[Document title]

[Company name] | [Company address]

[Document subtitle]

Mikołaj Grobelny

[Year]

Contents

**No table of contents entries found.**

# Abstract

# Introduction

## What is an Optimal Camera Placement?

FIGURE showing an example of camera placement problem.

Optimal Camera Placement is a type of NP-Complete problem where the objective is to place cameras in a way that maximises the coverage of cameras in a system whilst minimising cost. An example of such system can be seen in figure 1.1-1.

The need to solve this issue comes from the fact that security companies are struggling to create optimal camera systems. Their current approach is based on trial-and-error and experience of their employees **Invalid source specified.**. This results in a number of cameras that is not ideal, leading to either blind spots or a high maintenance cost.

## Aims and Objectives

The **main aim** of this project is to develop a web application allowing for security employees to receive a viable camera placement based on provided data. Viable solution means achieving performance same as or higher than a human camera placer whilst taking less than a minute to execute (Given that area is not large).

The **secondary aim** is to conduct a survey of algorithms that can be used to solve the camera placement problem and compare them against each other. This allows other researchers to continue the work done in this paper, leading to better algorithms that will be improve user’s experience.

Following objectives are required to achieve these aims:

1. Background and Related Work
   1. Research currently available Optimal Camera Placement algorithms.
   2. Filter out algorithms that were not designed to operate in real-life situations.
   3. Determine difficulty of implementation for algorithms and rank them from simplest to hardest.
2. Requirements
   1. Research how security companies place cameras currently.
   2. Write user stories based on the research
   3. Translate user stories into Functional Requirements
3. Design and Specification
   1. Research technologies best suited to meet requirements
   2. Design an Application Architecture Diagram
   3. Make a Low-Fidelity Prototype of Application’s interface
4. Backend Implementation
   1. Implement the home page
   2. Implement connection between Backend and Frontend
   3. Implement backend environment and helper functions (Such as visibility checker)
5. Frontend Implementation
   1. Implement the User Interface designed in part 3c.
   2. Implement the Functionality required by part 2c.
   3. Allow Frontend to send information to the Backend
6. Algorithm Implementation
   1. Implement as many algorithms from list 1c as possible in the time frame
   2. Conduct an analysis of implementations, comparing vs number of cameras vs performance
   3. Based on previous results, determine which algorithm has the highest average performance.

# Background

## Related Work (Frontend)

The first step on creating an Optimal Camera Placement application was evaluation of systems with similar functionalities.

Whilst there are application allowing to user to manually place cameras and examine coverage, such as one present in <https://ipvm.com/reports/camera-design-tool-with-google-maps-integration>, there is currently no publicly available programs that allow user to automatically place cameras based on user requirements.

As such, this project be inspired by features manual placement applications and where necessary use suggestions from research papers focused on real-life scenarios.

### IPVM Camera Calculator

The project integrates Google Maps with IPVM technology allowing user to create cameras with custom parameters and overlay them over the map. By doing so, security employees can see the vision cone of each individual camera.

Additionally, the project simulates how a human face looks at different distances within the visibility cone, simplifying camera placement in situations where face recognition is a requirement.

Unfortunately, this calculator suffers from several issues:

* The map only shows outdoors, preventing usage inside of buildings
* The map is displayed at an angle, disallowing precise planning
* Cameras have fixed height, making the application difficult to use in a varied-height terrain (As the simulated view is distorted).
* The application does not show blind spots, preventing efficient systems
* It is impossible to create circular walls due to polygon-only walls

As such, I have decided to add user-based camera creation to the list of potential features as it allows them to adjust the system depending on camera models that are available to them.

Additionally, the camera cone visualisation would make it easier for security officers to determine if a camera has been placed correctly. However, in contrast to IPVM application, the visualisation will account for obstacles such as walls.

### JVSG CCTV Design Software

In contrast to the IPVM, the JVSG uses a high-fidelity 3D game engine to simulate camera placement. This allows to account for varying height, making the system more realistic.

The software also includes a simplistic simulation of movement for both people and vehicles. Due to that a security employee can determine the accuracy of the CCTV in a non-static environment.

Another advantage is that the JVSG shows blind spots in the system, preventing situations in which an area of the environment is left unprotected.

Whilst all these features are helpful at creating an efficient system, they come at a cost of complicated UI. The company offers several video guides to help users, but the system is still difficult to use after watching them.

The CCTV Design Software also runs locally on every device, requiring high-end components to run effectively. As such, each security employee in the field would need to be equipped with an expensive laptop, further increasing costs for this already costly software.

Additionally, due to 3D modelling and high fidelity, each environment takes several hours to create as each building needs to be modelled and have elements such as windows and doors placed in correct places as otherwise the program results in an inaccurate system.

Experience with this program has pushed me towards a more abstract and less resource intensive environment where user would only need to model where element can block vision or not.

The main useful idea is the detection of blind spots. As the allocation of cameras will be done automatically, it is crucial for end-user to be able to determine which parts of algorithm-generated layout are covered by cameras.

## Related Work (Algorithms)

Whilst the user-oriented features are an important part of this project, the main functionality distinguishing it from other camera placement application is an automatic camera placement. As such, I need to research potential algorithms that can be used within the project. They would also be ranked in order of implementation complexity to prevent project from not being delivered by the deadline. There are few requirements for the algorithm to be considered.

Firstly, the algorithm needs to be responsive. This means returning results in less than a minute for an average size system. Whilst this might reduce number of available optimal algorithms, a security employee can use this responsiveness to make incremental changes to the environment, recovering from mistakes. It also excludes brute force algorithms that are efficient but would take too long to run for the result to be relevant.

Additionally, it needs to handle multiple objectives. The main objective is overall coverage, determining effectiveness of the system. The secondary objective tends to be overall cost, coverage per camera or some other metric that determines how efficiently cameras are placed. It is possible to implement this problem as a single objective with secondary objective being used as a constraint. However, this would make the project susceptible to user-defined constraint, leading to poor effectiveness when constraint is just below the optimal point for the system.

### Greedy Algorithm

The Greedy Algorithm examines the search domain containing all possible elements that can be included as part of the solution. Each element has a score assigned to it that depends on its effectiveness vs efficiency. It can be assumed that the camera’s fitness is equal to coverage divided by cost. The algorithm iterates through every element in the domain, attempting to find the maximum score.

Unfortunately, such algorithm would return poor results as cameras are not independent of each other. As each area needs to be viewed by only one camera, each additional camera viewing similar area reduces efficiency without increasing effectiveness.

As such, each time a selection is made from the array, the algorithm needs to exclude the area of already chosen cameras from every camera in the search domain. This process will be computationally expensive, potentially negating time complexity saved on the algorithm itself.

Regardless of the actual effectiveness, the Greedy Algorithm will still form a baseline to compare other solutions against, as it is the simplest to implement whilst having a significantly lower time complexity than brute force algorithm.

### Random Sampling

In this case, rather than evaluating individual cameras, the Random Sampling algorithm checks every possible camera placement, determining randomly if camera should be there, and with what parameters.

Each iteration, a random solution will be generated, scored and if better than current best solution, designated as the best solution.

The algorithm is best suited for best suited for dealing with small search spaces as it can more efficiently explore them. The bigger the search space becomes, the higher the number of possible iterations that need to happen in order to get an efficient solution.