing	Object Used	Object Position
1	3	3	3	0.5	No	N/A
2	3	3	3	0.5	Yes	Object 1: Lower corner: [1, 2, 0], Length: 1, Width: 0.5, Height: 2
3	3	3	3	0.5	Yes	Object 1: Lower corner: [1, 1, 0], Length: 1, Width: 0.5, Height: 2
4	3	3	3	0.5	Yes	Object 1: Lower corner: [1, 0, 0], Length: 1, Width: 0.5, Height: 2
5	3	3	3	0.5	Yes	Object 1: Lower corner: [1, 0, 0], Length: 1, Width: 0.5, Height: 2; Object 2: Lower corner: [1, 2, 0], Length: 1, Width: 0.5, Height: 2
6	3	3	3	0.5	Yes	Object 1: Lower corner: [1, 2, 0], Length: 1, Width: 0.5, Height: 2; Object 2: Lower corner: [1, 0, 0], Length: 1, Width: 0.5, Height: 2; Object 3: Lower corner: [0, 1, 0], Length: 1, Width: 0.5, Height: 1

in the same way like my project, however, there is no logic in it. It is a script where the user can manually define light positions, light illuminance levels, object positions and dimensions within the room. Then, the script calculates the average illuminance in the room. This is a to benchmark whether a human placed LED's in the room can create higher average illuminance in the room compared to optimization based algorithm.

As stated before, the reflections of lights are not considered throughout the project due to the LOS constraint of VLC. This situation also applies to this test script. In the test script, light reflection is not considered as well. In the calculation of average illuminance, there will be some effect to the average illuminance in the room from the reflection presumably, however, since it is not what we want in the VLC, the illuminance added up from the reflection is not considered. It is assumed that the light does not reflect throughout the project.

5.3 Results

As stated before, there were 6 different setups to test the algorithm. All of which are done in a $3 \times 3 \times 3m^3$ room with different object settings. Also, the code is able to generate a 3D room plot as well as a heat map of illuminance levels on the ground to visualize the results.

In the settings there were a total of 3 LED types with illuminance levels of 100, 300 and 500 lumens. All the other information related to each LED type are saved in seperate .csv files as stated in Table 5.2.

Table 5.2: Description of columns in the supplied CSV files

File	**Column Name**	**Description**	**Example Values**
LED.csv	LED	Identifies the LED number	1, 2, 3
LumenDay.csv	LED	Identifies the LED number	1, 2, 3
	LumenDay	Luminous flux (lumens) during the day for each LED	100, 300, 500
LumenNight.csv	LED	Identifies the LED number	1, 2, 3
	LumenNight	Luminous flux (lumens) during the night for each LED	100, 300, 500
PowerDay.csv	LED	Identifies the LED number	1, 2, 3
	PowerDay	Power consumption (Wh per minute) during the day for each LED	0.0167, 0.05, 0.083
PowerNight.csv	LED	Identifies the LED number	1, 2, 3
	PowerNight	Power consumption (Wh per minute) during the night for each LED	0.0167, 0.05, 0.083

The lighting optimization process was conducted for various setups, taking into account parameters such as minimum illuminance levels for day (`minLuxDay`) and night (`minLuxNight`) (minLuxDay was set to 100 and minLuxNight was set to 40), room dimensions, and the presence of objects. Table 5.3 presents the results of the optimization,

including the optimal positions and intensities of the lights, as well as the average illuminance and standard deviation of illuminance for each setup.

In Setup 1, the optimal positions of the lights were determined to be at coordinates (1.5, 1.0), (2.0, 1.5), and (1.0, 1.5), with corresponding light intensities of 500, 100, and 300 lumens, respectively. This configuration resulted in an average illuminance of 81.49 lux and a standard deviation of 9.36 lux. For Setup 2, the optimal light positions were found to be (1.5, 0.5), (2.5, 1.0), and (1.0, 1.5), with light intensities of 500, 300, and 100 lumens, resulting in an average illuminance of 75.65 lux and a standard deviation of 17.41 lux. Setup 3 had optimal light positions at (0.5, 1.0), (2.5, 2.0), and (1.5, 2.5), with intensities of 100, 300, and 500 lumens, yielding an average illuminance of 61.71 lux and a standard deviation of 26.84 lux.

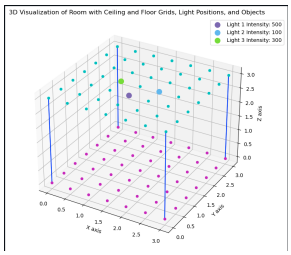
In Setup 4, the optimal light positions were (1.5, 1.0), (2.0, 2.0), and (1.5, 2.0), with intensities of 100, 300, and 500 lumens, resulting in an average illuminance of 81.46 lux and a standard deviation of 11.00 lux. For Setup 5, the optimal positions were (1.5, 1.0), (2.5, 1.5), and (0.5, 1.5), with light intensities of 500, 100, and 300 lumens, producing an average illuminance of 67.44 lux and a standard deviation of 25.85 lux. Finally, Setup 6 had optimal light positions at (1.5, 1.0), (2.5, 1.5), and (1.0, 1.5), with intensities of 300, 500, and 100 lumens, achieving an average illuminance of 67.31 lux and a standard deviation of 25.72 lux.

It is also important to note that, while running the setups, the (`spacing`) parameter in the code was set to 0.5.

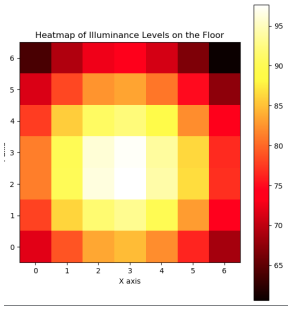
The visualisation of the results in Table 5.3 can be found in the Figure 5.1. In the heat maps, it is possible to observe the distribution of the illuminance level on the floor. Also in setup 2,5 and 6 the shadow effect is observable on the heat maps. It is important to note that, the illuminance levels at points on the floor where the objects lies on as not counted while calculating the average illuminance within the room since these values are simply zero.

Table 5.3: Lighting Optimization Results

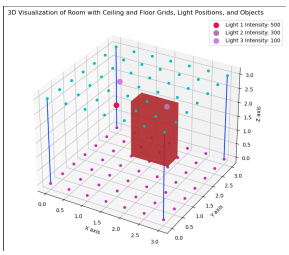
Setup	Optimal Positions	Optimal Intensities	Average Illuminance	Standard Deviation
1	(1.5, 1.0), (2.0, 1.5), (1.0, 1.5)	500, 100, 300	81.494841	9.356123
2	(1.5, 0.5), (2.5, 1.0), (1.0, 1.5)	500, 300, 100	75.647612	17.413239
3	(0.5, 1.0), (2.5, 2.0), (1.5, 2.5)	100, 300, 500	61.710801	26.843181
4	(1.5, 1.0), (2.0, 2.0), (1.5, 2.0)	100, 300, 500	81.460160	11.000592
5	(1.5, 1.0), (2.5, 1.5), (0.5, 1.5)	500, 100, 300	67.444336	25.849071
6	(1.5, 1.0), (2.5, 1.5), (1.0, 1.5)	300, 500, 100	67.306235	25.722324



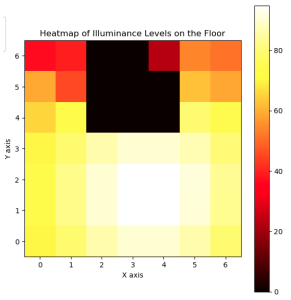
(a) Setup 1 Room



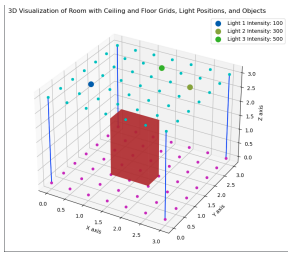
(b) Setup 1 Heatmap



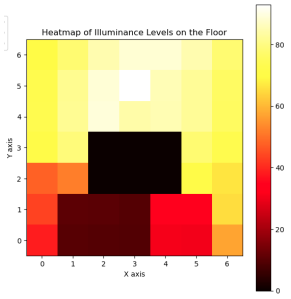
(c) Setup 2 Room



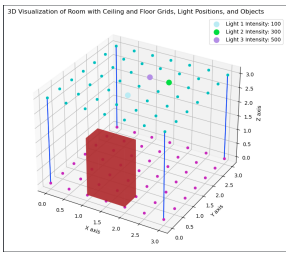
(d) Setup 2 Heatmap



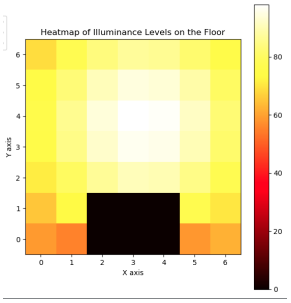
(e) Setup 3 Room



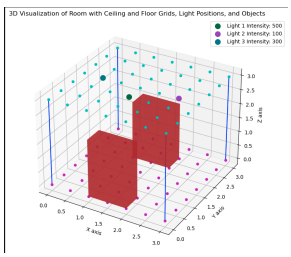
(f) Setup 3 Heatmap



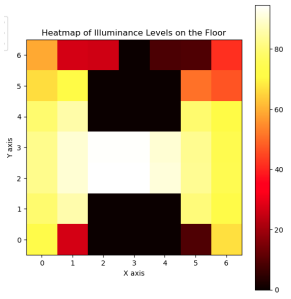
(g) Setup 4 Room



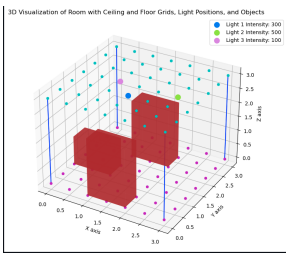
(h) Setup 4 Heatmap



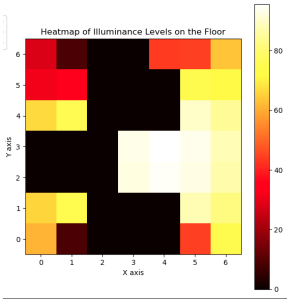
(i) Setup 5 Room



(j) Setup 5 Heatmap



(k) Setup 6 Room



(l) Setup 6 Heatmap

As stated before, an another Python script has been coded for validating the results. The validation process is based on Manual Validation. In the validation process, a user just thinks about what would be the best light positions in the room and runs the algorithm. Then, the results like average illuminance, standard deviation and so on compared with the ones found by the algorithm. Therefore, we can decide which creates better illuminance in the room.

For testing conditions to be equal, we assume that the user knows the illuminance levels of LED's and how much LED's should be in the room, otherwise, the user can put a lot of LED's in the room which will lead to a better average illuminance value. However, this would be energy inefficient and would create excessive cost.

Again, the validation test has been run in a $3 \times 3 \times \ 3m^3$ room.

Table 5.4: Comparison of Average Illuminance Levels

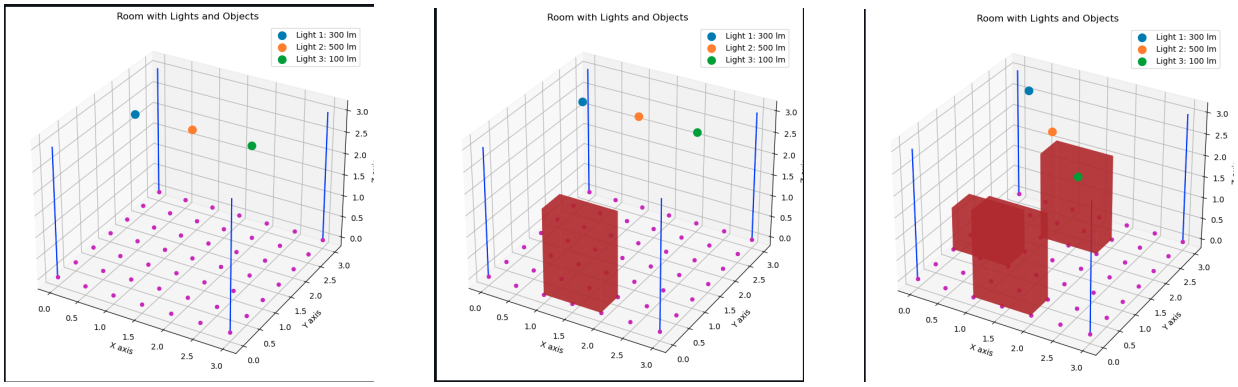
Setup	Manual Testing Avg Lux	Algorithm-based Avg Lux
1	80.53	81.49
4	70.06	81.46
6	33.35	67.31

For the manual testing of light positions, lights were placed in locations that an average human might intuitively choose. In test room 1, the lights were positioned at (0.5, 1.5), (1.5, 1.5), and (2.5, 1.5). In test room 4, the lights were placed at (0.5, 2.5), (1.5, 2.5), and (2.5, 2.5). For test room 6, the light positions were (0.5, 2.5), (1.5, 1.5), and (2.5, 0.5). These positions reflect a typical human approach to achieving even lighting across the room. The visualisation of 3D rooms for test cases can be seen in Figure 5.2

In comparison, the algorithm determined the optimal light positions for setup 1 as (1.5, 1.0), (2.0, 1.5), and (1.0, 1.5), resulting in an average illuminance of 81.49 lux. For setup 4, the algorithm selected light positions at (1.5, 1.0), (2.0, 2.0), and (1.5, 2.0), achieving an average illuminance of 81.46 lux. In setup 6, the algorithm placed the lights at (1.5, 1.0), (2.5, 1.5), and (1.0, 1.5), resulting in an average illuminance of 67.31 lux.

The manually placed light positions in test room 1 resulted in an average illuminance of 80.53 lux, which is slightly lower than the algorithm's result of 81.49 lux. For test room 4, the manual positions yielded an average illuminance of 70.06 lux, compared to the algorithm's 81.46 lux. In test room 6, the manual placements achieved an average illuminance of 33.35 lux, significantly lower than the algorithm's result of 67.31 lux.

The comparison shows that while manual positioning by a human can achieve reasonable illuminance levels, the algorithm consistently finds more optimal positions, resulting in higher average illuminance values. This highlights the effectiveness of the algorithm in optimizing light placement to achieve better illumination performance.



(a) Test 1 Room (b) Test 4 Room (c) Test 6 Room

Figure 5.2: 3D Visualizations of Test Room Setups

5.4 Inferences

The results totally support the hypotheses. As can be seen from the Section 5.3, all of the cases support our hypotheses. Thus, this is not that hard to imagine since algorithm finds the maximum average illuminance value. The only case that would have failed might have happened, when the user positions the light right in the same points where the algorithm finds. Even this could have led to the same result with the algorithm. Thus, there is no way that a user placed lighting can reach a better illuminance level compared to the algorithm based. Thus, to make a room support VLC, higher illuminance

is better. Therefore, using the algorithm would be very beneficial for creating spaces with VLC support.

Chapter 6

Appendix B: Professional and Ethical Issues

B1. Introduction

This chapter explains how the project accounts for environmental, ethical, social and economic concerns.

B2. PESTEL Analysis

Political

- **Regulations Compliance:** The project adheres to lighting standards such as the European Lighting Standard EN12464-1, ensuring compliance with political regulations regarding indoor lighting and energy usage.
- **Support for Green Policies:** By optimizing energy consumption, the project aligns with governmental policies aimed at reducing carbon footprints and promoting sustainability.
- **High-Level Purpose:** The project exists to optimize indoor VLC systems, providing energy-efficient and high-quality lighting solutions that align with political goals for energy conservation.

Economic

- **Cost Efficiency:** The project significantly reduces energy consumption, lowering operational costs for businesses and households, making it economically viable.
- **Market Potential:** Enhanced VLC systems create new market opportunities in smart lighting and communication technologies, potentially increasing business profitability.
- **Commercial Matters:** The solution is designed to be cost-effective, making it accessible for a broad range of users, from homeowners to large businesses. Initial setup costs and technical expertise requirements may be barriers to adoption.

Social

- **Health and Safety:** The solution ensures adequate lighting levels, which is crucial for the health and well-being of occupants, and enhances safety by maintaining optimal visibility in indoor environments.

- **Diversity and Inclusion:** The project is adaptable for different indoor environments, ensuring it meets diverse user needs, from residential to commercial spaces. This inclusivity caters to a wide range of cultural and societal contexts.
- **Target Audience and Beneficiaries:** The project is designed for a wide range of users, including residential households and commercial offices, benefiting building occupants with improved lighting and reduced energy costs.

Technological

- **Advanced Algorithms:** The project uses MILP models and KMeans clustering to optimize light placement and intensity, showcasing innovation in the field.
- **Integration Capability:** The system can be integrated with existing smart home and office technologies, enhancing its functionality and user-friendliness.
- **Key Enablers:** Advancements in LED technology and computational resources are critical enablers for the project, allowing for the practical implementation of the optimization algorithms.

Environmental

- **Energy Efficiency:** The optimization of light placement and intensity reduces energy consumption, contributing to lower greenhouse gas emissions and promoting environmental sustainability.
- **Sustainable Design:** The project encourages the use of energy-efficient LED lights, further supporting environmental conservation efforts.
- **Potential Positive Impacts:** The project promotes energy savings and reduced environmental impact through efficient lighting, contributing to sustainability goals.

Legal

- **Compliance with Standards:** The project complies with industry standards and codes of practice, ensuring it meets legal requirements for indoor lighting and energy efficiency, thus providing a reliable and accepted solution.
- **Data Protection:** Where applicable, the project adheres to data protection regulations, safeguarding user privacy and data security.
- **Limiting Assumptions and Risks:** The project does not currently account for real-time dynamic adjustments or advanced lighting technologies like dimmability. This limitation may affect the system's adaptability and full potential.

Future Extensions and Considerations

- **Political:** Further alignment with evolving environmental policies and standards.
- **Economic:** Exploring ways to reduce implementation costs and make the technology more accessible.
- **Social:** Enhancing user engagement and education on energy conservation benefits.
- **Technological:** Developing real-time adaptive systems and incorporating advanced features like dimmability.
- **Environmental:** Continued focus on reducing environmental impact through innovative lighting solutions.
- **Legal:** Ensuring ongoing compliance with new regulations and enhancing data security measures.

B3. Ethical Considerations

Respect for Life, Law, the Environment, and Public Good

Our solution prioritizes respect for life and public safety by ensuring that the lighting optimization promotes safe and well-illuminated indoor environments. The project adheres to relevant legal standards, including compliance with the European Lighting Standard EN12464-1, which ensures that all installations meet the necessary requirements for safety and efficiency. The focus on energy efficiency, through the use of optimized LED placement, contributes to significant reductions in energy consumption. This not only leads to cost savings but also reduces the environmental impact, supporting sustainability goals and the well-being of current and future generations. By minimizing energy usage and thus the associated carbon footprint, the project aligns with broader environmental and public good objectives.

Data Protection

Given that no personal information is gathered, stored or sent in this project, there are no data protection issues to take into account. Aims at working irrespective of the user's particulars and thus considerably reduces privacy problems. With no means for handling data, there is no chance of losses or abuses concerning it thus safeguarding one's secrecy . Therefore, all attention is geared towards improving the physical factors that affect the lumens without sacrificing individual freedoms.

B4. Inclusive Engineering Outcomes

Consideration of Stakeholder Perspectives

Our solution is designed to produce engineering outcomes that comprehensively consider the perspectives of all stakeholders, including users, installers, and facility managers. By optimizing lighting for safety, efficiency, and communication quality, the project addresses the needs and concerns of individuals in residential, commercial, and industrial settings. The use of energy-efficient LEDs and advanced algorithms ensures that the solution meets the requirements of both end-users and energy policymakers, promoting sustainability and cost-effectiveness.

Avoidance of Bias and Discrimination

The solution avoids bias and discrimination by being universally applicable and adaptable to a wide range of environments. It is designed to function effectively regardless of the socio-economic status, geographical location, or specific needs of the users. The technology can be implemented in diverse settings without favoring any particular group, ensuring fair and equitable access to advanced lighting solutions.

Consideration of Future Needs and Trends

The project considers future needs and trends by incorporating flexible and scalable technologies that can adapt to evolving lighting and communication requirements. The focus on energy efficiency and the potential integration with smart technologies positions the solution well for future advancements in the fields of sustainable building design and smart infrastructure. The use of LED technology, which is rapidly becoming a standard due to its efficiency and longevity, further ensures that the solution remains relevant as lighting technology evolves.

Accessibility and Equity

The outcomes of this project are designed to be accessible to all, regardless of the scale or budget of the installation. By optimizing energy consumption and reducing costs, the solution makes advanced lighting technology more affordable and accessible, particularly benefiting underserved communities and organizations looking to reduce their energy expenditures. The project does not require specialized or proprietary technology, further enhancing its accessibility.

Alignment with United Nations Sustainable Development Goals (SDGs)

The project aligns with several United Nations Sustainable Development Goals (SDGs), particularly:

- **SDG 7: Affordable and Clean Energy** - By improving energy efficiency and reducing consumption, the project supports the goal of ensuring access to affordable, reliable, sustainable, and modern energy for all.
- **SDG 9: Industry, Innovation, and Infrastructure** - The development and implementation of advanced lighting optimization technologies contribute to building resilient infrastructure and fostering innovation.
- **SDG 11: Sustainable Cities and Communities** - By optimizing lighting for better energy efficiency and reducing environmental impact, the project contributes to making cities and human settlements inclusive, safe, resilient, and sustainable.
- **SDG 13: Climate Action** - The reduction in energy consumption and carbon footprint directly supports efforts to combat climate change and its impacts.

These considerations ensure that the project not only addresses current needs but also contributes positively to long-term sustainable development goals and global challenges.

Conclusion

This project reflects a deep understanding of the wider professional responsibilities of engineers, encompassing not only technical excellence but also ethical, social, and environmental considerations. Beyond the traditional PESTEL analysis, the project integrates ethical and inclusive approaches to engineering by ensuring that the solutions developed are accessible, non-discriminatory, and environmentally sustainable. These approaches are critical in promoting fairness, equity, and responsibility in engineering practices.

The research aligns with several United Nations Sustainable Development Goals (SDGs), particularly by advancing SDG 7 (Affordable and Clean Energy), SDG 9 (Industry, Innovation, and Infrastructure), SDG 11 (Sustainable Cities and Communities), and SDG 13 (Climate Action). The focus on optimizing energy efficiency through advanced lighting technologies supports these goals by promoting sustainable energy use, fostering innovation in infrastructure, and contributing to the reduction of carbon emissions.

In addressing specific engineering challenges, this project abstracts to a broader context by demonstrating how technical solutions can and should be developed with a holistic view of their societal impact. By considering the long-term implications of engineering decisions, including their ethical and inclusive dimensions, this research underscores the essential role of engineers in driving positive societal change and supporting global sustainability efforts.

Chapter 7

Conclusion

This dissertation addressed the optimization of visible light communication (VLC) systems in indoor environments, hypothesizing that an algorithmic approach could better optimize LED placement and intensity than manual methods, particularly by accounting for object interference and light uniformity. The problem was modeled using a Mixed Integer Linear Program (MILP) framework, which was chosen for its robustness in handling complex constraints and variables such as power consumption and the spatial configuration of light sources. To test these hypotheses, the study employed a combination of theoretical modeling and practical experiments across various room setups, verifying the model's effectiveness against manually optimized lighting. The experiments demonstrated that the algorithm consistently achieved superior results, including higher average illuminance and improved uniformity, confirming its utility for practical applications. These findings suggest significant implications for enhancing energy efficiency and communication quality in smart lighting systems, offering a more precise and efficient method for VLC optimization.

Despite the promising results, the study had some limitations, such as not accounting for real-time changes in room configurations and the absence of considerations for dimmable LED technologies, which may affect the broader applicability of the findings. Future work should explore dynamic modeling capabilities and incorporate advanced lighting features like dimmability to further optimize energy use and adaptability. Additionally, investigating the impact of different materials and surface reflections could enhance the precision of VLC system designs, making the technology more versatile and efficient in varied settings.

References

[1] World energy climate statistics. https://yearbook.enerdata.net/total-energy/world-consumption-statistics.html, 2024. [Online; accessed 5-March-2024].

[2] Peter Robert Boyce. *Human Factors in Lighting*. CRC Press, 3 edition, 2014. eBook Published 7 April 2014.

[3] Pierre C. Energy consumption of lighting. https://www.linkedin.com/pulse/energy-consumption-lighting-pierre-crous/, 2023. [Online; accessed 5-March-2024].

[4] Maytee Zambrano Nuñez Carlos Medina. Led based visible light communication: Technology, applications and challenges – a survey. *International Journal of Advances in Engineering & Technology*, 8(4):482–495, 2015.

[5] Fabiano Cassol, Paulo Smith Schneider, Francis H.R. França, and Antônio J. Silva Neto. Multi-objective optimization as a new approach to illumination design of interior spaces. *Building and Environment*, 46(2):331–338, 2011.

[6] Hong-Gang Hao, Dan-Dan Zhang, and Shuai Tang. Analysis of the led lamp arrangement for uniformity of illumination in indoor vlc system. *J. Opt. Soc. Korea*, 18(6):663–671, Dec 2014.

[7] IBM. What does cplex do?, 2024.

[8] Evangelos-Nikolaos D. Madias, Panagiotis A. Kontaxis, and Frangiskos V. Topalis. Application of multi-objective genetic algorithms to interior lighting optimization. *Energy and Buildings*, 125:66–74, 2016.

[9] Luiz Eduardo Mendes Matheus, Alex Borges Vieira, Luiz F. M. Vieira, Marcos A. M. Vieira, and Omprakash Gnawali. Visible light communication: Concepts, applications and challenges. *IEEE Communications Surveys Tutorials*, 21(4):3204–3237, 2019.

[10] Dayu Shi, Jiacheng Li, Yourong Liu, Lina Shi, Yanqi Huang, Zhan Wang, Xun Zhang, and Andrei Vladimirescu. Effect of illumination intensity on led based visible light communication system. pages 1–4, 10 2020.

[11] Zhao Tian, Kevin Wright, and Xia Zhou. Lighting up the internet of things with darkvlc. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, HotMobile '16, page 33–38, New York, NY, USA, 2016. Association for Computing Machinery.

[12] Anagnostis Tsiatmas, Frans MJ Willems, Jean-Paul MG Linnartz, Stan Baggen, and Jan WM Bergmans. Joint illumination and visible-light communication systems: Data rates and extra power consumption. In *2015 IEEE International Conference on Communication Workshop (ICCW)*, pages 1380–1386, 2015.

[13] Yang Yang, Zhiyu Zhu, Caili Guo, and Chunyan Feng. Power efficient led placement algorithm for indoor visible light communication. *Optics Express*, 28:36389, 11 2020.

Appendix A: Raw Data

A1. Computer Code

Main Project File

```
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyomo.environ import *

from pyomo.core import (AbstractModel, Set, Param, Var, sum_product,
                        PositiveReals, NonNegativeReals, Constraint,
                        Objective, Expression, minimize)
from sklearn.cluster import KMeans
import warnings
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import random
import cplex

import os
from openpyxl import load_workbook
import uuid


def create_grid(length, width, spacing=1):
    """
    Create a grid on the ceiling or floor based on room dimensions.

    :param length: Length of the room
    :param width: Width of the room
    :param spacing: Spacing between grid points
    :return: List of grid points
    """
    x = np.arange(0, length + spacing, spacing)
    y = np.arange(0, width + spacing, spacing)
    grid = np.array(np.meshgrid(x, y)).T.reshape(-1, 2)
    return grid

def create_object(lower_corner, length, width, height=0.1):  # Default
↪   height is 0
```

```
    """
    Create a representation of a rectangular prism object in the room.

    :param lower_corner: The (x, y, z) coordinates of the lower corner of
    ↪  the object
    :param length: The length of the object
    :param width: The width of the object
    :param height: The height of the object (default is 0)
    :return: A dictionary representing the object
    """
    return {
        'lower_corner': np.array(lower_corner),
        'length': length,
        'width': width,
        'height': height
    }

def calculate_shadow_impact(light_pos, floor_grid, objects):
    """
    Calculate the impact of shadows cast by objects for a given light
    ↪  position.

    :param light_pos: The position of the light (3D)
    :param floor_grid: Grid points on the floor
    :param objects: List of objects in the room
    :return: Percentage of the floor grid not in shadow
    """
    shadowed_points = sum(is_in_shadow(floor_pos, light_pos, objects) for
    ↪  floor_pos in floor_grid)
    total_points = len(floor_grid)
    return 1 - shadowed_points / total_points  # Percentage of
    ↪  non-shadowed floor area



def point_in_plane(point, plane):
    """
    Determine if a given point is within a plane defined by three points.

    This function calculates whether a point lies within the bounds of a
    ↪  plane.
    The plane is defined by three points (forming two vectors on the
    ↪  plane), and the check involves:
```

```
    1. Determining if the point is on the plane by checking if the vector
    ↪  from the plane to the point
       is orthogonal to the normal vector of the plane.
    2. Checking if the point lies within the rectangular bounds defined
    ↪  by the plane's vertices.

    Args:
        point (numpy.ndarray): A 3D point represented as [x, y, z].
        plane (list[numpy.ndarray]): A list of three 3D points (each as
        ↪  [x, y, z]) that define the plane.

    Returns:
        bool: True if the point is within the plane, False otherwise.
    """

    normal_vector = np.cross((plane[1] - plane[0]), (plane[2] - plane[0]))
    vector_to_point = point - plane[0]
    dot_product = np.dot(vector_to_point, normal_vector)
    epsilon = 1e-6  # Tolerance for checking if point is in plane

    if abs(dot_product) > epsilon:  # Adjusted check
        return False  # Point is not in the plane

    # Check if point is within the bounds of the plane
    plane_x = [p[0] for p in plane]
    plane_y = [p[1] for p in plane]
    plane_z = [p[2] for p in plane]
    return (min(plane_x) <= point[0] <= max(plane_x) and
            min(plane_y) <= point[1] <= max(plane_y) and
            min(plane_z) <= point[2] <= max(plane_z))


def create_planes(obj):
    """
    Defines the planes of an object as an array of four vertices for each
    ↪  plane.

    Args:
        obj: A dictionary representing the object with keys
        ↪  'lower_corner', 'length', 'width', and 'height'.

    Returns:
        An array of planes, where each plane is represented by four
        ↪  vertices (x, y, z).
```

```
    """

    lower_corner = obj['lower_corner']
    length = obj['length']
    width = obj['width']
    height = obj['height']

    # Define vertices for each plane (bottom, top, front, back, right,
    ↪  left)
    vertices = np.array([
        [lower_corner[0], lower_corner[1], lower_corner[2]],
        [lower_corner[0] + length, lower_corner[1], lower_corner[2]],
        [lower_corner[0] + length, lower_corner[1] + width,
        ↪  lower_corner[2]],
        [lower_corner[0], lower_corner[1] + width, lower_corner[2]],
        [lower_corner[0], lower_corner[1], lower_corner[2] + height],
        [lower_corner[0] + length, lower_corner[1], lower_corner[2] +
        ↪  height],
        [lower_corner[0] + length, lower_corner[1] + width,
        ↪  lower_corner[2] + height],
        [lower_corner[0], lower_corner[1] + width, lower_corner[2] +
        ↪  height]
    ])

    # Define planes as sets of four vertices for each face
    planes = [
        vertices[[0, 1, 2, 3]],
        vertices[[4, 5, 6, 7]],
        vertices[[0, 1, 5, 4]],
        vertices[[2, 3, 7, 6]],
        vertices[[1, 2, 6, 5]],
        vertices[[0, 3, 7, 4]]
    ]

    return planes

def line_intersects_plane(p1, p2, plane):

    """
    Determine if a line segment intersects with a plane and find the
    ↪  intersection point.

    This function checks if the line segment defined by points p1 and p2
    ↪  intersects with a plane defined by three points.
```

```
    It calculates the intersection point using vector and plane
    ↪ equations.

    Args:
        p1 (numpy.ndarray): The starting point of the line segment [x, y,
        ↪ z].
        p2 (numpy.ndarray): The ending point of the line segment [x, y,
        ↪ z].
        plane (list[numpy.ndarray]): Three 3D points defining the plane.

    Returns:
        numpy.ndarray or None: The intersection point if it exists;
        ↪ otherwise, None.
    """

    p1 = np.array(p1)
    p2 = np.array(p2)
    direction_vector = p2 - p1
    normal_vector = np.cross((plane[1] - plane[0]), (plane[2] - plane[0]))

    dot_product = np.dot(direction_vector, normal_vector)
    epsilon = 1e-6  # A small tolerance value

    if abs(dot_product) < epsilon:  # Adjusted check for parallelism
        return None  # Line is parallel or almost parallel to the plane

    t = -np.dot(normal_vector, p1 - plane[0]) / dot_product
    if 0 <= t <= 1:
        return p1 + t * direction_vector
    else:
        return None


def is_in_shadow(floor_pos, light_pos, objects):
    """
    Determine if a point on the floor is in the shadow of any objects in
    ↪ the room.

    This function checks each object in the room to determine if a line
    ↪ from the light source to the floor point intersects with it,
    indicating the point is in shadow.

    Args:
```

```
        floor_pos (tuple): The x, y coordinates of the point on the
        ↪ floor.
        light_pos (tuple): The x, y, z coordinates of the light source.
        objects (list[dict]): A list of objects (dicts) in the room, each
        ↪ defined by its lower corner, length, width, and height.

    Returns:
        bool: True if the point is in shadow, False otherwise.
    """
    floor_x, floor_y = floor_pos
    light_x, light_y, light_z = light_pos

    for obj in objects:
        obj_planes = create_planes(obj)
        for plane in obj_planes:
            intersection = line_intersects_plane([light_x, light_y,
            ↪ light_z], [floor_x, floor_y, 0], plane)
            if intersection is not None and point_in_plane(intersection,
            ↪ plane):
                return True
    return False




def apply_kmeans_clustering(ceiling_grid, num_clusters):
    """
    Apply KMeans clustering algorithm to group ceiling grid points.

    This function uses the KMeans clustering algorithm to group grid
    ↪ points on the ceiling into clusters.
    This is typically used to determine initial positions for lights in a
    ↪ room.

    Args:
        ceiling_grid (numpy.ndarray): Grid points on the ceiling.
        num_clusters (int): The number of clusters (or lights) to form.

    Returns:
        numpy.ndarray: The centers of the formed clusters.
    """
    kmeans = KMeans(n_clusters=num_clusters,
    ↪ random_state=0).fit(ceiling_grid)
    cluster_centers = kmeans.cluster_centers_
```

```
    return cluster_centers


def calculate_illuminance(light_positions, light_intensities, floor_grid,
↪  ceiling_height, objects):
    """
    Calculate illuminance at each floor grid point considering shadows
    ↪  from objects.

    :param light_positions: Positions of the lights (2D)
    :param light_intensities: Intensities of the lights
    :param floor_grid: Grid points on the floor
    :param ceiling_height: Height of the ceiling
    :param objects: List of objects in the room
    :return: Array of illuminance values at each floor grid point
    """
    illuminance_floor = np.zeros(len(floor_grid))

    for light_pos, light_intensity in zip(light_positions,
    ↪  light_intensities):
        # Add the ceiling height as the z-coordinate for the light
        ↪  position
        light_pos_3D = np.array([light_pos[0], light_pos[1],
        ↪  ceiling_height])

        for i, floor_pos in enumerate(floor_grid):
            if not is_in_shadow(floor_pos, light_pos_3D, objects):
                distance = np.sqrt((light_pos_3D[0] - floor_pos[0])**2 +
                ↪  (light_pos_3D[1] - floor_pos[1])**2 +
                ↪  ceiling_height**2)
                illuminance_floor[i] += light_intensity / (distance**2)

    return illuminance_floor


def optimize_within_cluster(cluster_center, ceiling_grid, floor_grid,
↪  ceiling_height, light_intensity, objects):
    """
    Optimize the position of a light within its cluster to maximize
    ↪  illuminance while considering shadows.

    This function searches within a specified radius around a cluster
    ↪  center to find the best position for a light.
```

```
    It evaluates each potential position based on average illuminance,
    ↪ standard deviation of illuminance,
    and shadow impact, choosing the position that optimizes these
    ↪ factors.

    Args:
        cluster_center (numpy.ndarray): The center of the cluster (3D
        ↪ point).
        ceiling_grid (numpy.ndarray): Grid points on the ceiling.
        floor_grid (numpy.ndarray): Grid points on the floor.
        ceiling_height (float): Height of the ceiling.
        light_intensity (float): Intensity of the light.
        objects (list[dict]): List of objects in the room affecting
        ↪ shadows.

    Returns:
        numpy.ndarray: The best position for the light within the
        ↪ cluster.
    """

    best_position = None
    best_illuminance = float('-inf')
    best_std_dev = float('inf')
    best_shadow_impact = float('-inf')  # Initialize the best shadow
    ↪ impact

    search_radius = max(1, spacing)

    for point in ceiling_grid:
        cluster_center_2d = cluster_center[:2]
        if np.linalg.norm(point[:2] - cluster_center_2d) <= search_radius:
            light_pos_3D = np.array([point[0], point[1], ceiling_height])
            illuminance = calculate_illuminance([point],
            ↪ [light_intensity], floor_grid, ceiling_height, objects)
            avg_illuminance = np.mean(illuminance)
            std_dev = np.std(illuminance)
            shadow_impact = calculate_shadow_impact(light_pos_3D,
            ↪ floor_grid, objects)

            # Check for improvements considering shadow impact
            if (avg_illuminance > best_illuminance or
                (avg_illuminance == best_illuminance and std_dev <
                ↪ best_std_dev) or
```

```
                (avg_illuminance == best_illuminance and std_dev ==
                ↪  best_std_dev and shadow_impact > best_shadow_impact)):
                best_illuminance = avg_illuminance
                best_std_dev = std_dev
                best_shadow_impact = shadow_impact
                best_position = point

    return best_position



def find_optimal_lighting_setup(ceiling_grid, floor_grid, num_lights,
↪  ceiling_height, light_intensities, objects):
    """
    Find the optimal setup of multiple lights in the room considering
    ↪  shadows, illuminance, and permutations of light intensities.

    This function first applies KMeans clustering to determine initial
    ↪  positions for the lights.
    It then iterates through permutations of light intensities,
    ↪  optimizing the position of each light within its cluster
    to maximize overall illuminance and minimize shadow impact. The best
    ↪  combination of positions and intensities
    that yields the highest average illuminance is selected.

    Args:
        ceiling_grid (numpy.ndarray): Grid points on the ceiling.
        floor_grid (numpy.ndarray): Grid points on the floor.
        num_lights (int): Number of lights to be placed.
        ceiling_height (float): Height of the ceiling.
        light_intensities (list[float]): List of intensities for each
        ↪  light.
        objects (list[dict]): List of objects in the room affecting
        ↪  shadows.

    Returns:
        list[numpy.ndarray]: Optimal positions for each light.
        list[float]: Corresponding intensities for each light.
        float: Highest average illuminance achieved.
        float: Standard deviation of illuminance for the optimal setup.
    """
    best_overall_positions = []
    best_overall_intensities = []
```

```
best_overall_avg_illuminance = float('-inf')
best_overall_std_dev = float('inf')

# Apply clustering to find initial light positions
cluster_centers = apply_kmeans_clustering(ceiling_grid, num_lights)
cluster_centers_3D = [np.append(center, ceiling_height) for center in
↪  cluster_centers]

# Iterate through all permutations of light intensities
for permutation in itertools.permutations(light_intensities):
    current_positions = []
    for i, cluster_center in enumerate(cluster_centers_3D):
        # Optimize position for each light within its cluster
        best_position = optimize_within_cluster(cluster_center,
        ↪  ceiling_grid, floor_grid, ceiling_height, permutation[i],
        ↪  objects)
        if best_position is not None:
            current_positions.append(best_position)

    # Calculate average illuminance and standard deviation for these
    ↪  positions
    if current_positions:
        illuminance = calculate_illuminance(current_positions,
        ↪  permutation, floor_grid, ceiling_height, objects)

        # Filter out grid points with zero illuminance
        valid_illuminance = [ill for ill in illuminance if ill > 0]

        if valid_illuminance:
            avg_illuminance = np.mean(valid_illuminance)
            std_dev = np.std(valid_illuminance)
        else:
            avg_illuminance = 0
            std_dev = 0

        # Update if this permutation yields a better result
        if avg_illuminance > best_overall_avg_illuminance or
        ↪  (avg_illuminance == best_overall_avg_illuminance and
        ↪  std_dev < best_overall_std_dev):
            best_overall_positions = current_positions
            best_overall_intensities = permutation
            best_overall_avg_illuminance = avg_illuminance
            best_overall_std_dev = std_dev
```

```
    return best_overall_positions, list(best_overall_intensities),
    ↪  best_overall_avg_illuminance, best_overall_std_dev
def plot_light_positions_and_heatmap(room_length, room_width, room_height,
↪  light_positions, light_intensities, ceiling_grid, floor_grid, spacing,
↪  objects):
    """
    Plot the best light positions in a 3D representation of the room and
    ↪  a heatmap of illuminance levels on the floor.

    :param room_length: Length of the room
    :param room_width: Width of the room
    :param room_height: Height of the room
    :param light_positions: Positions of the best lights
    :param light_intensities: Array of illuminance levels for all lights
    :param ceiling_grid: Grid points on the ceiling
    :param floor_grid: Grid points on the floor
    :param spacing: Spacing between grid points
    :param objects: List of objects in the room
    """

    # Calculate the illuminance for the floor grid
    illuminance_floor = calculate_illuminance(light_positions,
    ↪  light_intensities, floor_grid, room_height, objects)

    # Reshape the illuminance array to match the floor grid dimensions
    grid_length = int(np.ceil(room_length / spacing) + 1)
    grid_width = int(np.ceil(room_width / spacing) + 1)
    illuminance_reshape = illuminance_floor.reshape(grid_length,
    ↪  grid_width)

    # Create the 3D plot for room and light positions
    fig = plt.figure(figsize=(12, 6))
    ax1 = fig.add_subplot(121, projection='3d')

    # Plot the room boundaries
    for x in [0, room_length]:
        for y in [0, room_width]:
            ax1.plot([x, x], [y, y], [0, room_height], color="b")

    # Plot the ceiling and the floor
    ceiling_floor_x = [0, room_length, room_length, 0, 0]
    ceiling_floor_y = [0, 0, room_width, room_width, 0]
    ax1.plot(ceiling_floor_x, ceiling_floor_y, 0, color="g")
    ax1.plot(ceiling_floor_x, ceiling_floor_y, room_height, color="r")
```

```
    # Generate random colors for each light
    colors = ["#" + ''.join([random.choice('0123456789ABCDEF') for _ in
    ↪  range(6)]) for _ in range(len(light_positions))]

    # Plot the light positions with random colors and unique labels
    for i, (pos, intensity, color) in enumerate(zip(light_positions,
    ↪  light_intensities, colors)):
        ax1.scatter(pos[0], pos[1], room_height, color=color, s=100,
        ↪  label=f'Light {i+1} Intensity: {intensity}')

    # Create custom legend
    handles, labels = ax1.get_legend_handles_labels()
    ax1.legend(handles, labels, loc='upper right', bbox_to_anchor=(1.15,
    ↪  1))

    ax1.set_xlabel('X axis')
    ax1.set_ylabel('Y axis')
    ax1.set_zlabel('Z axis')
    ax1.set_title('3D Visualization of Light Positions in Room')

    # Create the 2D heatmap for illuminance levels on the floor
    ax2 = fig.add_subplot(122)
    heatmap = ax2.imshow(illuminance_reshape.T, cmap='hot',
    ↪  interpolation='nearest', origin='lower')
    ax2.set_title('Heatmap of Illuminance Levels on the Floor')
    ax2.set_xlabel('X axis')
    ax2.set_ylabel('Y axis')
    plt.colorbar(heatmap, ax=ax2, orientation='vertical')

    plt.tight_layout()
    plt.show()


def plot_room_with_grids_and_lights(room_length, room_width, room_height,
↪  ceiling_grid, floor_grid, light_positions, light_intensities,
↪  objects):
    """
    Plot the ceiling grid, floor grid, and best light positions in a 3D
    ↪  representation of the room along with objects.

    :param room_length: Length of the room
    :param room_width: Width of the room
    :param room_height: Height of the room
```

```
    :param ceiling_grid: Grid points on the ceiling
    :param floor_grid: Grid points on the floor
    :param light_positions: Positions of the best lights
    :param light_intensities: Intensities of the lights
    :param objects: List of objects in the room
    """
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Plot the room boundaries
    for x in [0, room_length]:
        for y in [0, room_width]:
            ax.plot([x, x], [y, y], [0, room_height], color="b")

    # Plot the ceiling and the floor grids
    for pos in ceiling_grid:
        ax.scatter(pos[0], pos[1], room_height, color='c', s=20)  #
        ↪ Ceiling grid points
    for pos in floor_grid:
        ax.scatter(pos[0], pos[1], 0, color='m', s=20)  # Floor grid
        ↪ points

    for obj in objects:
        lower_corner = obj['lower_corner']
        length = obj['length']
        width = obj['width']
        height = obj['height']

        # Vertices of the object
        vertices = np.array([
            [0, 0, 0],
            [length, 0, 0],
            [length, width, 0],
            [0, width, 0],
            [0, 0, height],
            [length, 0, height],
            [length, width, height],
            [0, width, height]
        ]) + lower_corner

        # Faces of the object
        faces = [
            [vertices[0], vertices[1], vertices[2], vertices[3]],  #
            ↪ Bottom
```

```
            [vertices[4], vertices[5], vertices[6], vertices[7]],  # Top
            [vertices[0], vertices[1], vertices[5], vertices[4]],  # Front
            [vertices[2], vertices[3], vertices[7], vertices[6]],  # Back
            [vertices[1], vertices[2], vertices[6], vertices[5]],  # Right
            [vertices[0], vertices[3], vertices[7], vertices[4]]   # Left
        ]

        obj_poly3d = Poly3DCollection(faces, alpha=0.7,
        ↪  facecolors='brown')
        ax.add_collection3d(obj_poly3d)

    # Generate random colors for each light
    colors = ["#" + ''.join([random.choice('0123456789ABCDEF') for _ in
    ↪  range(6)]) for _ in range(len(light_positions))]

    # Plot the light positions with random colors and unique labels
    for i, (pos, intensity, color) in enumerate(zip(light_positions,
    ↪  light_intensities, colors)):
        ax.scatter(pos[0], pos[1], room_height, color=color, s=100,
        ↪  label=f'Light {i+1} Intensity: {intensity}')

    # Create custom legend
    handles, labels = ax.get_legend_handles_labels()
    ax.legend(handles, labels, loc='upper right', bbox_to_anchor=(1.15,
    ↪  1))

    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('Z axis')
    plt.title('3D Visualization of Room with Ceiling and Floor Grids,
    ↪  Light Positions, and Objects')
    plt.show()


# Ignore future warnings that might clutter the output
warnings.simplefilter(action='ignore', category=FutureWarning)

# Define some objects in the room with their positions and dimensions
object1 = create_object(lower_corner=(1, 2, 0), length=1, width=0.5,
↪  height=2)  # Example object
object2 = create_object(lower_corner=(1, 0, 0), length=1, width=0.5,
↪  height=2)  # Example object
object3 = create_object(lower_corner=(0, 1, 0), length=1, width=0.5,
↪  height=1)  # Example object
```

```
objects = [object1, object2, object3]

# Room and lighting specifications
room_length = 3  # Length of the room in meters
room_width = 3  # Width of the room in meters
room_height = 3  # Height of the room in meters
min_lux_Day = 100  # Minimum required lux during the day
min_lux_Night = 40  # Minimum required lux during the night

# Time and cost details
minutes_day = 600  # Minutes of daylight usage
minutes_night = 360  # Minutes of night usage
price = 0.5  # Price per unit of power
spacing = 0.5  # Spacing for grid creation

# Calculate the number of grid points based on room dimensions and
↪  spacing
grid_points = ((room_width + 1) * (room_length + 1)) * spacing

# Create an abstract model for optimization
model = AbstractModel()
model.Bulbs = Set()  # Set of bulbs
model.PowerDay = Param(model.Bulbs, within=NonNegativeReals)  # Power
↪  consumption during the day
model.PowerNight = Param(model.Bulbs, within=NonNegativeReals)  # Power
↪  consumption during the night
model.LumenDay = Param(model.Bulbs, within=NonNegativeReals)  # Lumens
↪  produced during the day
model.LumenNight = Param(model.Bulbs, within=NonNegativeReals)  # Lumens
↪  produced during the night
model.X = Var(model.Bulbs, bounds=(0, grid_points),
↪  within=NonNegativeIntegers, initialize=0)  # Number of bulbs

# Constraint: Total bulbs should not exceed the number of grid points
def ConstrainGridCapacity_rule(model):
    return sum(model.X[i] for i in model.Bulbs) <= grid_points

model.ConstrainGridCapacity = Constraint(rule=ConstrainGridCapacity_rule)

# Constraint: Minimum lux requirement during the day
def ConstrainMinLuxDay_rule(model):
    return sum(model.X[i] * model.LumenDay[i] for i in model.Bulbs) /
    ↪  (room_length * room_width) >= min_lux_Day
```

```
model.ConstrainMinLuxDay = Constraint(rule=ConstrainMinLuxDay_rule)

# Constraint: Minimum lux requirement during the night
def ConstrainMinLuxNight_rule(model):
    return sum(model.X[i] * model.LumenNight[i] for i in model.Bulbs) /
    ↪  (room_length * room_width) >= min_lux_Night

model.ConstrainMinLuxNight = Constraint(rule=ConstrainMinLuxNight_rule)

# Objective: Minimize total power consumption cost
def TotalPower_rule(model):
    return sum(model.X[i] * model.PowerDay[i] * minutes_day * price +
               model.X[i] * model.PowerNight[i] * minutes_night * price
               for i in model.Bulbs)

model.PowerObjective = Objective(rule=TotalPower_rule, sense=minimize)

# Load data for the model parameters
data = DataPortal()
data.load(filename="Bulbs.csv", set=model.Bulbs)
data.load(filename="PowerDay.csv", param=model.PowerDay)
data.load(filename="PowerNight.csv", param=model.PowerNight)
data.load(filename="LumenDay.csv", param=model.LumenDay)
data.load(filename="LumenNight.csv", param=model.LumenNight)

# Create an instance of the model with loaded data
instance = model.create_instance(data)

# Specify the solver and solve the model
solver = SolverFactory('cplex',
↪  executable='/Applications/CPLEX_Studio_Community2211/cplex/bin/arm64_osx/cplex')
solver.solve(instance)

# Calculate the total number of lights used and store their light
↪  intensities
num_lights = 0
for i in instance.Bulbs:
    num_lights = round(num_lights + instance.X[i].value)
    print(instance.X[i].value)

light_intensities = []
for i in instance.Bulbs:
    for j in range(round(instance.X[i].value)):
        light_intensities.append(instance.LumenDay[i])
```

```
# Create grid for ceiling and floor for optimal placement of lights
ceiling_grid = create_grid(room_length, room_width, spacing)
floor_grid = create_grid(room_length, room_width, spacing)

# Find the optimal lighting setup
optimal_positions, optimal_intensities, optimal_avg_illuminance,
↪  optimal_std_dev = find_optimal_lighting_setup(
    ceiling_grid, floor_grid, num_lights, room_height, light_intensities,
    ↪  objects)

# Print the optimal light positions and associated metrics
print("Best light positions:", optimal_positions)
print("Best light intensities:", optimal_intensities)
print("Average Illuminance:", optimal_avg_illuminance)
print("Standard Deviation of Illuminance:", optimal_std_dev)

# Prepare results for logging or further analysis
new_results = {
    "Room Length (m)": room_length,
    "Room Width (m)": room_width,
    "Room Height (m)": room_height,
    "Spacing": spacing,
    "Number of Lights": num_lights,
    "Optimal Positions": str(optimal_positions),
    "Optimal Intensities": str(optimal_intensities),
    "Average Illuminance": optimal_avg_illuminance,
    "Standard Deviation of Illuminance": optimal_std_dev,
    "Object Used": "Yes" if objects else "No",
    "Object Position": str(objects) if objects else "N/A"
}

# File path to save the results
file_path = "lighting_results.xlsx"

# Check if the results file exists and append new data or create a new
↪  file
if os.path.exists(file_path):
    # Load existing data without modifying it
    existing_df = pd.read_excel(file_path)
    # Append new results
    results_df = pd.concat([existing_df, pd.DataFrame([new_results])],
    ↪  ignore_index=True)
else:
```

```
    results_df = pd.DataFrame([new_results])

# Convert any NaN or Inf values to strings to avoid xlsxwriter errors
results_df = results_df.replace({pd.NA: 'NaN', pd.NaT: 'NaN',
↪  float('inf'): 'Inf', float('-inf'): '-Inf'})

# Save results to the Excel file using xlsxwriter to support images
results_df.to_excel(file_path, index=False)

# Plot the room layout with grids and optimal light placements
plot_room_with_grids_and_lights(room_length, room_width, room_height,
↪  ceiling_grid, floor_grid, optimal_positions, optimal_intensities,
↪  objects)

# Plot the light positions and illuminance heatmap
plot_light_positions_and_heatmap(room_length, room_width, room_height,
↪  optimal_positions, optimal_intensities, ceiling_grid, floor_grid,
↪  spacing, objects)
```

Tester File

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

# Function to create a grid of points in a room based on given length,
↪  width, and spacing
def create_grid(length, width, spacing=1):
    x = np.arange(0, length + spacing, spacing)
    y = np.arange(0, width + spacing, spacing)
    grid = np.array(np.meshgrid(x, y)).T.reshape(-1, 2)
    return grid

# Function to define an object in the room with its position and
↪  dimensions
def create_object(lower_corner, length, width, height=0.1):
    return {
        'lower_corner': np.array(lower_corner),
        'length': length,
        'width': width,
        'height': height
    }
```

```
# Function to calculate the illuminance at each point on the floor grid
def calculate_illuminance(light_positions, light_intensities, floor_grid,
↪  ceiling_height, objects):
    illuminance_floor = np.zeros(len(floor_grid))

    for light_pos, light_intensity in zip(light_positions,
    ↪  light_intensities):
        light_pos_3D = np.array([light_pos[0], light_pos[1],
        ↪  ceiling_height])

        for i, floor_pos in enumerate(floor_grid):
            if not is_in_shadow(floor_pos, light_pos_3D, objects):
                distance = np.sqrt((light_pos_3D[0] - floor_pos[0])**2 +
                                   (light_pos_3D[1] - floor_pos[1])**2 +
                                   ↪  ceiling_height**2)
                illuminance_floor[i] += light_intensity / (distance**2)

    return illuminance_floor

# Function to determine if a point on the floor is in shadow from the
↪  light source
def is_in_shadow(floor_pos, light_pos, objects):
    floor_x, floor_y = floor_pos
    light_x, light_y, light_z = light_pos

    for obj in objects:
        obj_planes = create_planes(obj)
        for plane in obj_planes:
            intersection = line_intersects_plane([light_x, light_y,
            ↪  light_z], [floor_x, floor_y, 0], plane)
            if intersection is not None and point_in_plane(intersection,
            ↪  plane):
                return True
    return False

# Function to create the planes (faces) of an object in the room
def create_planes(obj):
    lower_corner = obj['lower_corner']
    length = obj['length']
    width = obj['width']
    height = obj['height']

    vertices = np.array([
```

```
        [lower_corner[0], lower_corner[1], lower_corner[2]],
        [lower_corner[0] + length, lower_corner[1], lower_corner[2]],
        [lower_corner[0] + length, lower_corner[1] + width,
        ↪ lower_corner[2]],
        [lower_corner[0], lower_corner[1] + width, lower_corner[2]],
        [lower_corner[0], lower_corner[1], lower_corner[2] + height],
        [lower_corner[0] + length, lower_corner[1], lower_corner[2] +
        ↪ height],
        [lower_corner[0] + length, lower_corner[1] + width,
        ↪ lower_corner[2] + height],
        [lower_corner[0], lower_corner[1] + width, lower_corner[2] +
        ↪ height]
    ])

    planes = [
        vertices[[0, 1, 2, 3]],  # Bottom face
        vertices[[4, 5, 6, 7]],  # Top face
        vertices[[0, 1, 5, 4]],  # Front face
        vertices[[2, 3, 7, 6]],  # Back face
        vertices[[1, 2, 6, 5]],  # Right face
        vertices[[0, 3, 7, 4]]   # Left face
    ]

    return planes

# Function to check if a line from a light to a floor point intersects a
↪ plane (object face)
def line_intersects_plane(p1, p2, plane):
    p1 = np.array(p1)
    p2 = np.array(p2)
    direction_vector = p2 - p1
    normal_vector = np.cross((plane[1] - plane[0]), (plane[2] - plane[0]))

    dot_product = np.dot(direction_vector, normal_vector)
    epsilon = 1e-6

    if abs(dot_product) < epsilon:
        return None

    t = -np.dot(normal_vector, p1 - plane[0]) / dot_product
    if 0 <= t <= 1:
        return p1 + t * direction_vector
    else:
        return None
```

```
# Function to check if a point lies within a plane (object face)
def point_in_plane(point, plane):
    normal_vector = np.cross((plane[1] - plane[0]), (plane[2] - plane[0]))
    vector_to_point = point - plane[0]
    dot_product = np.dot(vector_to_point, normal_vector)
    epsilon = 1e-6

    if abs(dot_product) > epsilon:
        return False

    plane_x = [p[0] for p in plane]
    plane_y = [p[1] for p in plane]
    plane_z = [p[2] for p in plane]
    return (min(plane_x) <= point[0] <= max(plane_x) and
            min(plane_y) <= point[1] <= max(plane_y) and
            min(plane_z) <= point[2] <= max(plane_z))

# Function to plot the room with objects and lights in 3D
def plot_room(room_length, room_width, room_height, light_positions,
↪   light_intensities, floor_grid, spacing, objects):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Plot the room boundaries
    for x in [0, room_length]:
        for y in [0, room_width]:
            ax.plot([x, x], [y, y], [0, room_height], color="b")

    # Plot the floor grid points
    for pos in floor_grid:
        ax.scatter(pos[0], pos[1], 0, color='m', s=20)

    # Plot the objects
    for obj in objects:
        lower_corner = obj['lower_corner']
        length = obj['length']
        width = obj['width']
        height = obj['height']

        vertices = np.array([
            [0, 0, 0],
            [length, 0, 0],
            [length, width, 0],
```

```
            [0, width, 0],
            [0, 0, height],
            [length, 0, height],
            [length, width, height],
            [0, width, height]
        ]) + lower_corner

        faces = [
            [vertices[0], vertices[1], vertices[2], vertices[3]],
            [vertices[4], vertices[5], vertices[6], vertices[7]],
            [vertices[0], vertices[1], vertices[5], vertices[4]],
            [vertices[2], vertices[3], vertices[7], vertices[6]],
            [vertices[1], vertices[2], vertices[6], vertices[5]],
            [vertices[0], vertices[3], vertices[7], vertices[4]]
        ]

        obj_poly3d = Poly3DCollection(faces, alpha=0.7,
        ↪  facecolors='brown')
        ax.add_collection3d(obj_poly3d)

    # Plot the light positions
    for i, (pos, intensity) in enumerate(zip(light_positions,
    ↪  light_intensities)):
        ax.scatter(pos[0], pos[1], room_height, s=100, label=f'Light
        ↪  {i+1}: {intensity} lm')

    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('Z axis')
    plt.title('Room with Lights and Objects')
    plt.legend()
    plt.show()

# Example usage
room_length = 3
room_width = 3
room_height = 3
spacing = 0.5

# Manually specify light positions and intensities
light_positions = [(0.5, 2.0), (1.5, 2.0), (2.5, 2.0)]
light_intensities = [300, 500, 100]  # in lumens

# Create floor grid
```

```
floor_grid = create_grid(room_length, room_width, spacing)

# Manually specify objects
object1 = create_object(lower_corner=(1, 0, 0), length=1, width=0.5,
↪  height=2)  # Example object
object2 = create_object(lower_corner=(1, 0, 0), length=1, width=0.5,
↪  height=2)  # Example object
object3 = create_object(lower_corner=(0, 1, 0), length=1, width=0.5,
↪  height=1)  # Example object
objects = [object1]

# Calculate illuminance
illuminance = calculate_illuminance(light_positions, light_intensities,
↪  floor_grid, room_height, objects)
average_illuminance = np.mean(illuminance)
print(f'Average illuminance: {average_illuminance
```

A2. Raw Data

Experimental Setup

The experiments were conducted in a controlled indoor environment with dimensions of 3m x 3m x 3m. The room was equipped with various LED configurations and object placements to evaluate the optimization algorithm for visible light communication (VLC) systems.

Room Configurations

Setup	Room Length (m)	Room Width (m)	Room Height (m)	Objects	Object Position (m)
1	3	3	3	No	N/A
2	3	3	3	Yes	Object 1: Lower corner: (1, 2, 0), Length: 1, Width: 0.5, Height: 2
3	3	3	3	Yes	Object 1: Lower corner: (1, 1, 0), Length: 1, Width: 0.5, Height: 2
4	3	3	3	Yes	Object 1: Lower corner: (1, 0, 0), Length: 1, Width: 0.5, Height: 2
5	3	3	3	Yes	Object 1: Lower corner: (1, 0, 0), Length: 1, Width: 0.5, Height: 2; Object 2: Lower corner: (1, 2, 0), Length: 1, Width: 0.5, Height: 2
6	3	3	3	Yes	Object 1: Lower corner: (1, 2, 0), Length: 1, Width: 0.5, Height: 2; Object 2: Lower corner: (1, 0, 0), Length: 1, Width: 0.5, Height: 2; Object 3: Lower corner: (0, 1, 0), Length: 1, Width: 0.5, Height: 1

Table 7.1: Summary of room configurations and object placements for each experimental setup.

Lighting Configurations

LED Type	Lumen (Day)	Lumen (Night)	Power Consumption (Wh/min)
1	100	100	0.0167
2	300	300	0.05
3	500	500	0.083

Table 7.2: LED specifications used in the experiments.

Data Analysis

The data were analyzed to determine the effectiveness of different LED configurations in achieving uniform light distribution and meeting the minimum illuminance requirements for VLC systems. The average illuminance for each setup was calculated and used to assess the quality of lighting.

Results Summary

A summary of the experimental results, including the optimal LED positions and the standard deviation for each setup, can be found in the main body of the dissertation.

Data Availability

The complete dataset, including all measurements and calculations, is available upon request. Please contact the author for access to the raw data files.

A3. Detailed Results

Average Illuminance Values

The following table provides the average illuminance values recorded for each setup. These values represent the overall average illuminance achieved across all measured grid points. The setup configurations can be found in Table 7.1.

Setup	**Average Illuminance (lux)**
1	81.49
2	75.65
3	61.71
4	81.46
5	67.44
6	67.31

Table 7.3: Average illuminance values for each experimental setup.

A4. Ethical Considerations and Approval

This project, titled *"Enhancing Spaces for VLC Communication: Optimal Room Design"*, primarily focuses on the technical optimization of visible light communication (VLC) systems within controlled indoor environments. The scope of the research involves the development and testing of algorithmic solutions for LED light placement and intensity optimization, utilizing simulation and theoretical modeling.

Ethical Approval Determination

Upon thorough review, it has been determined that this project does not require formal ethical approval for the following reasons:

1. **No Human Participants**: The research does not involve the collection of data from human participants, either directly or indirectly. No surveys, interviews, or observational studies have been conducted as part of this project.
2. **No Personal Data Collection**: The project does not involve the collection, use, or processing of any personal data. All data utilized within the research are derived from theoretical models and simulations, ensuring that no identifiable or sensitive information is used.
3. **Technical and Theoretical Focus**: The project is confined to technical and theoretical exploration within a controlled environment, with no real-world implementation affecting individuals or public spaces.

While every effort has been made to adhere to ethical standards and protect all stakeholders, the author and associated parties disclaim any responsibility for any unintended consequences arising from the use or application of the research findings. The research is presented in good faith, with the understanding that the scope of work has been carefully evaluated to minimize any potential ethical concerns.