ALEXANDRIA UNIVERSITY

OPTIMIZATION PROBLEMS IN VISUAL SURVEILLANCE

by

Ahmed Abdelkader Abdelrazek

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science

in the
FACULTY OF ENGINEERING
ENGINEERING MATHEMATICS AND PHYSICS DEPARTMENT

August 2013

OPTIMIZATION PROBLEMS IN VISUAL SURVEILLANCE

Presented by

Ahmed Abdelkader Abdelrazek

For The Degree of

Master of Science

In Engineering Mathematics

Advising Committee	Approved
Prof. Dr. Mina Badie Abd-el-malek	. . . .
Dr. Amr Mohamed Abdelrazek	. . . .
Dr. Hazem Mohamed El-Alfy	. . . .

ALEXANDRIA UNIVERSITY

Abstract

FACULTY OF ENGINEERING
ENGINEERING MATHEMATICS AND PHYSICS DEPARTMENT

Master of Science

by Ahmed Abdelkader Abdelrazek

Visual Surveillance is concerned with the utilization of modern digital cameras to achieve common surveillance goals. Thanks to the wide availability of low cost and highly efficient sensors, there has been a proliferation of deployments of visual security systems as seen in banks, airports and train stations. High end visual sensors also play a crucial role in autonomous robots like unmanned aerial vehicles. In this thesis, we study two variants of the visual surveillance problem.

In the first problem, surveillance video streamed from a top-view camera is processed to control the orientation of multiple pan-tilt-zoom cameras in order to cover as many targets as possible at high resolutions. The problem of covering a set of static targets with a set of cameras is a planar variant of the classical combinatorial *set cover problem* and has been shown to be computationally intractable. We develop two new heuristics, compare them to existing solutions and show their superiority by extensive simulations. Moreover, to demonstrate the applicability of the proposed methods, we build and evaluate a real surveillance system for indoors pedestrian tracking.

In the second problem, we study a variant of the classical *pursuit-evasion game*, in which an agent moving amongst obstacles is to be maintained within *sight* by a pursuing robot. We design an efficient algorithm that decides if the evader can take any moving strategy to hide from the pursuer and win the game. For situations where the evader cannot win, we compute a pursuit strategy that keeps the evader within sight. Finally, if it is determined that the evader wins, we compute its optimal escape trajectory and the corresponding optimal pursuit trajectory. We analyze the algorithm, present several optimizations and show results for different environments.

Acknowledgements

All praise be to Allah, the lord of the worlds, the most gracious, the most merciful.

I would like to express my sincere gratitude to my advisors. I am very thankful to Dr. Hazem El-Alfy for introducing me to many exciting research problems and working by my side over the past couple of years. I cannot thank enough Prof. Dr. Mina Abd-el-malek for welcoming me into the department and actually introducing me to Dr. Hazem, which is how this thesis work started. I also thank Dr. Amr Abdelrazek for his honest advices and continuous encouragement.

I am most indebted to my friends, professors, colleagues and students who supported me in many ways. I have to thank Moamen Mokhtar for helping me complete crucial parts of this work. I also thank Mohamed Fadl, Ahmad Mamdouh, Ahmed Zakaria and Usama Elnily. Finally, I would like to express unparalleled gratitude to Mohammed Karmous for his good company all along this journey.

Last but not least; I would like to thank my family.

Contents

List of Figures

List of Tables

Abbreviations

CNF	**C**onjunctive-**N**ormal **F**orm
FOV	**F**ield **o**f **V**iew
FPS	**F**rames **P**er **S**econd
NP	**N**ondeterministic **P**olynomial
PDE	**P**artial-**D**ifferential-**E**quation
PTZ	**P**an-**T**ilt-**Z**oom
SAT	Boolean **Sat**isfiability Problem
SCP	**S**et **C**over **P**roblem
UAV	**U**nmanned **A**erial **V**ehicle

Chapter 1

Introduction

Visual Surveillance is concerned with the utilization of modern digital cameras to achieve common surveillance goals like monitoring, intrusion detection and target tracking. The obtained visual signals are readily available for higher level functions like face, object and action recognition which makes it particularly appealing for a wide range of application domains. Thanks to the availability of low cost and highly efficient sensors, there has been a proliferation of deployments of visual security systems as seen in banks, airports and train stations. High end visual sensors also play a crucial role in autonomous robots such as unmanned aerial vehicles (UAVs).

The design and tuning of surveillance systems is a very active area with a number of interesting research problems. Such systems are required to efficiently utilize the available resources to achieve certain surveillance quality metrics by sampling and extracting information from noisy input signals. Due to their critical nature, these system have to operate in the presence of hostile entities which make it even harder to maintain the required quality of monitoring. In this thesis, we study two variants of the visual surveillance problem.

1.1 Coverage Maximization in Multi-Camera Surveillance

Video surveillance systems have witnessed great developments and wide adoption in the past few years to answer the ever increasing demands in security and public safety [1]. Advances in cheap network cameras, digital and communications capabilities rapidly increased the number of cameras in use. This made the task of following monitors by human operators not only outdated but also practically infeasible. For that reason, several methods have been developed to automate the detection and reporting of scenes, events or subjects of interest. Traditionally, hundreds of hours of surveillance video from static cameras are stored to backup systems and examined later if needed. Covering a large environment obviously requires the use and coordination of several cameras. With the increased numbers of cameras in use, interest has recently shifted to control individual pan-tilt-zoom (PTZ) cameras to focus on specific events or subjects.

Monitoring several targets moving in an environment with multiple PTZ cameras has many conflicting objectives which make it a challenging task [2]. Among these objectives are:

- Maximizing the number of covered targets.
- Obtaining a high resolution view of each target.
- Obtaining an unobstructed view of targets in the presence of obstacles.
- Efficient recomputation of camera to target assignments as targets move about.
- Obtaining a smooth video per camera without abrupt jumps.
- Coping with camera hardware restrictions such as pan and zoom speeds.
- Selecting appropriate processing configurations, whether centralized or distributed.

Many attempts at the problem of monitoring multiple targets with multiple cameras have been suggested to date, with each one looking at the problem from a different angle. We can fairly claim that this problem is still far from being satisfactorily solved.

1.1.1 Problem Overview

We consider the problem of controlling multiple cameras to track multiple targets moving among obstacles [3]. We are mainly concerned with small scale networks. Hierarchical methods can be used to handle large camera networks. Our end goal is to develop a method that can be applied in real time to an actual surveillance system. We assume a top-view wide-angle camera (*master camera*) returns ground coordinates of moving targets. We are not concerned with the details of any particular tracking method in the present work as much as we are concerned with strategies to pan cameras in order to maximize the number of covered targets. The returned target locations are then used to automatically control the orientation and, eventually, the zoom of a set of high resolution PTZ cameras (*slave cameras*) in order to follow (*track*) those targets.

We have studied many approaches presented in the literature to the problem of maximizing the number of covered targets with a set of cameras (or sensors). Obtaining an optimal solution is computationally intractable and, for that reason, heuristics have been suggested to increase efficiency at the expense of optimality. In addition, the dynamic nature of the problem requires a solution that can be cleverly updated as targets move, which turns out to be much harder than just maximizing the coverage at any given time instance with fixed target locations.

1.1.2 Summary of Contributions

We compare several heuristics found in the literature, suggest two new ones and verify their superior performance by extensive simulations. In addition to the common objective of maximizing the number of targets covered by cameras, our new heuristics present, to the best of the authors' knowledge, a first attempt at continuous panning as opposed to selecting from a discrete set of orientations like similar existing methods.

1.2 Visibility-based Pursuit-Evasion

We consider the problem of target tracking, that is planning the motion of a mobile robot (pursuer) as it tracks a target (evader) moving amongst obstacles. We use the term *target tracking* to mean *following* that target or, more precisely, maintaining its visibility. This terminology is common within the robotic planning community [4] as opposed to the broader notion of tracking, known in the computer vision literature, which refers to the identification of paths of different targets. Several applications of that problem are suggested in the literature [4–6]. Those cover surveillance and security in sensitive or restricted areas, providing home care by watching over children or elderly people and monitoring the performance of human workers.

The problem we study belongs to the more general *visibility-based pursuit-evasion problems* [7]. In those problems, the task of a pursuer is to compute a path which guarantees finding an evader that might be hiding in an environment. Obviously, the evader might *sneak* between different hiding places making a single pursuer unable to solve the problem. The natural extension becomes that of finding the minimum number of pursuers needed to eliminate any hiding places for one or more evaders under different constraints of knowledge of the environment and information about other players' locations [8–11].

Once the evader is found, the related problem that arises is that of maintaining its visibility which is known as *target tracking*. An early attempt at that problem is presented in [12] for fully predictable targets (optimal tracking paths found offline) and partially predictable targets (best next step found online). Recently, there has been interest in a variant of that problem in which a pursuer, tracking an unpredictable evader, loses immediately if its view of the evader is obstructed by an obstacle [5, 6]. The problem of deciding which player wins for any pair of initial positions has been shown to be NP-complete [13].

1.2.1 Problem Overview

In this thesis, we are interested in the latter problem of deciding which player wins the pursuit-evasion game described above. This is a two-player game, with one pursuer and one evader modeled as points that can move in any planar direction, i.e., holonomic robots. Each player knows exactly both the position and velocity of its opponent. We consider two-dimensional environments containing obstacles that obstruct the view of the players. Both players have bounded speeds and can maneuver to avoid obstacles. Players are equipped with sensors that can *see* in all directions.

The pursuit-evasion game proceeds as follows. Initially, the pursuer and the evader are at positions from which they can see each other. It is common in the literature to define two players to be visible to one another if the line segment that joins them does not intersect any obstacle. The goal of the game is for the pursuer to maintain visibility of the evader at all times. The game ends immediately, if at any time, the pursuer loses sight of the evader. In that case, we say that the pursuer loses and the evader wins.

1.2.2 Summary of Contributions

The main contribution of this part of the thesis is the development of an algorithm that enhances recent results in AI planning and visibility-based pursuit-evasion to tackle the computationally prohibitive task of deciding the outcome of the pursuit-evasion game. This is a significant improvement over earlier depth-limited or heuristic-based approaches. The computed results are also used to find optimal player trajectories and optimize various objectives. The basic model [14] is formally studied to facilitate the derivation of several suggested optimizations. This, to our knowledge, provides first evidence of the feasibility of optimal decisions at such high grid-resolutions under varying speed ratios, different visibility constraints and regardless of obstacle geometries. Furthermore, we present several extensions and applications to the presented solution, beyond deciding the result of the game, namely trajectory planning. Finally, we present our own variant of the problem, where the pursuer is allowed to lose sight of the evader for a limited duration before regaining visibility, and show how the developed techniques extend to this interesting case as well.

Chapter 2

Related Work

In this chapter, we survey the literature for relevant work in our area. We highlight both classical results and recent progress. By demonstrating the wide variety of approaches and different kinds of results achieved in the problems we study, the richness of this area becomes clear. We pay more attention to closely related results and briefly mention other interesting works.

2.1 Coverage by Directional Sensors

In the area of combinatorial optimization, several attempts have been made at the problem of maximizing the number of covered targets using the minimum number of directional sensors. Relevant literature is found in [15–17]. The optimal solution is computationally expensive and hence, several sub-optimal heuristics are suggested. An overview of the more generic assignment problem is presented in [18].

When moving to the practical area of camera management, many challenges appear. The first step is to set-up and calibrate the cameras [19]. Then, comes the task of controlling the cameras. It is worth noting that one of the first attempts that uses one wide-view camera and one high-resolution camera is a decade old [20]. Depending on the type of application, different restrictions apply. In classroom videos [21], the location of the targets are known which simplifies the problem of camera control. In multimedia and entertainment areas, selecting the best pose of targets might be the challenge [22].

Different combinatorial approaches have also been suggested. A self updating bipartite matching approach with clustering of neighboring targets is presented in [3]. Probabilistic approaches can be found in [23, 24]. A centralized approach is suggested in [25], a 3D real system is presented in [26] and a large scale system can be found in [27]. Good surveys with comparisons of several methods are found in [24] and, for game theoretic approaches, in [28].

2.1.1 Hardness and Lower Bounds

Several attempts [16, 17, 29] have been made to show the NP-hardness of maximizing the number of covered points by a set of directional sensors and other closely similar problems e.g., energy efficient target coverage. Earlier attempts presented reductions from the Boolean Satisfiability Problem (SAT). Namely, the satisfiability of boolean formulae in Conjunctive Normal Form (CNF) with up to 3 literals per clause (3SAT). However, these proofs failed to introduce rigorous geometric constructions that apply to realistic camera models. A rigorous reduction that uses Planar 3SAT [30] finally appeared in [31]. A lower bound of 0.5 was shown to hold for approximation ratio of the greedy heuristics presented in [31] and [32]. This means that the coverage computed by these methods is at least half the optimal coverage for any input configuration.

2.1.2 Linear Programming Relaxation

One of the earliest attempts to attack the inherent hardness in combinatorial optimization was the linear programming model presented by Lovász for fractional set covers [33]. The fractional solution can be rounded back to an integer solution to give a bounded approximation ratio of the original combinatorial problem. Even more interestingly, it has been shown that a naive greedy algorithm achieves the same approximation ratio [34].

2.1.3 Centralized Methods

The first efficient heuristic was introduced in [17] and was called the Centralized Greedy Algorithm (CGA). The algorithm repeatedly selects the camera that covers the largest number of uncovered targets until all cameras are assigned. Later in [15], the algorithm was slightly modified to give higher priority to cameras having fewer options in hope of leaving out fewer targets when the algorithm terminates. As the modified weight function divides the original weight by the number of visible targets, it was called the Centralized Force-Directed Algorithm (CFA). Both methods were compared against optimal solutions of the integer programming formulation and shown to yield very good coverage in reasonable time.

2.1.4 Scalable Methods

When moving to larger camera networks, the simple greedy methods presented in 2.1.3 are no longer suitable and would take considerable time to terminate. Instead, it is assumed that each sensor is allowed to communicate with its neighbors to reach an agreement on how each camera should be assigned. While global coverage would suffer, this results in a more computationally scalable scheme. One such scheme is the Distributed Greedy Algorithm in [17], which assigns a priority for each node in order to guarantee unique tie breaking for any target that could possibly be covered by more than one camera. Another method is the Modified Single-Linkage Algorithm in [15] which starts by clustering the network into separate clusters then computes camera assignment within each cluster.

2.2 Pursuit-Evasion Games

A large amount of literature has been devoted to pursuit-evasion games. In this section, we review closely related work, with a focus on attempts at deciding the outcome of the game. In the field of robotics, we survey recent work in the problem where one pursuer maintains the visibility of one evader, in an environment with obstacles. In artificial intelligence, we review the related problem of cops and robbers.

In the area of robotics motion planning, the main approach used is to decompose the environment into noncritical regions with critical curve boundaries, across which critical changes in occlusion and collision occur, then use a combinatorial motion planning method. Murrieta-Cid et al. [6] model the pursuit-evasion game as a motion planning problem of a rod of variable length, creating a partitioning of the environment that depends on the geometry of obstacles. Later in [13], they present a convex partitioning that is modeled as a *mutual visibility graph*. Their method alternates between an evader assumed to take the shortest step to escape, countered by a pursuer that computes a prevention-from-escape step, which produces a sequence of locally optimal paths. This leads to an interesting result: to decide which player wins, every feasible ordering of local paths has to be checked, concluding it is an NP-complete problem.

In the robotics motion planning area, two approaches are used. Using the terminology in [7], these are *combinatorial approaches* that find exact solutions through a continuous space and *sampling-based approaches* that divide the space (probabilistically or deterministically) into discrete regions and find paths through these regions. The simplest form of deterministically dividing the space (a.k.a cell decomposition) is with a grid of fixed resolution. The main advantage of this approach is its simple implementation in contrast to combinatorial methods, many of which are impractical to implement. However, cell decomposition methods are *resolution complete* (unlike combinatorial methods), which means that they find a solution when one exists only if the resolution of the sampling grid is fine enough. Another common drawback with grid methods is that their complexity depends on the grid size [35].

Within the AI planning community, the related problem of cops and robbers consists of one or more players (cops) trying to find and catch one or more evaders (robbers). The players perform alternating moves. This makes the game naturally discrete, often modeled as a graph with vertices representing the game's states. The mathematical foundations for solving these problems are surveyed in [36]. A polynomial time optimal algorithm that determines whether the cops or the robbers win, and in how many steps, was given in [37]. Unfortunately, methods for computing optimal strategies have always been impractical to implement.

Grid discretization is often used in Artificial Intelligence to solve problems in *AI planning* [38]. In AI, the focus is on solving the problem of finding and/or catching the evader using algorithms that search the discretized state space along with heuristics to speed up the process. The problem of deciding if a solution exists is only approached theoretically and is often considered intractable. In contrast, our approach presents a grid-based solution that can be applied in practice to a robotics path planning problem. It enriches the literature by linking to existing research in that area, modeling realistic constraints of bounded speeds, different player speeds and limited sensor range.

2.2.1 Classical Results

Two problems of special appeal are studied in [39] and [40] which were first posed by Isaacs in his famous book on Differential Games [41]. In [39], Fitzgerald presents a solution to the

Princess and Monster Problem, while a solution to the Angel Problem is presented in [40].

In [42], Gal considers search games in which the searcher moves along a continuous trajectory until he captures the hider in either a network or a two dimensional region. Both mobile and immobile hiders are considered. The strategy of an immobile hider is a probability distribution of the hiding point, while the strategy of a mobile hider is a continuous path inside the region.

Such classical results continue to find applications to real-world critical problem like: biology [43], finding terrorists [44] and scheduling patrols for fare inspection in transit systems [45], to name only a few recent applications.

2.2.2 Decidability and Differential Games

Bhattacharya et al. address the problem of maintaining the visibility of an escaping evader and show that it is completely *decidable* around one corner with infinite edges [46]. The authors then extend their work in [5] to deal with more general environments with convex obstacles. They split the environment into decidable and non-decidable regions and approximate bounds on these regions. They also provide a sufficient condition for escape of the evader. Recently, they used differential games theory to analyze that problem under complete information, suggesting a formulation in which the pursuer maximizes the time for which it can track the evader while the evader minimizes it [47, 48]. Computing equilibrium strategies gives necessary and sufficient conditions for tracking. They present results around a point obstacle, a corner and a hexagonal obstacle.

2.2.3 Static Games

In contrast to feedback strategy games, which require solutions to Hamilton-Jacobi-Issacs partial differential equations (PDEs) in the joint configuration space, like the one described in 2.2.2, the authors in [49] study a static version of the game which can be solved directly in the state space by a proposed PDE-based technique. In the static setting, both players choose their controls at the beginning and run in open-loop with the same objectives of maximizing vs minimizing the total visibility time. The advantage of this restricted version is significant savings in computational cost, at the expense of more conservative information pattern compared to the original close-loop version. Furthermore, the authors describe how to generalize their algorithm to games with multiple evaders and other applications to target tracking.

2.2.4 Cops and Robbers

Recently, a polynomial time algorithm that decides which player wins and in how many steps has been developed [37]. Mathematical results from graph theory are used to present bounds on the time complexity of the problem that can be generalized, in theory, to the case of more players. A realization of such algorithms in real world problems is still not practical. In contrast, the grid-based solution we present can be applied in practice to a robotics path planning problem.

Moldenhauer and Sturtevant [50] compute optimal move policies offline in 2.5 hours per environment using an enhanced form of the algorithm in [37]. For that reason, several heuristics have been used in order to compute near optimal approximations in practical time. In [51], different optimal strategies are studied on small maps while in [50], larger maps are used to evaluate less optimal strategies against optimal ones. One of the first practical implementations of the cops and robbers game is presented in [38] under the name of *moving-target search*. Since then, that problem has been extensively studied using a variety of heuristics such as incremental heuristic search [52] and *Cover* heuristic [53], to name a few recent references.

Chapter 3

Coverage Maximization in Multi-Camera Networks

We study the problem of orchestrating a network of multiple cameras to track multiple mobile targets in an environment populated with obstacles of arbitrary shapes [3]. We are primarily interested in small scale networks consisting of a relatively small number of cameras. Several hierarchical methods have been presented to handle larger networks. Our primary objective is to devise an efficient algorithm that can be applied in real time to a real surveillance system. We assume a top-view wide-angle camera (*master camera*) covers the whole area with all moving targets. Frames captured by this master camera are used to estimate target locations. The system uses these estimated locations to automatically control the orientation of a set of high resolution PTZ cameras (*slave cameras*) to provide high quality coverage of these targets. Our problem reduces to *assigning an orientation for each camera* in the network, in order to cover the maximum possible number of targets.

3.1 Problem Definition

Traditionally, each camera is assumed to choose from a limited set of discrete orientations e.g., 8 different angles. The combinatorial nature of the discrete *camera orientation assignment problem* makes it difficult to define a continuous objective function as far as target coverage is concerned. This situation leaves us with either the computationally prohibitive, but exact, integer programming formulation or trying different suboptimal heuristics that allow for computationally efficient and evidently practical solutions as in [17]. The problem, however, could be attacked by linear programming relaxation but such a formulation is, to the best of our knowledge, not yet presented in the literature. Taking visibility constraints into consideration, to account for the possible presence of walls and obstacles, as is the case in this study, would make it even harder to model.

The assignment problem at hand has been believed to be NP-hard for quite some time. The authors in [16, 17, 29] present several arguments to this point which all start by first mapping the problem to a set structure similar to the Maximum Coverage Problem. This essentially rids the problem of its geometric nature and no formal reduction was presented. A rigorous hardness proof, which presents a formal reduction from a variant of the Planar 3SAT problem, later appeared in [31]. This implies that, unless $P = NP$, seeking an optimal solution incurs a computational cost that is exponential in the size of the problem e.g., for m cameras each having 8 possible orientations this amounts to an $\mathcal{O}(8^m)$ cost per assignment.

The problem does bear great similarity to the well known Set Cover Problem (SCP), with the additional difficulty of multiple orientations per camera. A linear programming relaxation of SCP was first considered by Lovász [33] which could yield a logarithmic factor approximate solution through randomized rounding [54]. It was later showed that the greedy strategy of repeatedly selecting the set covering the largest number of uncovered elements, or the set with the maximum weight in the weighted version of the problem, performs equally well [55]. This should justify why the greedy algorithms suggested in [15, 17] are very effective.

In this chapter, we develop two novel heuristic methods for the camera orientation assignment problem. Both methods are based on the greedy heuristic for the set cover problem as in [15, 17]. The first method defines a relaxed notion of coverage and uses it as a weight function to choose from the available sets at each step. The second method reuses the sweeping concept from computational geometry to expose maximal sets at each step.

We use the following notation below: We let C be the set of cameras in the network and T be the set of targets to be covered. We also set $M = |C|$ and $N = |T|$.

3.2 A Continuous Coverage Function

Taking hint from Lovász's relaxation, we seek a relaxed coverage function where covered and non-covered targets are not strictly mapped to 1 and 0, respectively, as one would expect. Such a function might be easier to handle while still preserving the coverage maximization objective. The following technical lemma provides useful hints towards the development of such flexible objective functions to our problem.

Lemma 3.1. *Fix a positive real $w_{c,t}$ for each camera c and target t, such that $\frac{w_{min}}{w_{max}} > \frac{N-1}{N}$. Any weight function of the form*

$$f_c(t) = \begin{cases} w_{c,t}, & c \text{ covers } t, \\ 0, & \text{otherwise}, \end{cases} \tag{3.1}$$

describes at least one optimal solution maximizing $\sum_{c \in C} \sum_{t \in T} f_c(t)$.

Proof. Let n and n' be the sizes of two sets of targets the system could possibly cover, with $n, n' \in \{0, 1, 2, ..., N\}$. For $n > n'$, any coverage maximization objective function should always assign a larger total weight to the larger set. Now, assume all targets in the larger set were assigned the smallest possible weight, i.e., w_{min} while all targets in the smaller set were assigned the largest possible weight, i.e., w_{max}. We must have $n\, w_{min} > n'\, w_{max}$ or $\frac{w_{min}}{w_{max}} > \frac{n'}{n}$. This is trivially satisfied by the usual binary coverage function with $w_{min} = w_{max} = 1$. However, we desire to find the lower bound for this ratio which is clearly determined by the largest value of the right hand side of the inequality $\frac{n'}{n}$. Maximizing the numerator simplifies this to $\frac{n-1}{n}$, which is strictly increasing in n. Setting n to the maximum possible value, N, yields the largest ratio and forces $\frac{w_{min}}{w_{max}} > \frac{N-1}{N}$. When $n = n'$, the two sets do not necessarily get assigned the same weight. Hence, the set maximizing the objective function above always achieves maximum coverage. □

Such a characterization of weight functions reveals the undesirable discontinuity over covered and non-covered targets. Notice that non-covered targets still had to receive zero weight while any covered target receives a positive weight. This corresponds to the discrete transition from the covered state to the non-covered state as the camera pans so that the target becomes outside its FOV. We suggest a workaround exploiting the geometric structure of the problem through what we call *Angular Relaxation* (AR).

It is usually preferred to have subjects at the center of the FOV rather than near the sides. This leads to a natural weight function we derive from the angular offset θ computed in the reference frame defined by the camera and its current direction d. Motivated by Lemma 3.1, we start with the following expression

$$f_c(t) = \left(\frac{N-1}{N}\right)^{\theta_c(t)/\theta_{max}} \tag{3.2}$$

where $\theta_c(t)$ is the offset of target t with respect to camera c and θ_{max} is the maximum offset, i.e., half the FOV. For covered targets, $\theta_c(t) \in [0, \theta_{max})$ and we get $f_c(t) \in (\frac{N-1}{N}, 1]$. This means subjects at the center are assigned weights of 1 which decays exponentially to the sides of the field of view, and attains a value of $\frac{N-1}{N}$ right outside the FOV. Subjects further outside the FOV get smaller weights approaching 0. However, to achieve near zero weights for targets right outside the FOV we relax the bounding ratio for targets inside the FOV and use $\frac{1}{e}$ as the base instead of $\frac{N-1}{N}$.

As the camera pans, $\theta_c(t)$ becomes a function of d. In particular, a given target receives an additional offset defined by the triangular wave $2\pi|\frac{d}{2\pi} - \lfloor\frac{d}{2\pi} - 0.5\rfloor|$. We approximate $\theta_c(d,t)$ by its Fourier series using only the first two terms, $\frac{\pi}{2}\Big(1 - \cos\left(\theta_c(t) - d\right)\Big)$, as shown in Figure 3.1 for a target located at $0°$.

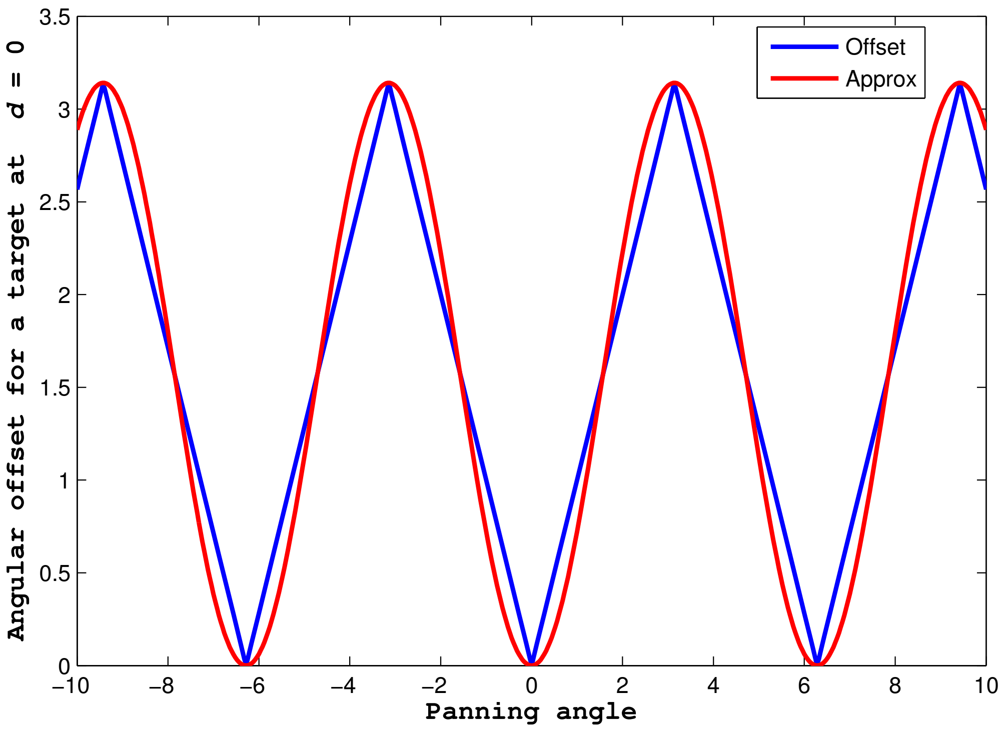


FIGURE 3.1: Angular offset θ_c as a function of the panning angle d.

With that, we reach the following form:

$$f_c(d,t) = e^{-\frac{\pi}{2}\left(\frac{1-\cos\theta_c(d,t)}{\theta_{max}}\right)} \tag{3.3}$$

Furthermore, we weight the whole set of targets covered by having any one camera assigned to a given direction by:

$$F(c,d) = \frac{1}{|T_c|} \sum_{t \in T_c} f_c(d,t) \tag{3.4}$$

where T_c is the set of all targets that could possibly be covered by c as in [15]. Figure 3.2 shows an example with four targets at angular offsets $[1, 1.3, 1.8, 4]$. Notice how each individual target contributes a peak of height 1 at the offset where it is located. The distribution of targets defines the shape of the objective function which peaks at the *hottest* region with a value approximating the anticipated coverage, before getting normalized by $\frac{1}{|T_c|}$.

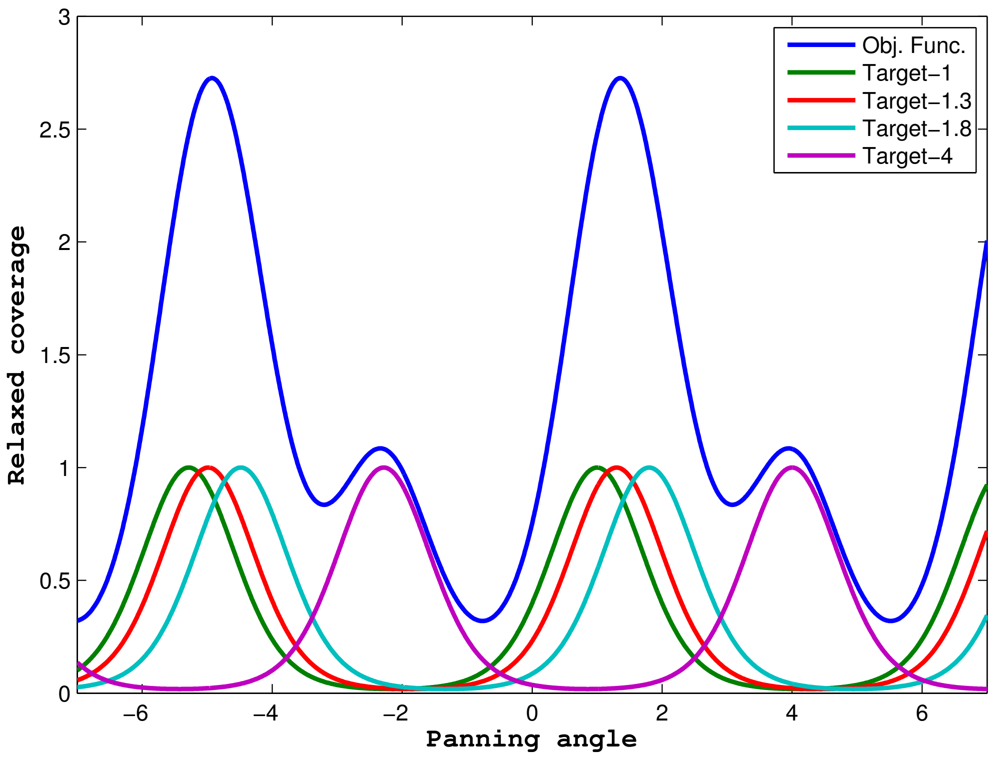


FIGURE 3.2: Composition of the objective function for four targets.

Our heuristic is a direct application of the greedy algorithm for weighted set cover [56] using Equation 3.4. While the algorithm was not developed for our case it still serves as a good start and we consider more tailored formulations as future work e.g., [57]. The algorithm repeatedly selects the pair (c^*, d^*) maximizing $F(c,d)$, till all cameras are assigned. Gradient ascent initialized at the target yielding the highest value of the objective function can be used to find the direction d^* for each camera. As the number of iterations required to compute

d^* is limited, i.e., a small constant, maximizing the relaxed coverage achievable by a single camera takes $\mathcal{O}(N^2)$. Hence, each iteration costs $\mathcal{O}(MN^2)$ to compute the best orientation for each camera in $\mathcal{O}(N^2)$, and we repeat for all remaining cameras. With M cameras to assign, the algorithm terminates after M iterations at a total cost of $\mathcal{O}(M^2N^2)$.

3.3 Coverage by Angular Sweeping

Taking hint from the algorithms suggested in [15] and [17], where panning was restricted to just 8 orientations, we make the following observation.

Observation 3.2. Any set of targets covered by a given camera can always be covered by panning the camera till one or more of the targets lie exactly on either side of the FOV.

To greedily maximize its coverage, a camera can iterate over the targets it could possibly cover and compute the coverage achieved by panning to have the target at hand at either the right or left sides of the FOV as in Figure 3.3. We also note that, using only one side of the FOV to perform the sweep is enough.

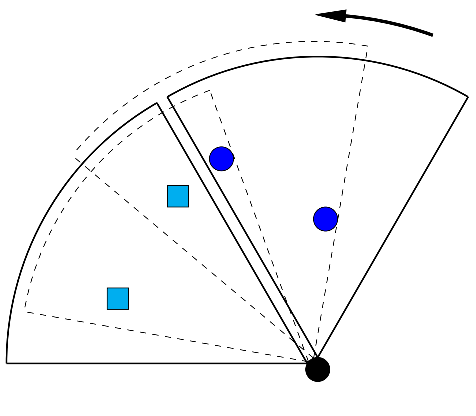


FIGURE 3.3: Sweeping always finds maximal groupings (dashed) unlike discrete orientations (solid).

As the FOV rotates, all possible groupings of targets will be encountered. After the panning angle yielding the highest coverage is found, it may be arbitrarily shifted to center the FOV onto the targets to be covered. Greedily choosing the camera achieving the highest coverage and proceeding till all cameras are assigned amounts to an $\mathcal{O}(M^2N^2)$ algorithm with $\mathcal{O}(N)$ panning angles and an $\mathcal{O}(N)$ cost to compute the coverage. This method easily presents an

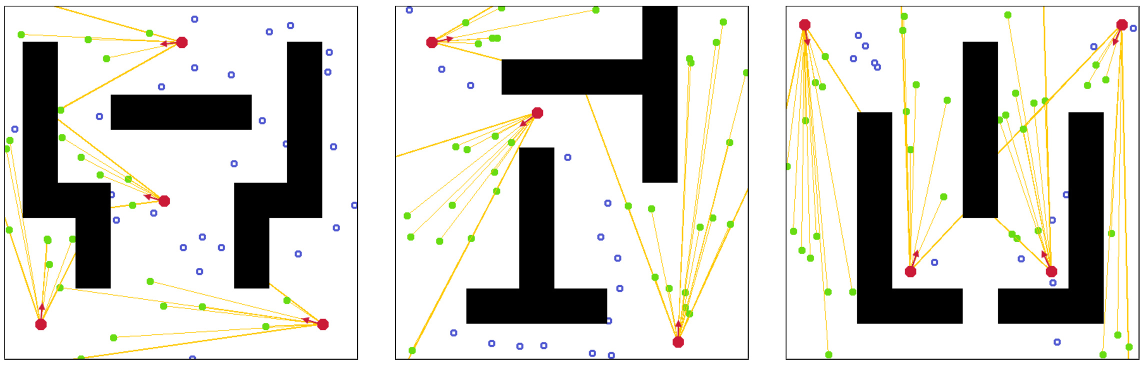

(A) (B) (C)

FIGURE 3.4: Sample test maps with assignment and coverage, showing walls (black), cameras (red), covered (green) and non-covered (blue) targets.

upper bound for similar greedy heuristics with discrete orientations, regardless of how many they are e.g., 8 [15] or higher, while at the same time being more efficient.

3.4 Experimental Evaluation

We study the performance of our new heuristics, Angular Relaxation (AR) and Angular Sweep (AS), compared to the Centralized Greedy Algorithm (CGA) [17] and Centralized Force Algorithm (CFA) [15], all against optimal solutions obtained by brute-force search over the camera orientation space (OPT).

As all earlier methods we compare against use a discrete number of panning angles, that number is indicated in the label of each graph. We present results for different scenarios with and without walls to simulate different indoors and outdoors environments. We use the coverage metric as the number of covered subjects divided by the total number of subjects in any given frame or iteration. We do not bound the FOV by maximum/minimum ranges as we are interested in small scale setups and study centralized algorithms. We use manually designed maps with different camera setups and random initial target locations as in Figures 3.4A to 3.4C.

Figure 3.5 shows the superior performance of AS. AR performed much better in the presence of walls as in Figure 3.6. Figure 3.7 and 3.8 show runtimes for different numbers of static targets, performed on an Intel i7 Quad Core with 4 GB of RAM. Our methods are more efficient than the CFA-32 and higher, with the AR performing faster in the presence of walls.

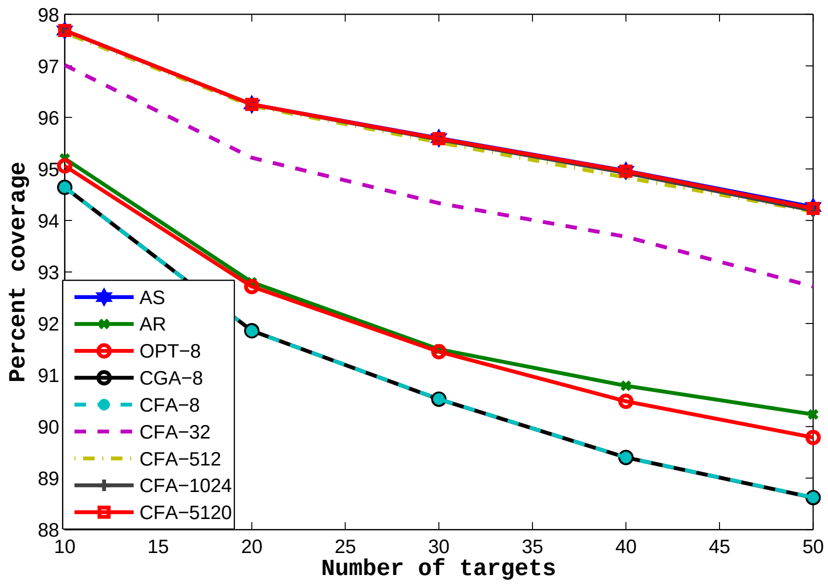


FIGURE 3.5: Varying the number of static targets without walls.

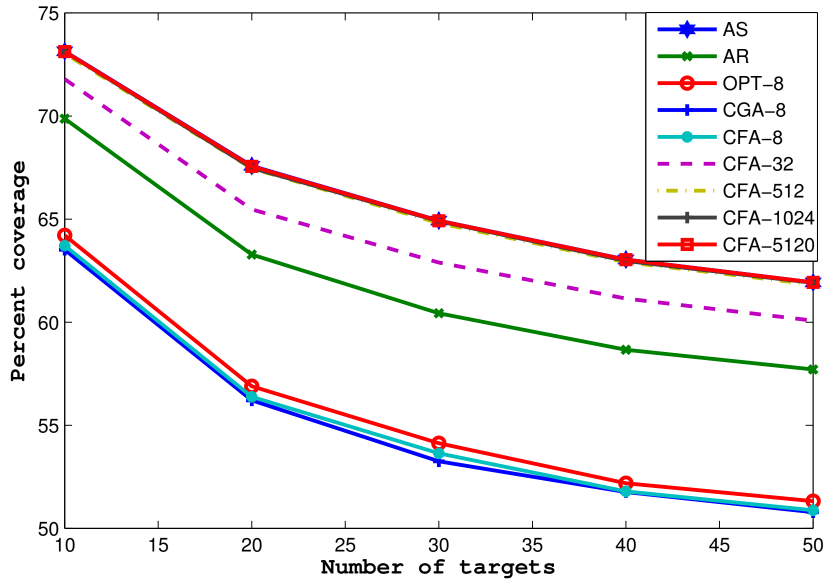


FIGURE 3.6: Varying the number of static targets with walls.

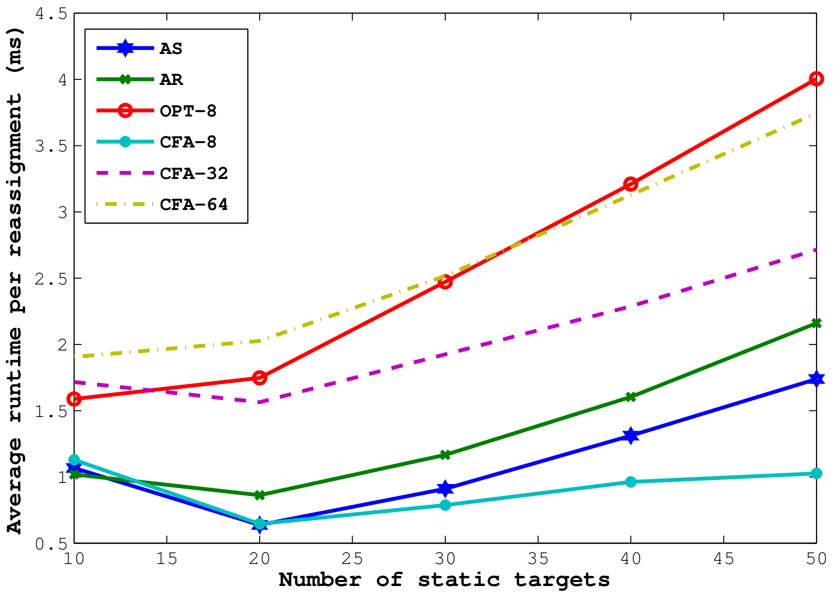


FIGURE 3.7: Runtime varying the number of static targets without walls.

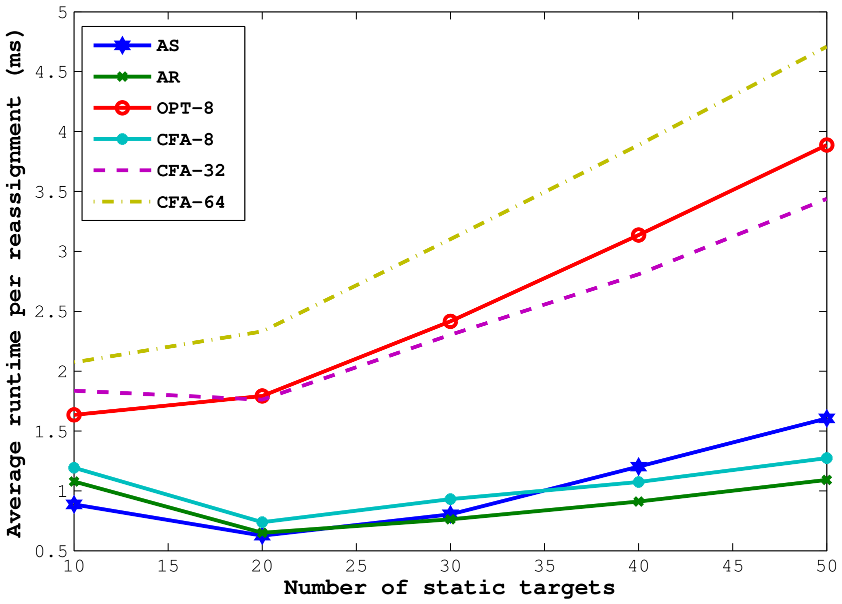


FIGURE 3.8: Runtime varying the number of static targets with walls.

The table below shows the coverage results of Figure 3.6. By examining the coverage rate achieved by each method, one easily realizes the clear advantage of our AS.

N	10	20	30	40	50
CGA-8	63.61	56.26	53.40	51.78	50.85
CFA-8	63.71	56.37	53.63	51.79	50.87
OPT-8	64.21	56.89	54.12	52.18	51.31
AR	69.87	63.27	60.42	58.66	57.70
CFA-32	71.78	65.47	62.89	61.14	60.07
CFA-512	73.02	67.45	64.80	62.90	61.82
CFA-1024	73.11	67.47	64.89	62.97	61.90
CFA-5120	73.12	67.55	64.91	63.03	61.92
AS	73.14	67.57	64.92	63.03	61.93

TABLE 3.1: Coverage rate for varying numbers of static targets with walls.

Finally, we simulate dynamic scnerios by allowing targets to move. For each map, we did 1000 iterations with camera assignments being recomputed every 10 frames. We did not pan cameras until the next assignment and assume that panning is performed instantaneously. We assume all targets move at the same speed for every step they take. Targets repeatedly select a random destination point and move towards it in straight lines. Figures 3.9 and 3.10 show the average coverage rate for each method over all test maps. Again, AS proves to be the best available solution.

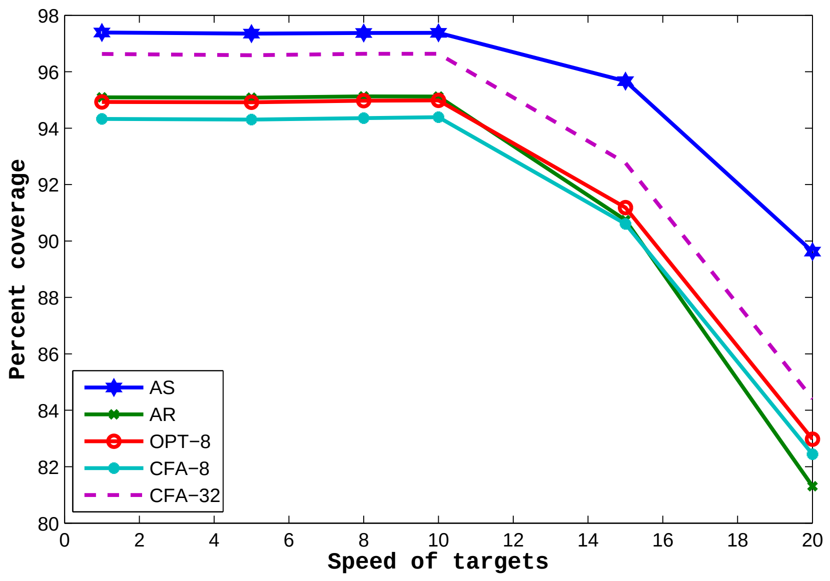


FIGURE 3.9: Varying speed of targets without walls.

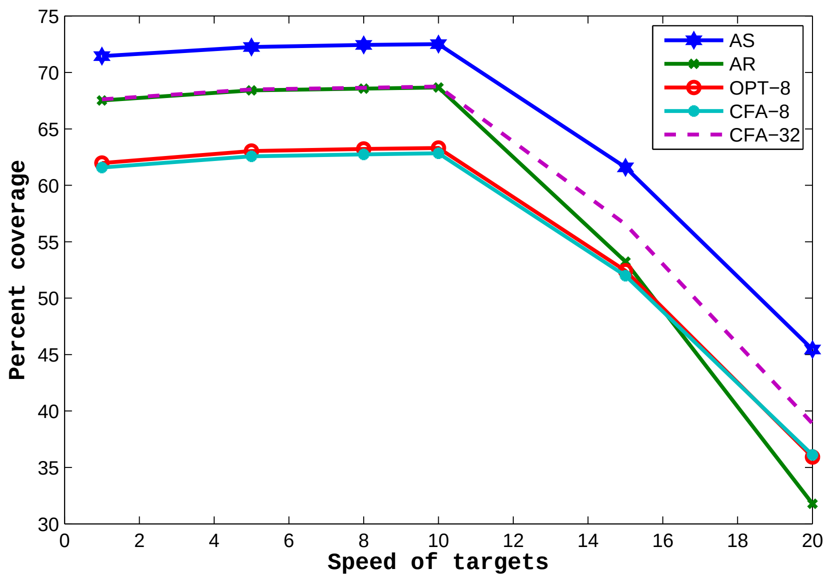


FIGURE 3.10: Varying speed of targets with walls.

Chapter 4

Camera Experiments

In this chapter, we give an overview of the associated experimental study performed during this thesis work to test our proposed methods in different environments. The experimental work proceeded in phases corresponding to the Alex-Eng-REP-2[1] grant funding and purchase cycles.

First of all, I would like to acknowledge our past collaborators who made significant contributions towards completing this experimental work. Namely, I must begin by acknowledging Moamen Mokhtar who joined our team as an intern in Summer 2011, and became responsible for implementing crucial parts of our software tools. Moamen have always been ready to help and continued to provide useful comments even after leaving the team. In Summer 2012, we were joined by Ibrahim Bassiouni, who took over the responsibilities of Moamen and helped maintain and enhance our tools to perform the final stages of the experiments. Other members who joined this effort are: Ahmed Emara, Mohamed Fahmy, Ahmed Gad and Mohamed Aref. Finally, I would like to thank our colleagues, students and friends who volunteered as actors in the human tracking experiments.

4.1 Simulated Environments

We implemented and used two simulators. The first is a full-fledged 3D simulator with animation and human models based on the jMonkeyEngine game engine [58]. We developed the second simulator, which employs a simpler 2D model, but enables thousands of runs in a few seconds. Both simulators perform the basic requirement of targets moving among opaque obstacles. Screen shots are shown in Figures 4.1 to 4.3 below.

[1]Alex-Eng-REP-2 is the Alexandria University, Faculty of Engineering, Research Enhancement Program, Cycle 2, March 2010. The grant is offered by the Scientific Publication and Research Development Fund (SPRDF) for enhancing the research conducted by faculty members.

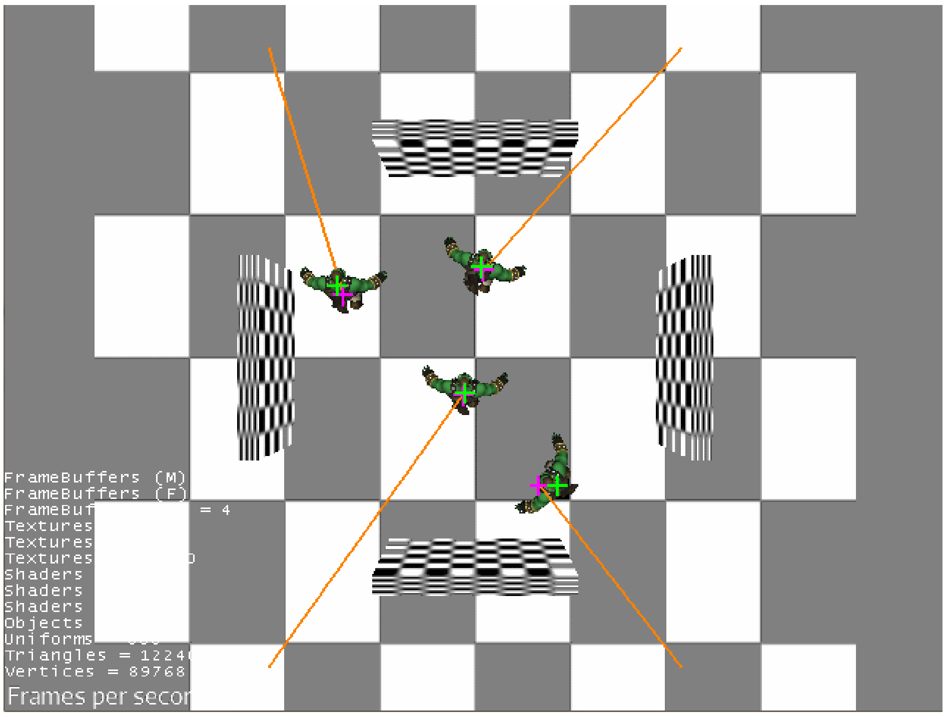


FIGURE 4.1: Master view in the 3D simulator showing camera to target assignments. Green and pink crosses indicate true and estimated target locations, respectively.

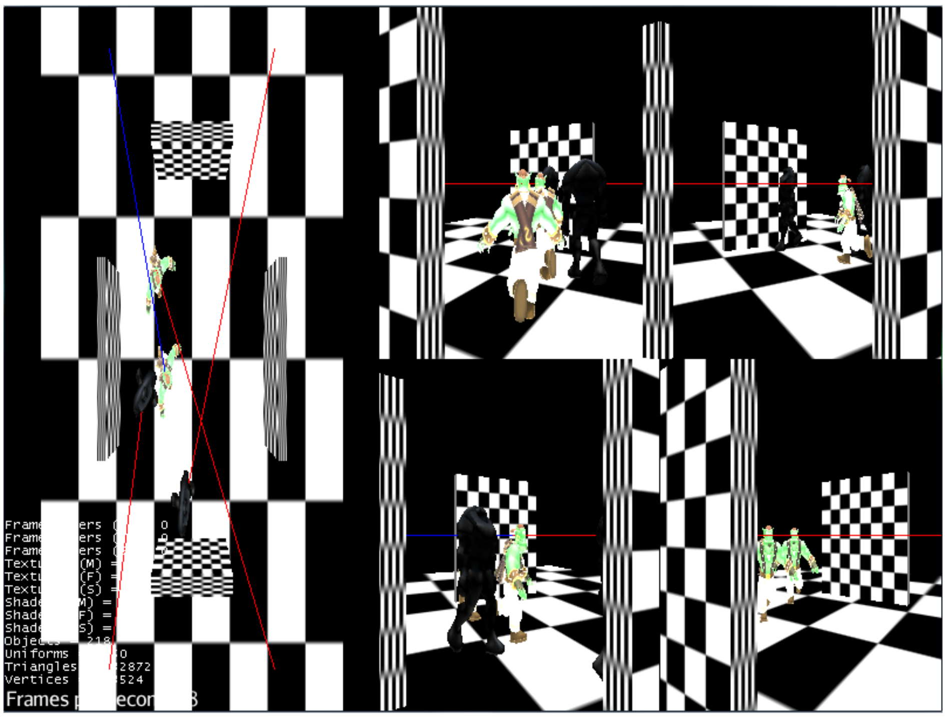


FIGURE 4.2: Full view of the 3D simulator, with frames from both master and slave cameras. The lines connecting cameras to targets indicate if the target is covered (red) or not (blue).

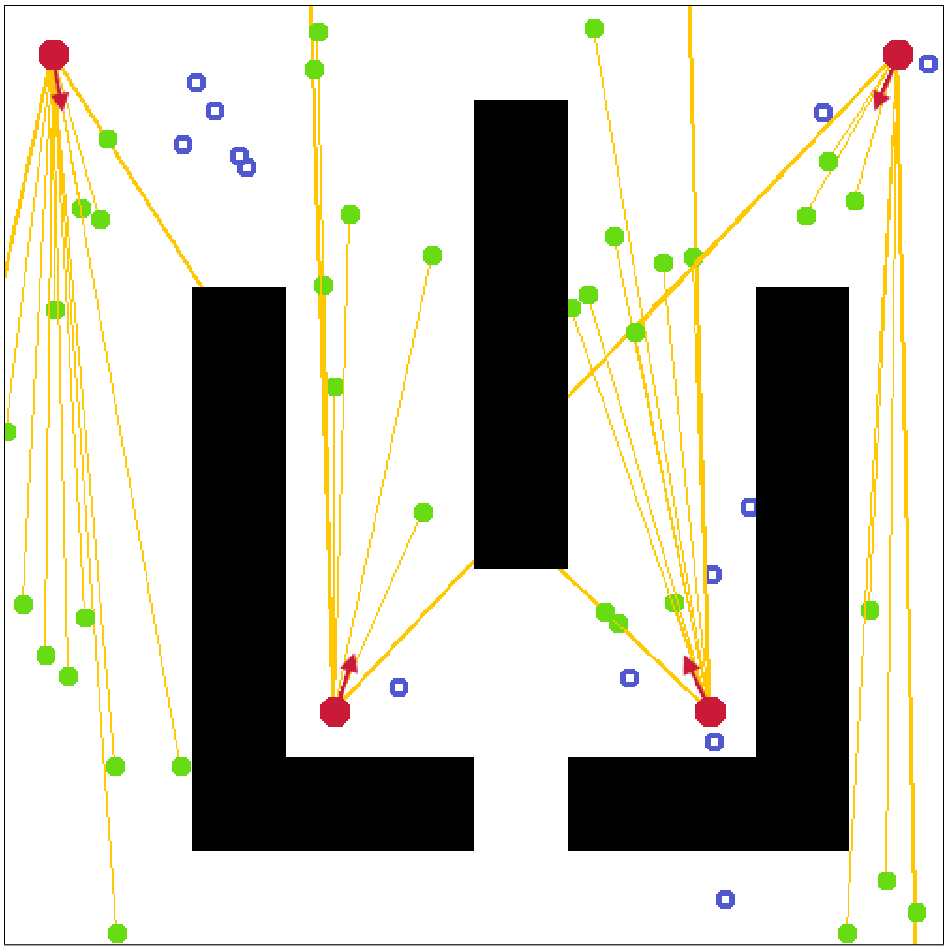


FIGURE 4.3: 2D simulator with a sample map, showing walls (black), cameras (red), covered (green) and non-covered (blue) targets.

4.2 Computer Vision Methods

Moving from simulators to real world experiments requires the use of a computer vision system to analyze video, detect moving objects and track them. A recent good survey of these methods is detailed in [59]. Though our research has not focused on enhancing video analyzing techniques, video tracking is applied to the stream of frames being sampled from the master camera. We assume a constant sampling frequency. Each frame goes through a processing chain of five stages in order to detect moving objects and estimate their locations, as summarized in Figure 4.4. A brief description of each stage is presented below.

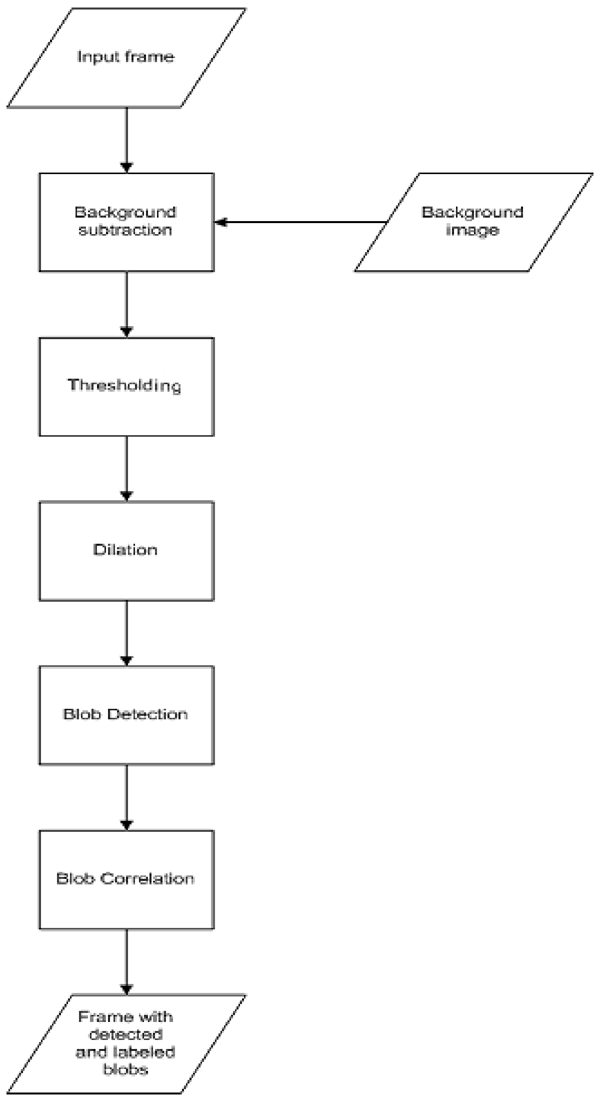


FIGURE 4.4: Processing stages of sampled frames.

4.2.1 Background Subtraction

The first step in object detection is to compute the difference between the input frame and the background image. An input frame is represented as a two dimensional matrix of pixels. Differences in pixel values indicate pixels that are likely to be part of a moving object. Figure 4.5B shows an input frame after applying background subtraction, with potentially moving pixels shown in white. With such pixels being distinguished from the background, they constitute the foreground image of moving objects which we are interested in processing.

4.2.2 Thresholding

Fuzziness, bad illumination, and other factors may introduce noise to the input video stream, which results in erroneously detecting background pixels as part of a moving object. Consequently, a threshold is applied to discard pixels with relatively small difference values which are most likely due to noise. A pixel exceeding the threshold is considered part of the foreground and is colored in white. Otherwise, the pixel is completely ignored and colored in black. Figure 4.5C shows the thresholded image.

4.2.3 Dilation

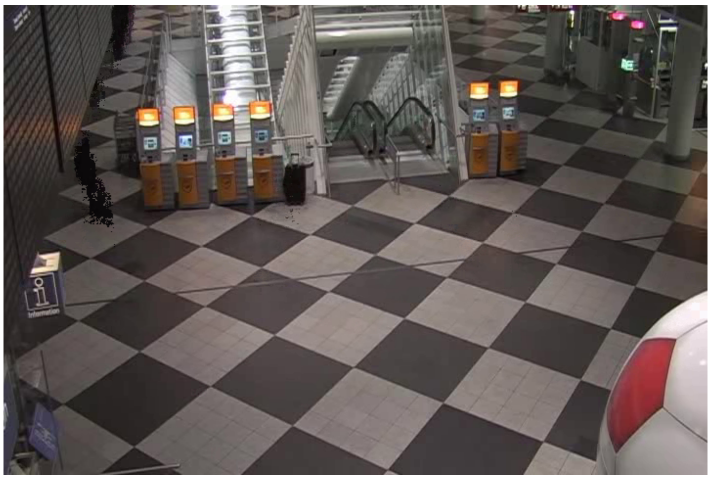
The input to this stage are blobs of white pixels representing moving objects. However, a single moving object might appear as multiple disconnected objects within the area where the real object is located in the frame. Dilation [60] is basically a morphological operator that diffuses a pixel value to its neighboring pixels, so close by disconnected object will be linked to form a single blob as required. Figure 4.5D shows the results of blob dilation to the thresholded frame.

4.2.4 Blob Detection

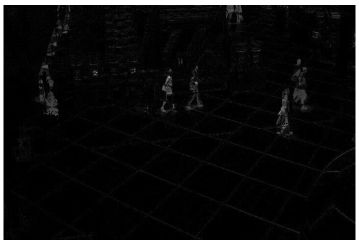
Now, foreground objects can be located in the frame. Blob detection is the process of labeling each pixel with a number identifying which object it belongs to. Pixels labeled with the same number are considered a single entity with a unique ID and a red border is drawn around it. Figure 4.5E shows the final result with the labeled blobs.

4.2.5 Blob Correlation

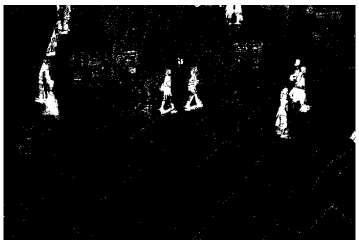
After assigning a label to each foreground blob, it is necessary to correlate the blobs in the current frame to the blobs in previous frames. A basic correlation algorithm is employed which simply calculates the difference in position of each pair of blobs in two successive frames. Blob pairs with the smallest offset are most likely the same blob after moving in the new frame and are given the same label as the matched blob.



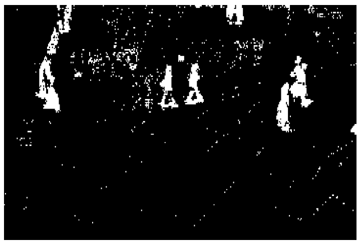
(A) Generated background.



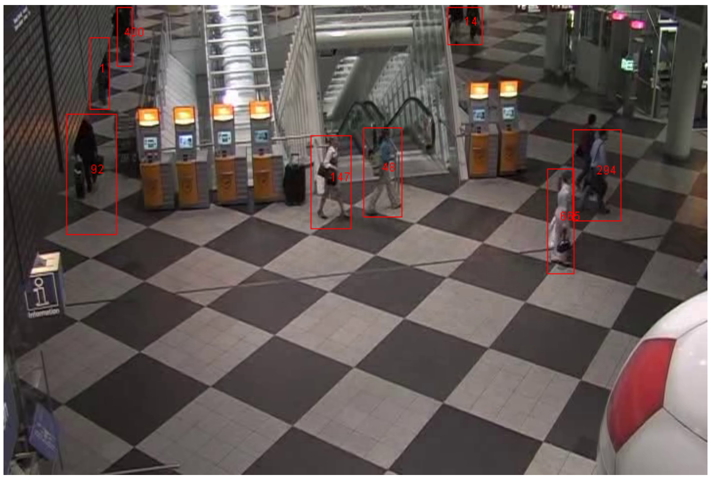
(B) Background subtraction.



(C) Thresholding



(D) Dilation



(E) Final result with labeled blobs.

FIGURE 4.5: Video processing steps needed to track moving targets.

4.2.6 Background Generation

As described above, the tracking algorithm assumes a static background for use in background subtraction. A static background may not be available, i.e., when the system first goes online, so it is required to generate such a background. A sequence of frames at the beginning are taken. The estimated background is constructed using the most frequent values at each pixel.

4.2.7 Closed Loop Testing

In order to test the video tracking implementation of the method described above, before going through the hassle of setting up a real camera network, we used our 3D simulator to experiment with four moving subjects and obstacles. Figure 4.1 depicts the simulated environment with the objects being tracked. Green crosses indicate the true location of the object while pink crosses mark the estimated object locations.

4.3 Hardware Calibration

The next step in moving to real world experimentation is to understand the limitations of our hardware to be able to design our camera management schemes accordingly.

4.3.1 Hardware Specifications

We used two models of cameras in the test. PZ7132 was used as the master camera, because it offers a wide field of view, while PZ7122 was used as the slave cameras. Table 4.1 shows the specifications of each model. The camera network we used consists of the following:

- One Linksys wireless router model WAG120N.
- One VivoTeK PTZ camera model PZ7132 WLAN.
- Four VivoTeK PTZ camera model PZ7122 WLAN
- One personal laptop works as application server.

Specifications	Camera Model	
	PZ7132 (Master Camera)	PZ7122 (Slave Cameras)
Pan Range	350°	300°
	(-175° , +175°)	° (-150° , +150°)
One Pan Move	19.4°	16.6°
Pan Speed	350° in 41 seconds	300° in 38 seconds
	1.3 seconds per 10°	
Tilt Range	125°	135°
	(-35°, +90°)	(-45° , +90°)
One Tilt Move	11.6°	15°
Tilt Speed	125° in 17.1 seconds	135° in 17.6 seconds
	1.3 seconds per 10°	
Zoom	2.6x Optical zoom	10x Optical zoom
Zoom speed	1.1 seconds per one zoom move	
Complete zoom in	10.1 seconds	17.4 seconds

TABLE 4.1: Cameras Specifications

The plan was to make use of wireless connection abilities in the camera models. However, that did not work as the access point could not handle the load of all five wireless cameras, especially the remote slave cameras. Therefore, we resorted to using Ethernet cables instead. The access point has four Ethernet ports, so all slave cameras could be wired to the access point. Still, the master camera and application server connected to the access point wirelessly.

4.3.2 Latency Measurements

This section presents results of the experiments we conducted to measure camera latencies which is defined as the difference in time between the scene in front of a camera changing, and that change appearing on an appropriate monitor [61].

4.3.2.1 Equipment

The following equipment were used to perform the test.

- One Linksys wireless router (WAG120N).
- One VivoTeK PTZ camera (PZ7132).

- One VivoTeK PTZ camera (PZ7122).
- One personal laptop as the application server.

4.3.2.2 Methodology

Testing was performed by running the mini camera network for 10 minutes then calculated the average latency in milliseconds for 3 different sampling frequencies, measured in Frames Per Second (FPS). The process was repeated twice to study the performance with and without real-time video tracking. Captured frames from the master camera are 320×240 pixels while frames from the slave camera are 352×288 pixels. Note that samples from slave cameras are 1.32 times the size of master camera samples.

4.3.2.3 Master Camera Only

Tables 4.2 and 4.3 show the results of testing a single master camera at two different distances from the access point. The results are summarized in Figure 4.6. We notice a considerable increase in latency after increasing the distance from the access point. The added latency due to real-time video tracking was negligible for this single camera scenario.

Camera	Distance	FPS		
		20	10	5
Master Camera	1 meter	152	232	273
	8 meters	263	277	321

TABLE 4.2: Latency results for one camera without real-time tracking.

Camera	Distance	FPS		
		20	10	5
Master Camera	1 meter	201	234	314
	8 meters	322	291	361

TABLE 4.3: Latency results for one camera with real-time tracking.

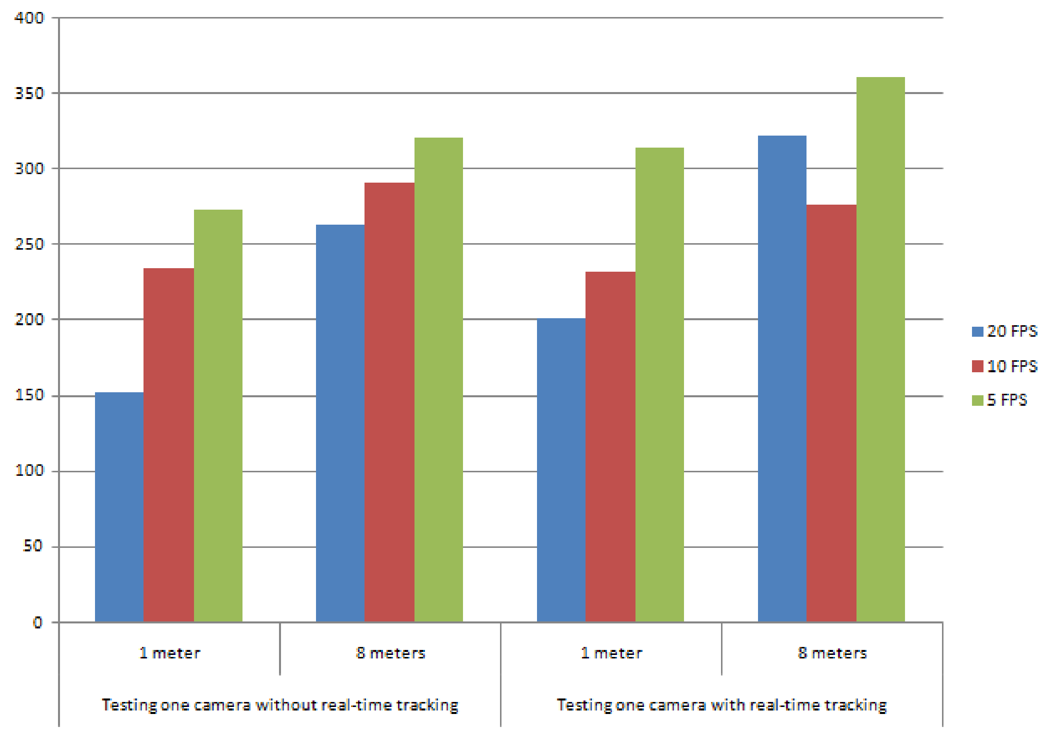


FIGURE 4.6: Latency tests of one camera as in Tables 4.2 and 4.3.

4.3.2.4 Master/Slave Network

Next, we measure latency for two cameras connected simultaneously, with each camera working on a dedicated thread. Again, we study two cases with different distances between the cameras and the access point, as summarized in Tables 4.4 and 4.5. Figure 4.7 shows a bar chart with latency values. Note that both cameras were unable to serve 20 frames per second at a distance of 8 meters from the access point while running the tracking code. However, lower frame rates seemed to work fine. This limitation placed a hard constraint on the design of our physical setup.

Camera	Distance	FPS		
		20	10	5
Master Camera	1 meter	197	214	313
	8 meters	660	281	276
Slave Camera	1 meter	303	281	323
	8 meters	828	447	401

TABLE 4.4: Latency results for two cameras without real-time tracking.

Camera	Distance	FPS		
		20	10	5
Master Camera	1 meter	197	331	393
	8 meters	connection lost	281	276
Slave Camera	1 meter	303	281	323
	8 meters	connection lost	447	401

TABLE 4.5: Latency results for two cameras with real-time tracking.

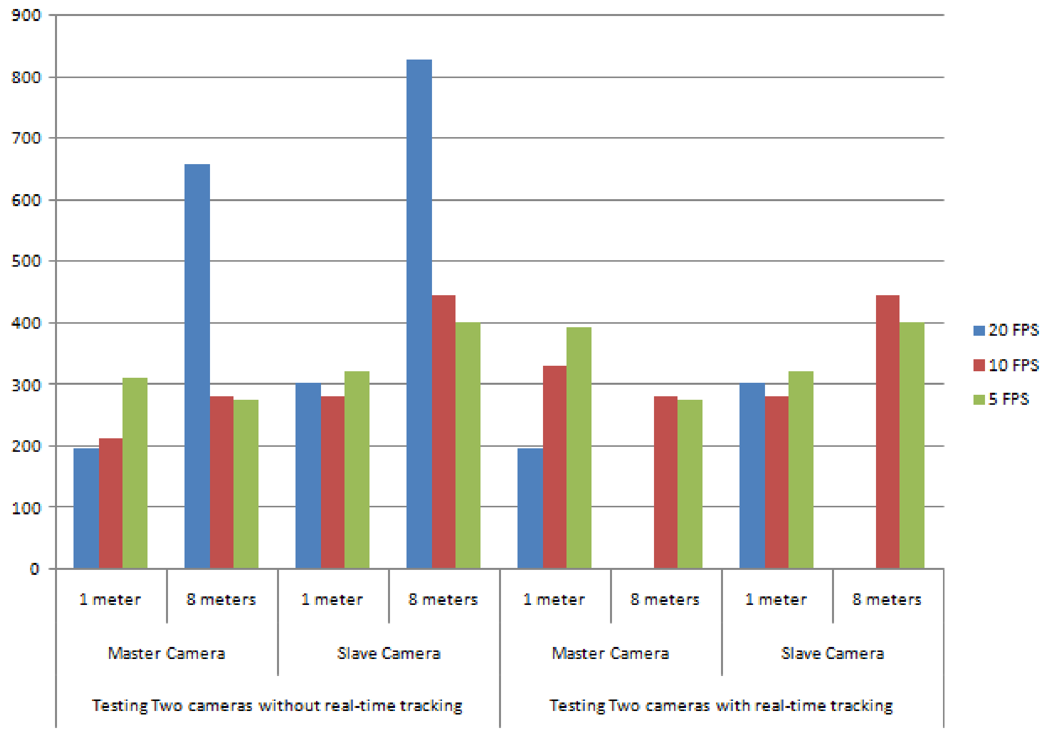


FIGURE 4.7: Latency tests of two cameras working simultaneously as in Tables 4.4 and 4.5.

4.3.3 Conclusion

The presented results showed that both the distance and number of running cameras have a great impact on the latency of captured frames. When two wireless cameras are working at a distance from the access point, latency increased considerably. The access point could not handle the load from the two cameras and connections got lost. Higher frame rate gave better results when the two cameras were operating near the access point, while lower frame rate gave better results when operating at a distance. However, with one camera only, higher frame rate always gave better results due to the light load on the access point.

We summarize here the key specs that affected our decisions. The Vivotek PZ7131 model has a 2.6x optical zoom only but an angle of view $73°$ in the horizontal direction and $55°$ in the vertical direction. This makes the PZ7131 suitable for capturing wide top-view frames. We used the PZ7112 model, which allows up to 10x zoom, as slave cameras. Both cameras have a pan step of $10°$. It takes 1.3 seconds to perform a pan command and 1.1 seconds to perform a zoom command. Additional constraints exist on the range of the wireless signal, estimated at $8m$, and the number of wireless cameras that can work simultaneously on the same access point. These limitations are taken into consideration when controlling the cameras. More information can be found in the data sheets [62].

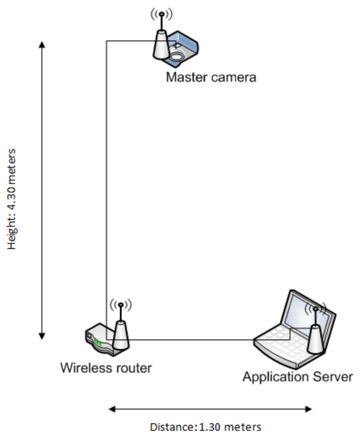
4.4 Toy Example

We started our first real world experiment with electric wires still unavailable. The experiment was installed in the 7^{th} floor of the Electrical Engineering Building. The experiment setup diagrams, photos from the setup and typical frames are shown in Figures 4.8 to 4.10. The center frame in Figure 4.10 is for the top-view from the master camera with the four slaves (not visible) installed at the four corners of the scene. Targets being tracked are surrounded by red boxes while yellow lines indicate the direction of each camera. The views from the slaves are shown as the small frames at the corresponding corners of the main view.

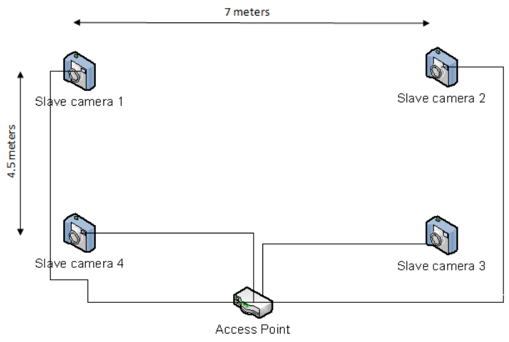
We used toy vehicles initially to test the system. Tracking humans with this setup was not successful due to the low height at which the master camera is installed, at $4.3m$, and the small area it was able to cover at that height. We performed a basic video tracking test nonetheless as shown in Figures 4.11 and 4.12.

4.5 Prototype System

As soon as we received the first shipment of electric wiring and outlets, we moved our setup to the main hall in the Preparatory Building. This hall has the advantage of having higher ceilings, allowing the master camera to be installed at $6.5m$, which provides a wider top-view of $8 \times 5.5\, m^2$.

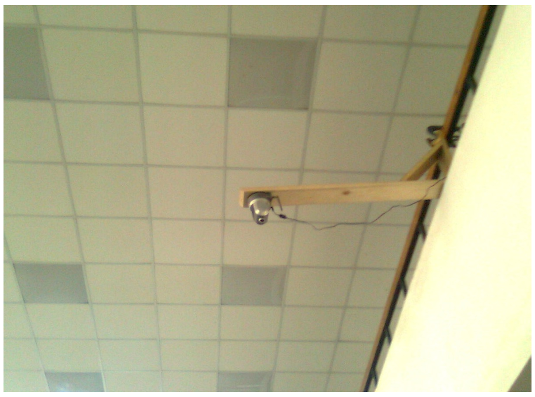


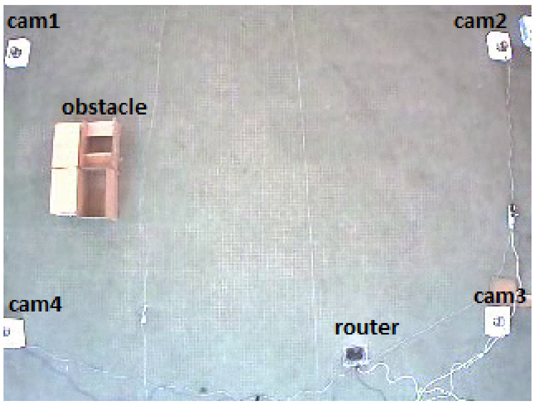
(A) Connections of the master camera and laptop to the access point.



(B) Connection of slave cameras to the access point.

FIGURE 4.8: Connection diagrams.


(A) Master camera at height $4.3m$.



(B) Top-view frame showing the setup.

FIGURE 4.9: Physical setup

The main setup of the experiment is shown in Figures 4.13 to 4.16. Figure 4.17 shows a screenshot of the system state for one experiment with the deployment described above.

The main problems that we faced in this setup are the lack of sufficient network cable (wireless connections congest easily) and slow processing power. In addition, strong sunlight coming from the building's main gate contributed noise to the tracking results.

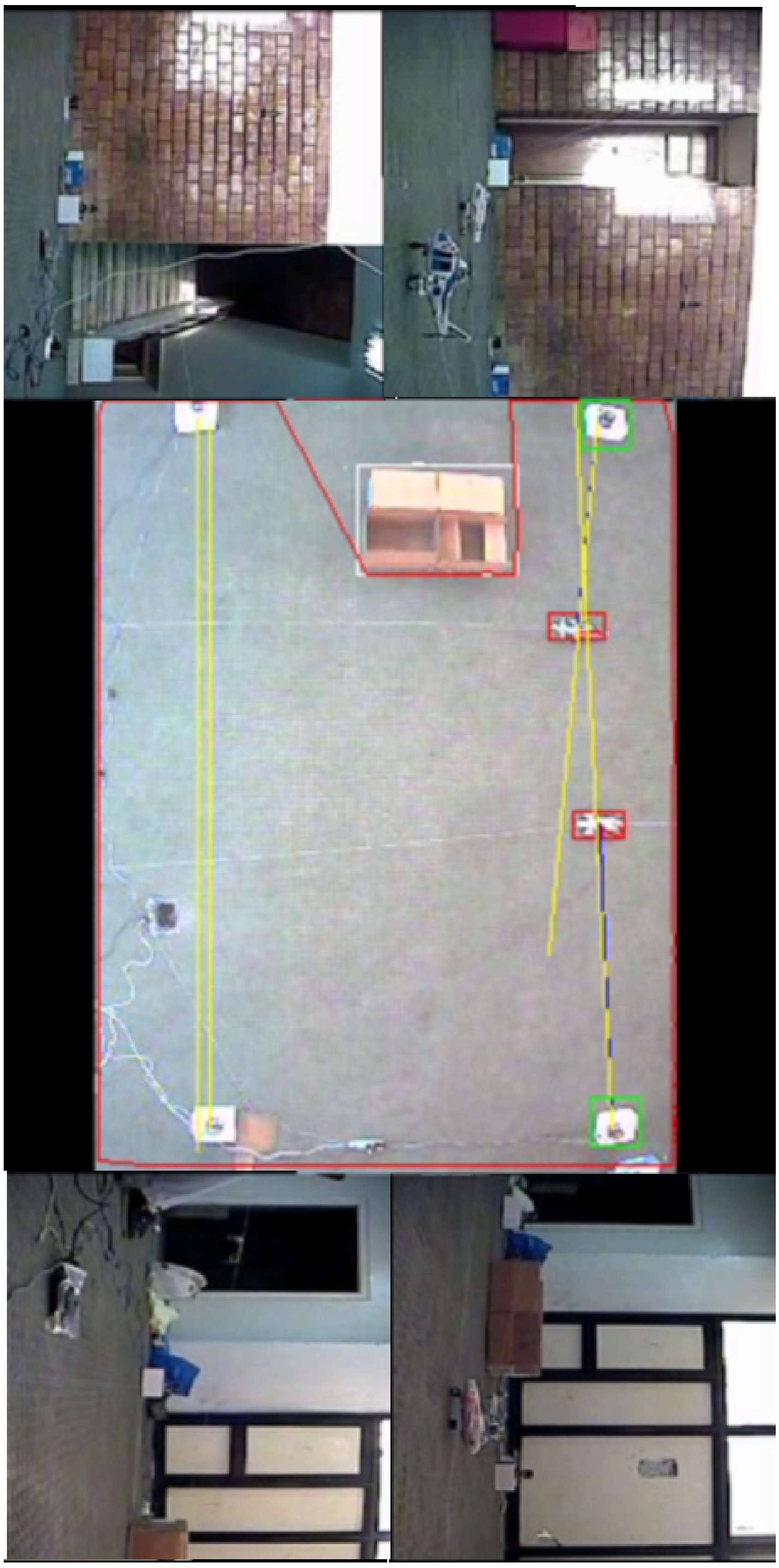


FIGURE 4.10: Example frame from an indoors toy tracking experiment.

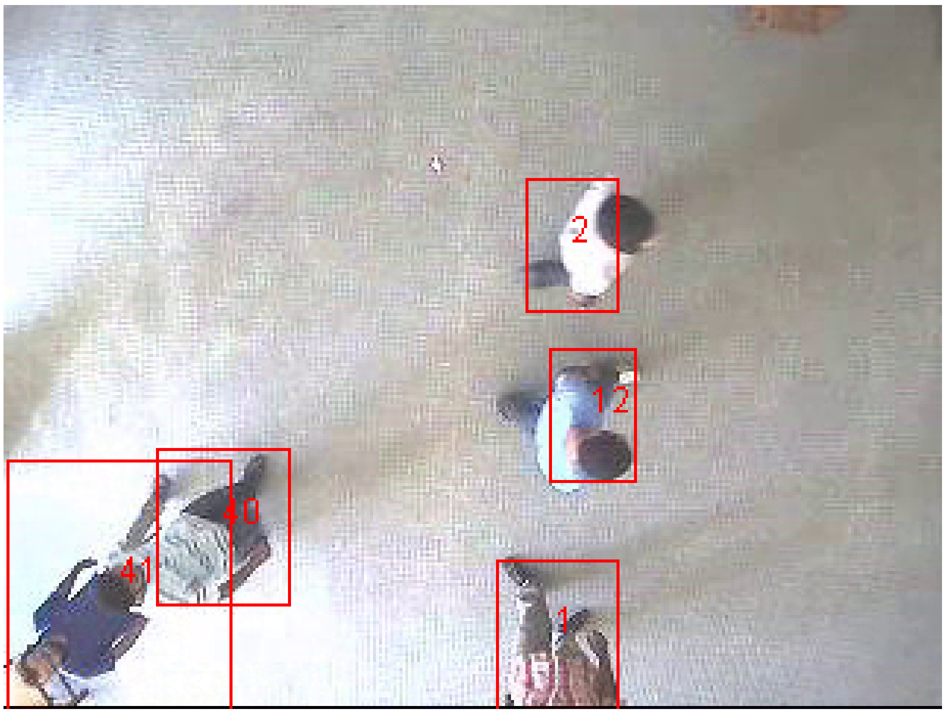


FIGURE 4.11: First attempt at pedestrian tracking.

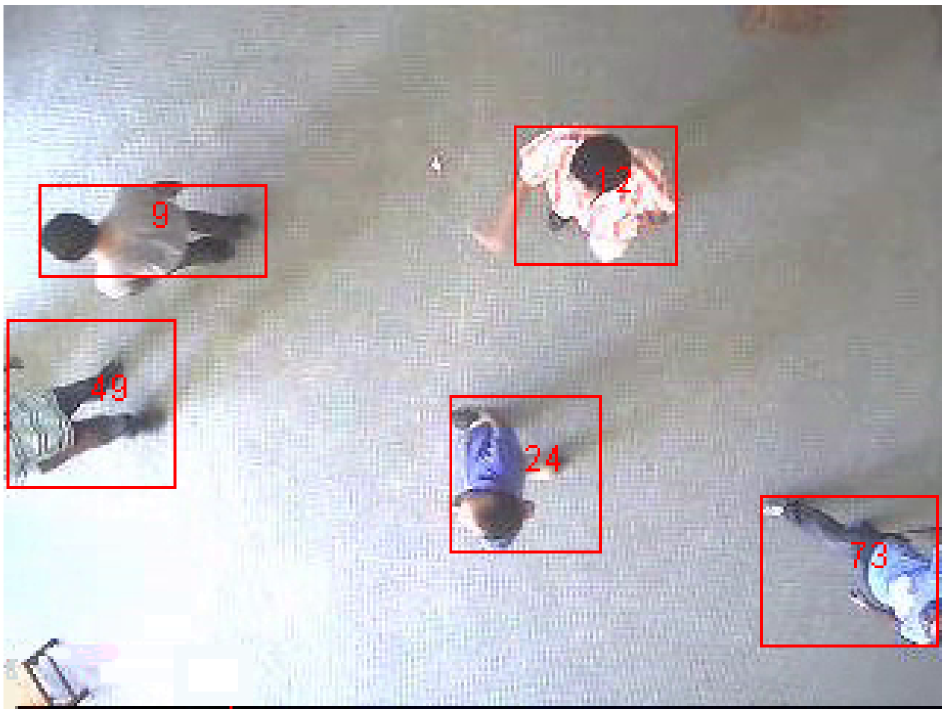


FIGURE 4.12: With the master camera at a low height, only a small area could be covered.

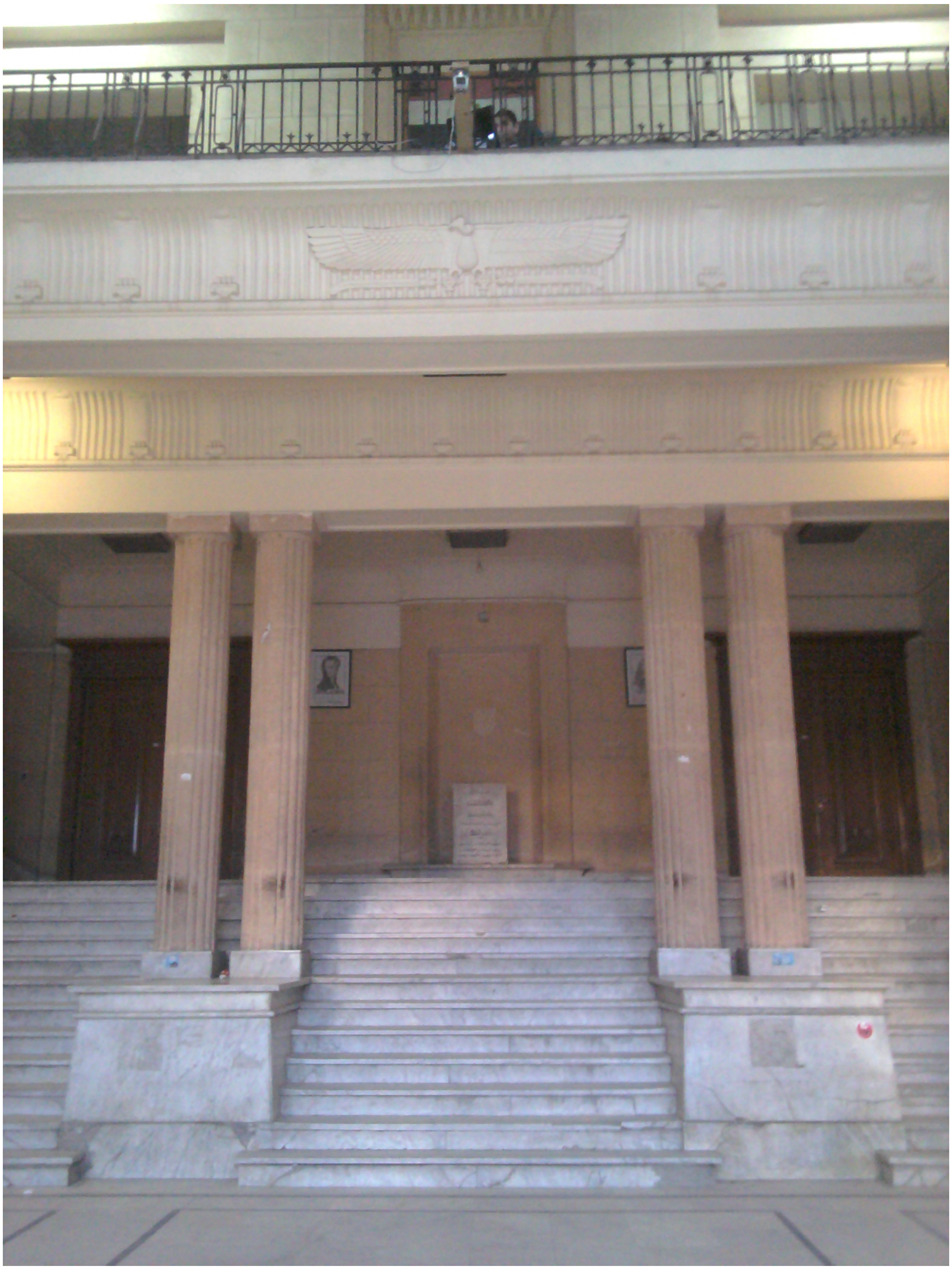


FIGURE 4.13: Master camera at height $6.5m$.

FIGURE 4.14: Below the master camera.

4.6 Autonomous Surveillance System

Upon completing the full equipment set, we moved to the main hall of the Administration Building. With an 8-port switch, a powerful mobile workstation and additional Ethernet and electric cables. We were able to install and test a full working prototype for an autonomous surveillance system.

We installed the top-view master camera hanging from the second upper floor balcony to offer a $14 \times 6m^2$ view of the ground floor from a height of $12m$. The system setup is shown in Figure 4.18 and 4.19 with a typical frame in Figure 4.20. The average coverage is shown in Table 4.6 for a test footage of over 8 minutes.

# targets	1	2	3	4	5	6
% coverage	100	99.1	98.9	97.6	96.9	95.2

TABLE 4.6: Coverage evaluation in real experiments

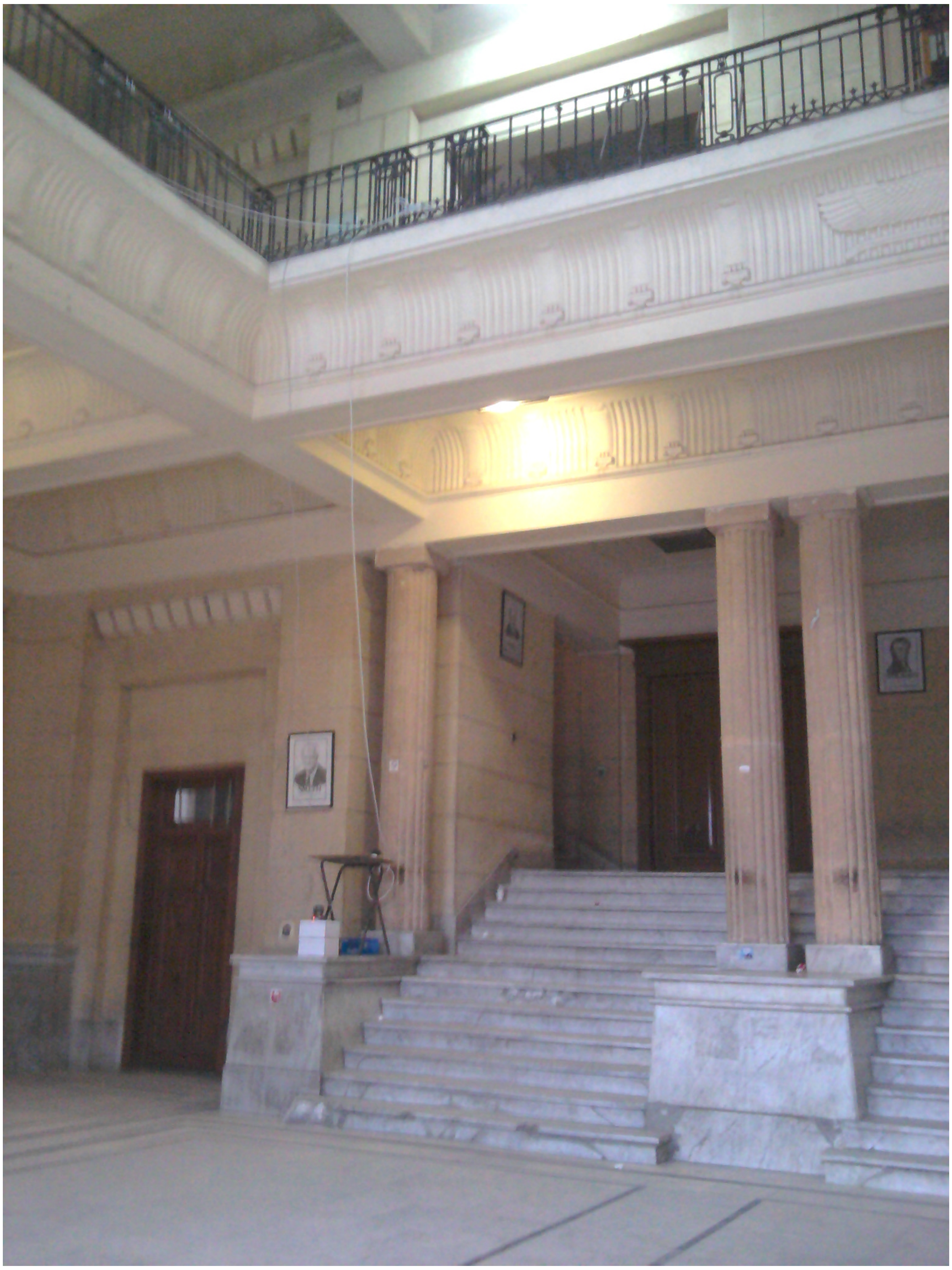


FIGURE 4.15: Slave camera (wired).

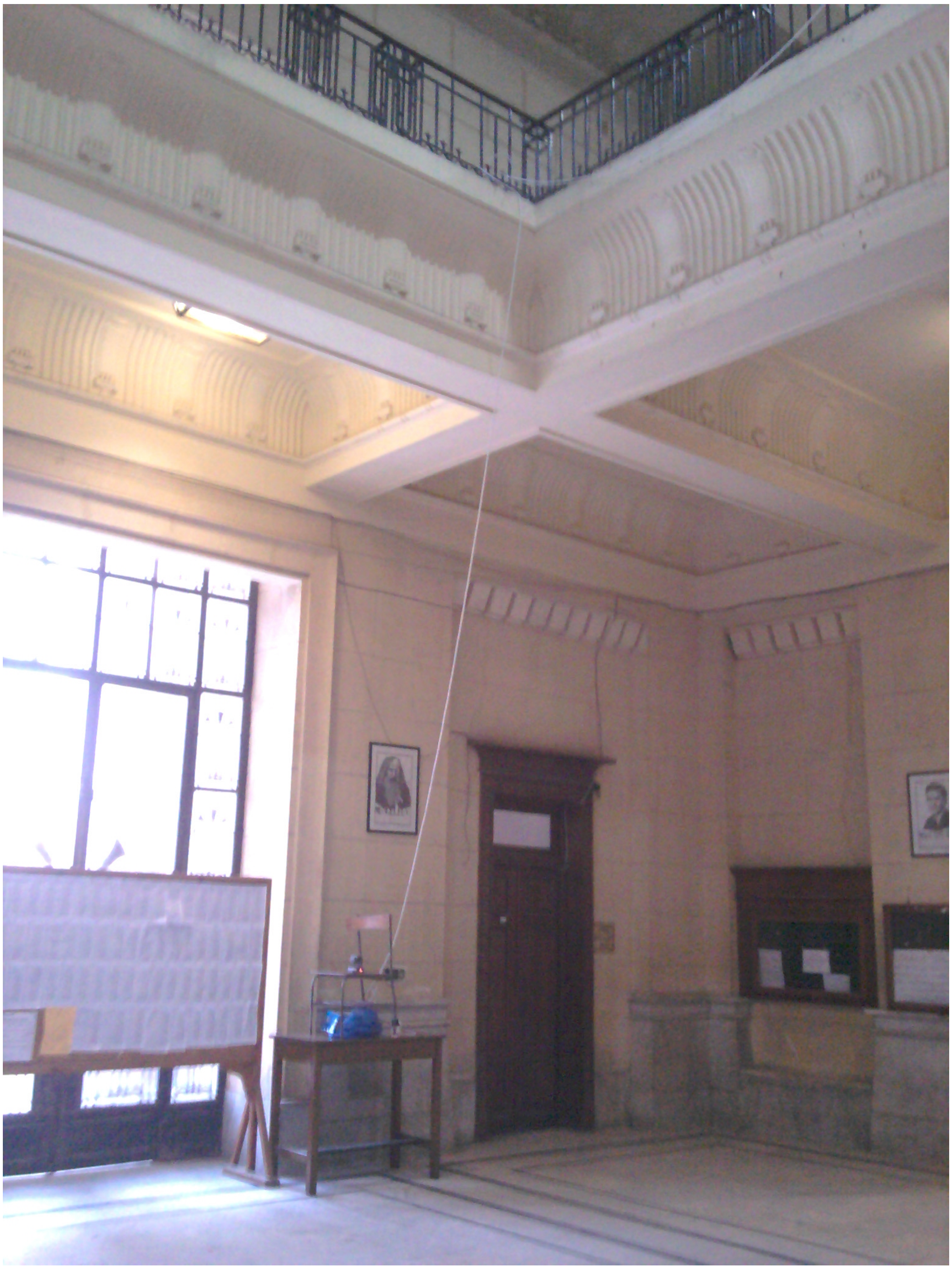


FIGURE 4.16: Slave camera (wireless).

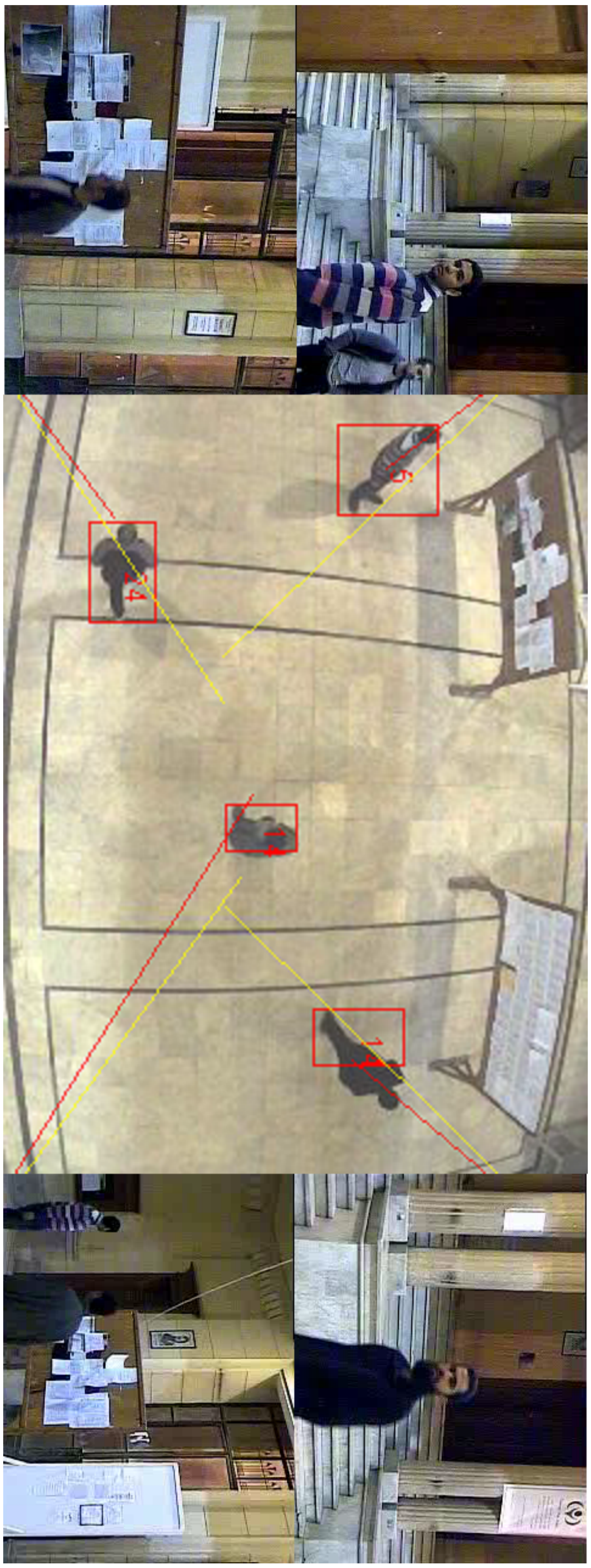


FIGURE 4.17: Example frame from an indoors human tracking experiment.

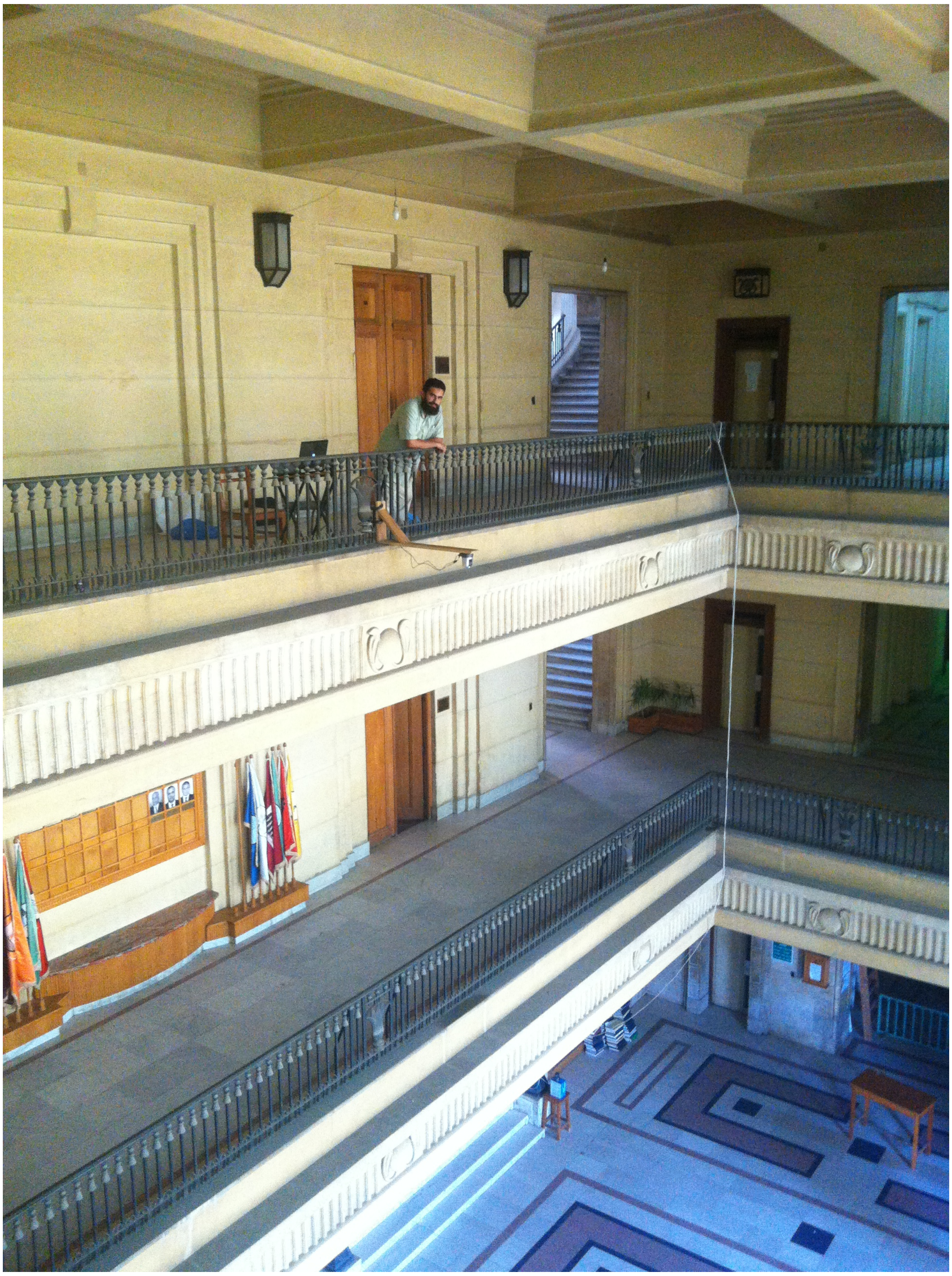


FIGURE 4.18: Main setup in Administration Building (master camera).

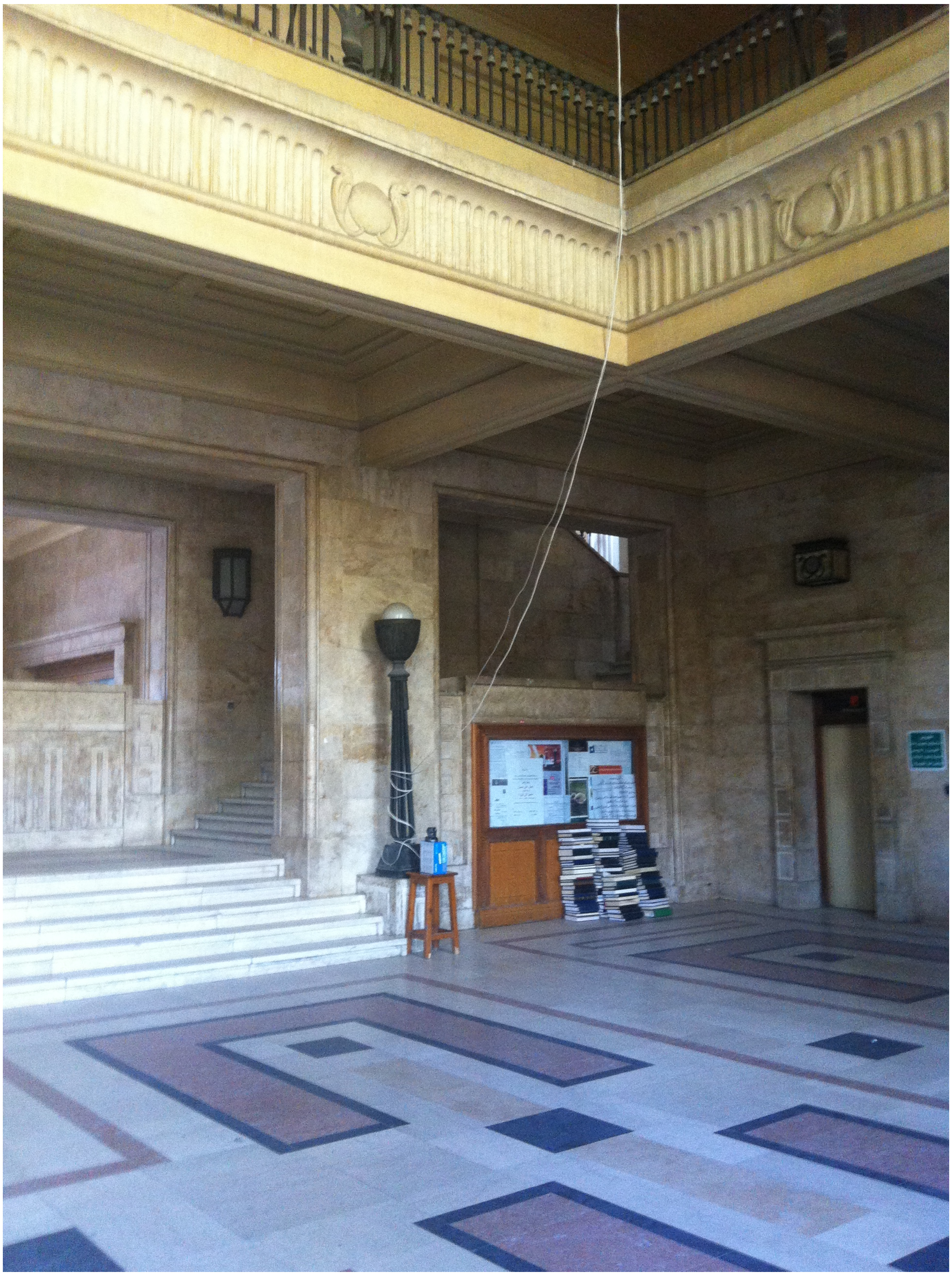


FIGURE 4.19: Main setup in Administration Building (slave camera).

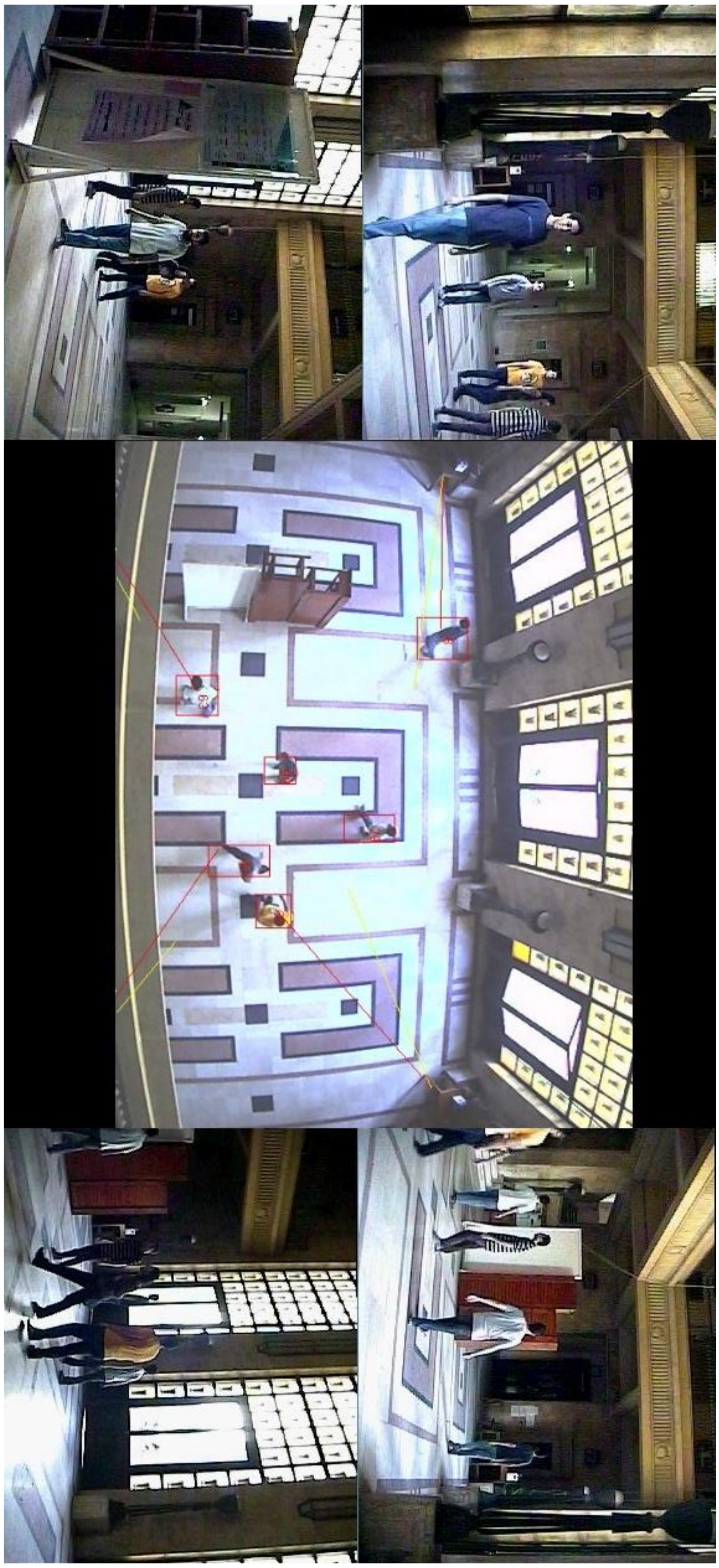


FIGURE 4.20: Example frame from the indoors experiment in Administration Building.

4.7 Conclusion

The multi-camera tracking system works as follows. The cameras are first calibrated and the slaves locations are identified in the master top-view. A lot of work on calibration has been made, such as [19, 63]. The master camera is initialized by capturing an empty scene for a few seconds to compute the background. More sophisticated techniques for background subtraction has been suggested in the computer vision community and can be incorporated in our system. However, we are more concerned with camera coordination schemes in the present work. A detailed survey of visual tracking can be found in [59]. After the background has been computed, the system runs autonomously, tracking all persons that appear in the scene and controlling the cameras according to the employed objective function. We use a variable minimum blob size and shadow removal techniques in the top tracking procedure.

We designed and implemented a real-time surveillance system that continuously tracks targets as they move into a surveyed scene. We use Vivotek pan-tilt-zoom network cameras with up to 10x zoom capabilities [62]. A top-view wide-angle low-resolution master camera, i.e., Vivotek PZ7131, is installed above the scene to provide a good view of the area, while four high zoom slave cameras, i.e., Vivotek PZ7112, are placed at the four corners of the surveyed environment to follow the human targets. The system is portable and can be easily installed in different environments. We experimented in the main hall of two buildings placing makeshift obstacles to block the view of targets. Volunteers acted as pedestrian targets moving in and out of the scene. The first set of experiments was performed with a master camera placed at a height of $6.5m$ covering a rectangle approximately $8 \times 5.5m^2$. With this first setup, we tested both coverage and zoom quality for up to 4 targets. In the second setup, the master camera was placed at a height of $12m$ to cover a larger area of approximately $14 \times 6m^2$. Maximizing target coverage is the primary objective of this study, where coverage is defined as the number of targets captured by at least on slave camera. We showed the average coverage achieved for up to 6 persons.

Chapter 5

Visibility Induction for Discretized Pursuit-Evasion

In this chapter, we move to the second variant of the visual surveillance problem we have addressed in this thesis. We study a variant of the classical pursuit-evasion game, in which an agent moving amongst obstacles is to be maintained within *sight* by a pursuing robot. This problem belongs to the more general visibility-based pursuit-evasion problems [7] where a pursuer needs to compute a path which guarantees finding an evader that might be hiding in an environment.

5.1 Problem Definition

Both players, the pursuer and the evader, are modeled as points that can move in any planar direction (holonomic robots). Each player knows exactly both the position and the velocity of the other player. We consider two-dimensional environments containing obstacles that obstruct the view of the players. Obstacles have known arbitrary geometries and locations. The assumption of complete information is used here to derive the outcome of the game, since if some player loses with complete information, it will always lose under other conditions. Both players have bounded speeds, move at different speeds and can maneuver to avoid obstacles. We will denote the maximum speed of the pursuer by v_p, that of the evader by v_e and their ratio by $r = v_e/v_p$. Players are equipped with sensors that can "see" in all directions (ommnidirectional) and as far as the environment boundaries or obstacles, whichever is closer. We will see later that we can model minimum and maximum ranges for sensors with a simple variation in our algorithm.

Deciding the game requires the construction of a binary function in two variables for the initial positions of both players. In order to study the progress of the game, we introduce a third parameter for the time index which is a discrete version of time in the continuous case. We call this function $Bad(p, e, i)$. When Bad evaluates to 1, it means there exists a strategy for an evader starting at e to go out of sight of a pursuer at p by the i^{th} time index. It is clear that $Bad(p, e, 0)$ corresponds directly to visibility constraints and is straight forward to compute. We seek an algorithm to determine the value of $Bad(p, e)$, with the time index dropped to indicate the end result of the game as time tends to infinity. In logical contexts, Bad is used as a predicate.

5.2 The Visibility Induction Algorithm

Fix a pursuer and evader at grid cells p and e and let i be the time index. If $Bad(p, e, 0) = 1$, then the evader is not initially visible to the pursuer and the game ends trivially with the pursuer losing. Now, consider the case where the evader manages to escape at step $i + 1$. To do so, the evader must move to a neighboring cell e' where no corresponding move exists for the pursuer to maintain visibility. In other words, all neighbors p' of the pursuer either cannot see the evader at e' or, otherwise, were shown unable to keep an evader at e' in sight up to step i, i.e., $Bad(p', e', i) = 1$. This means that when $Bad(p, e, i + 1)$ is updated to 1, the outcome of the game has been decided for the configuration in question as a losing one, i.e. there is a strategy for the evader to escape the sight of the pursuer at some step $\geq i$, but not any sooner. We use $\mathcal{N}(c)$ for the set of neighboring cells a player at c can move to. With that, we have the following recurrence:

$$\begin{aligned} Bad(p,e,0) &= \begin{cases} 0 & e \text{ is visible to } p \\ 1 & \text{otherwise} \end{cases} \\ Bad(p,e,i+1) &= 1 \quad \forall (p,e)\ \exists e' \in \mathcal{N}(e) \text{ s.t.} \\ &\qquad\quad \forall p' \in \mathcal{N}(p)\ Bad(p',e',i) = 1 \end{aligned} \tag{5.1}$$

Equation (5.1) computes $Bad(p, e, i+1)$ inductively by evaluating the necessary escape conditions as of the i^{th} step. For any given non-trivial initial configuration, the very first application of the inductive step would only yield a change for cases where the evader starts right next to an obstacle and is able to hide behind it immediately. That is because the $Bad(p, e, 0)$ is only 1 for initially obstructed cells. After many iterations of the induction over all cells, more pairs farther and farther from obstacles get marked as bad. The expansion of the bad region only stops at cells which the pursuer in question is able to track and the function stabilizes with $Bad(p, e) = 1$ *if and only if* the evader can win. This bears a discrete resemblance to integrating the *adjoint equations* backward in time from the termination situations as presented in [48].

Algorithm 1 performs *backward visibility induction* as defined in (5.1) to matrix M which is initialized with the visibility constraints. To determine mutual visibility between cells, we use Bresenham's line algorithm [64] to draw a line connecting every two cells and see if it passes through any obstacle. This approach works for any geometry and is easy to implement. More sophisticated algorithms could be used but are hardly justified. Restrictions on visibility can be easily incorporated by modifying the initialization part at line 5. For example, for a limited sensing range R_{max}, we add **or** $D(p, e) > R_{max}$, where D is the distance. If the pursuer is not to come any closer than a minimum distance to the evader, a lower bound R_{min} can be added as well.

Algorithm 1: Decides the game for a given map.

```
Input  : A map of the environment.
Output: The $Bad$ function encoded as a bit matrix.
Data: Two $N \times N$ binary matrices $M$ and $M'$.
1  begin
2      Discretize the map into a uniform $grid$ of $N$ cells.
       // Visibility initialization
3      Initialize $M$ and $M'$ to 0.
4      foreach $(p, e) \in grid \times grid$ do
5          if $e$ not visible to $p$ then $M[p, e] = 1$
6      end
       // Induction loop
7      while $M' \neq M$ do
8          $M' = M$
9          foreach $(p, e) \in grid \times grid$ do
10             if $\exists e' \in \mathcal{N}(e)$ s.t. $\forall p' \in \mathcal{N}(p)$ $M'[p', e'] = 1$ then $M[p, e] = 1$
11         end
12     end
13     return $M$
14 end
```

5.2.1 Correctness

We start by showing that the algorithm always terminates. At the end of each iteration, either $M' = M$ and the loop exits or more cells get marked as bad, which stops when all cells are marked, leaving $M' = M$. Next, we use this induction:

1. By line 6, M contains the correct decision at step 0 as enforced by the visibility constraints computed in line 5.
2. After the i^{th} iteration of the loop at line 9, $M[p,e] = 1$ *iff* the evader has an escape strategy e' where the pursuer has no corresponding strategy p' with $M[p',e'] = 0$.

5.2.2 Complexity Analysis

The algorithm uses $\mathcal{O}(N^2)$ storage for the output and temporary matrices M and M'. Initializing the matrices takes $\mathcal{O}(N^2\sqrt{N})$ time *if naive line drawing is used* for each pair, which is linear in the length of the line. The inner loop at line 9 processes $\mathcal{O}(N^2)$ pairs each costing $\mathcal{O}(\kappa^2)$ where $\kappa = max(|\mathcal{N}(p)|, |\mathcal{N}(e)|)$. Per the preceding discussion, at the i^{th} iteration, the algorithm decides the game for all escape paths of length $(i+1)$. Let L be the length of the longest minimal escape trajectory for the given environment. Obviously, the induction loop at line 7 is executed $\mathcal{O}(L)$ times. We can see that L depends on the largest open area in the environment and also the speed ratio r, with equal speeds being the worst case, where the distance between the players may not change, as the game ends earlier otherwise. A worst case scenario is an equally fast evader starting very close to the pursuer. For such an evader to win, it would need to move along with the pursuer to the closest obstacle where it can break visibility. We conclude that $L = \mathcal{O}(N)$ and would typically be smaller in practice. With that, the visibility induction algorithm is $\mathcal{O}(\kappa^2 N^3)$.

Theorem 5.1. (Visibility Induction) *Algorithm 1 decides the discretized game for a general environment in $\mathcal{O}(\kappa^2 N^3)$.*

Proof. By the discussion above in 5.2.1 and 5.2.2, the proof follows. □

5.3 Practicalities and Optimizations

We present several enhancements to the visibility induction algorithm and the speedups they yield. Our time measurements are performed using test maps of 100×100 cells for speed ratios $[1, \frac{4}{5}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{5}]$. All run times are averaged over 10 runs.

5.3.1 Memory Savings

As binary matrices, both M and M' need only 1 bit per entry. It is also obvious we need only store bits for valid states, which reduces N to the number of free cells. This allows N to exceed $60,000$ using less than 1 GB of memory, which enables processing at resolutions around 250×250 cells.

5.3.2 Parallelization

Observe that the loop at line 9 reads from matrix M and writes to M'. This means that M' updates are *embarrassingly parallel*. In our C++ implementation, we used the cross-platform OpenMP library to exploit this property. By adding a single line of code, we were able to harness the multiprocessing capabilities commonly available today. This allowed a 36% average speedup.

5.3.3 Caching

It is evident most of the computations are dedicated to evaluating the escape conditions as presented in (5.1). It is crucial to skip any unnecessary evaluations that yield no updates.

Lemma 5.2. $Bad(p, e, i) \implies Bad(p, e, j)\, \forall\, j > i$

Proof. For $Bad(p, e, i + \delta)$, $\delta \geq 1$, in (5.1), put $e' = e$. □

Corollary 5.3. $\neg Bad(p, e, i) \implies \neg Bad(p, e, j)\, \forall\, j < i$

Lemma 5.2 allows skipping pairs that have already been decided by a simple addition to the condition in line 10.

As Lemma 5.4 suggests, we need only re-evaluate the conditions for those players who witnessed a change at the previous iteration. This can be applied independently to pursuers and evaders leading to a 45% average speedup. When applied to both we reached 50%. This comes at an additional $\mathcal{O}(N)$ storage, which is negligible compared to the M matrix.

Lemma 5.4. (Coupled Neighborhoods)

$$\begin{aligned}\neg Bad(p,e,i) \wedge Bad(p,e,i+1) \implies&\\ &\exists (p^*,e^*) \in \mathcal{N}(p) \times \mathcal{N}(e) \textit{ s.t.}\\ &\neg Bad(p^*,e^*,i-1) \wedge Bad(p^*,e^*,i)\end{aligned}$$

Proof.

$$\begin{aligned}\neg Bad(p,e,i) &\implies \neg Bad(p,e,i-1) \qquad \text{(by C.3)}\\ &\implies \forall e' \in \mathcal{N}(e)\ \exists p^* \in \mathcal{N}(p) \text{ s.t.}\\ &\qquad \neg Bad(p^*,e',i-1)\\ &\implies \neg Bad(p^*,e^*,i-1)\end{aligned} \tag{5.2}$$

$$\begin{aligned}Bad(p,e,i+1) &\implies \exists e^* \in \mathcal{N}(e) \text{ s.t. } \forall p' \in \mathcal{N}(p)\\ &\qquad Bad(p',e^*,i)\\ &\implies Bad(p^*,e^*,i)\end{aligned} \tag{5.3}$$

By (5.2) and (5.3), the existence of (p^*, e^*) is established. □

Strict application of Lemma 5.4 results in re-evaluating the conditions only for pairs who witness *related* changes. Keeping track of that comes at a higher storage cost of $\mathcal{O}(N^2)$, which is equivalent to the M matrix. Adding the level-2 cache resulted in a 52% average speedup. With that, we reach a new complexity result. Note that caching under parallelization is particularly tricky and requires careful update and invalidation mechanisms.

Lemma 5.5. *Level-2 caching makes the induction loop* $\mathcal{O}(\kappa^4 N^2)$.

Proof. By only processing a pair (p, e) having a related update in both $\mathcal{N}(p)$ and $\mathcal{N}(e)$, no pair gets processed more than $|\mathcal{N}(p)| \times |\mathcal{N}(e)| = \mathcal{O}(\kappa^2)$ times. As the total number of pairs is $\mathcal{O}(N^2)$ and processing a single pair takes $\mathcal{O}(\kappa^2)$, this amounts to $\mathcal{O}(\kappa^4 N^2)$. □

5.3.4 More Memory Savings

It is possible to do without the auxiliary M' matrix. Instead of copying values before the inner loop and doing all checks on old values, we can use the M matrix for both checks and updates. If some entry $M[p, e]$ is not updated, the behavior would be the same. On the other hand, if $M[p, e]$ got updated, the algorithm would use a newer value instead of waiting for the next iteration which results in a minor speedup as a side-effect.

Applying all the enhancements discussed in this section led to a 64% average speedup on our test sample. We notice that for the medium sized square grids we consider, initialization of matrices is above quadratic by a small factor which is dominated by the number of iterations L. Furthermore, as κ is typically limited (players have bounded speeds) and can be considered constant for a given realization, it may be ignored in comparison to N as the algorithm approaches $\mathcal{O}(N^2)$.

For the typical case of a limited sensing region of size R, the algorithm need only consider that many evaders. This effectively replaces one N in all the above expressions and allows processing at much higher resolutions.

5.4 Experimental Results

We implemented our algorithms in C++ and performed experiments on an Intel Core i7 CPU running at 2.67GHz with 4GB of RAM. We experiment with manually created environments containing obstacles of various forms. Initial players positions are randomly selected satisfying certain visibility constraints and the evader's paths are automatically generated as discussed in the above section on tracking strategies. We discretized the maps of the used environments with regular grids of sizes ranging from 50×50 to 400×400 cells and used 4-connected and 8-connected neighborhoods. In contrast to the diamond shape of 4-connected neighborhoods, 8-connected neighborhoods are have a square shape. With such fine granularities, we anticipate moving to real world environments.

The effect of varying the grid size on the resolution of decision maps is shown in Figures 5.1 to 5.4 for speed ratios $[1, \frac{1}{2}, \frac{1}{3}, \frac{1}{5}]$. The boundaries of the nested convex regions are such that if the two players fall inside a region, the pursuer can track the evader indefinitely, while if one player is inside and the other outside, the evader can escape. The darker the gray shade, the smaller the speed ratio. Obstacles are in black and the player mentioned in each figure is the black dot roughly centered inside all nested regions.

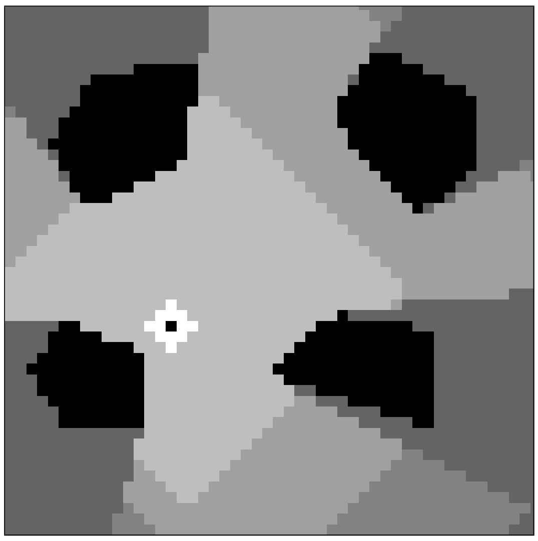


FIGURE 5.1: Decision map (pursuer) - 50x50 grid.

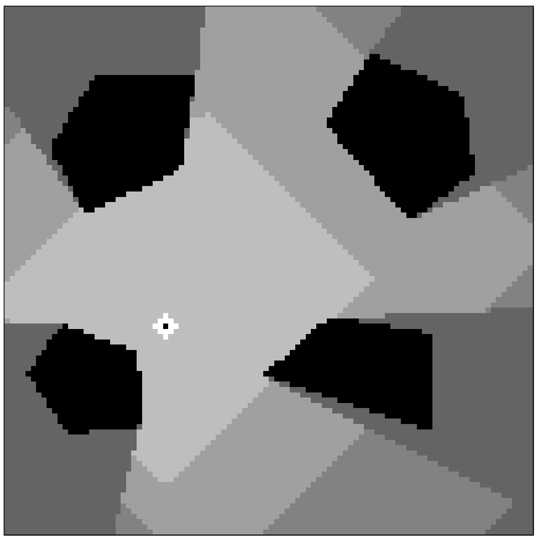


FIGURE 5.2: Decision map (pursuer) - 100x100 grid.

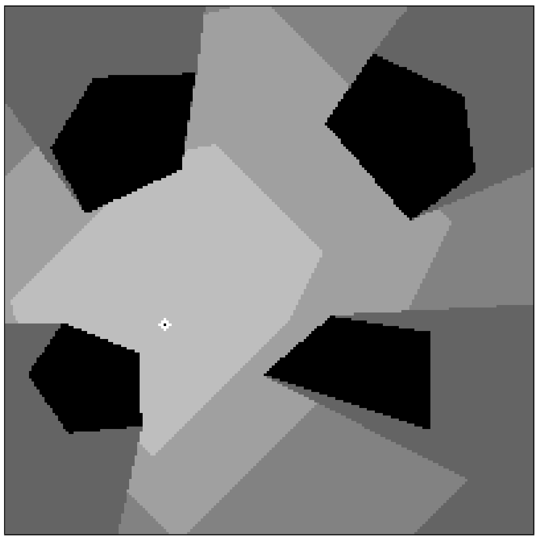


FIGURE 5.3: Decision map (pursuer) - 200x200 grid.

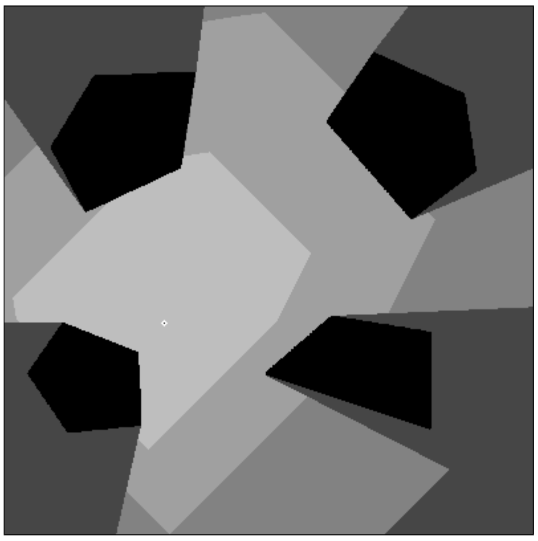


FIGURE 5.4: Decision map (pursuer) - 400x400 grid.

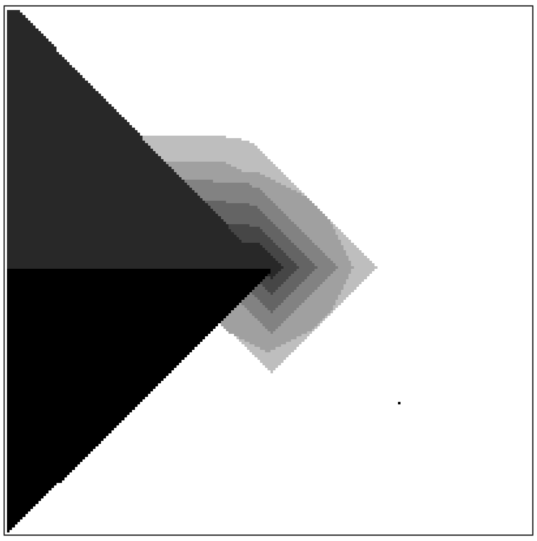


FIGURE 5.5: Decision map (pursuer) around a corner.

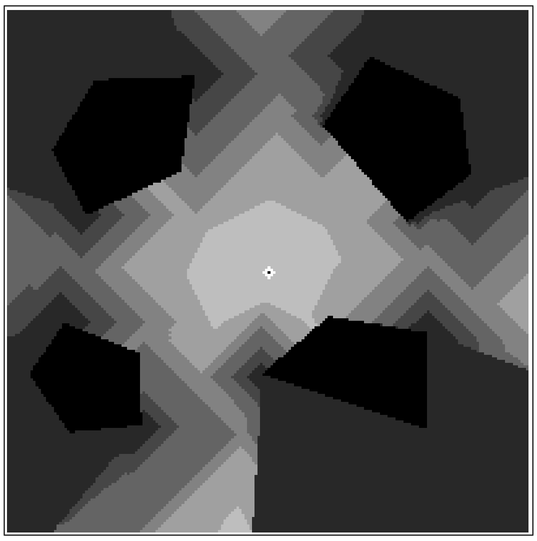


FIGURE 5.6: Decision map (pursuer) for polygonal obstacles.

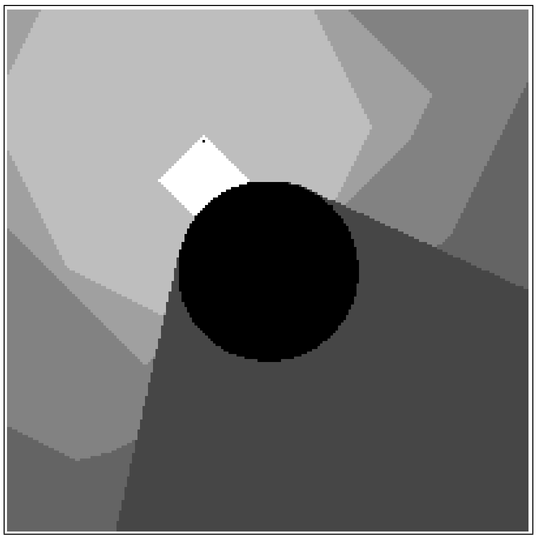


FIGURE 5.7: Decision map (evader) for a circular obstacle.

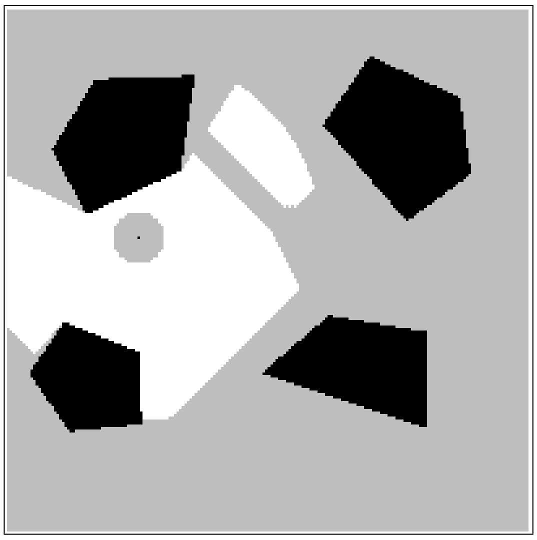


FIGURE 5.8: Decision map (evader) with restricted visibility.

Our approach works independently of obstacle shapes and layouts as shown in figures 5.5 to 5.7. Modeling visibility constraints (e.g., limited sensor range) affects the decision regions as in Figure 5.8. All previous results are computed for 4-connected neighborhoods. Figures 5.9 and 5.10 contrast 4-connected to 8-connected neighborhood maps for a speed ratio of $\frac{1}{2}$.

As we vary the grid size, runtime is affected as shown in table 5.1. It fits a quadratic model in N (for a fixed neighborhood size) per the discussion following Lemma 5.5 and seen in

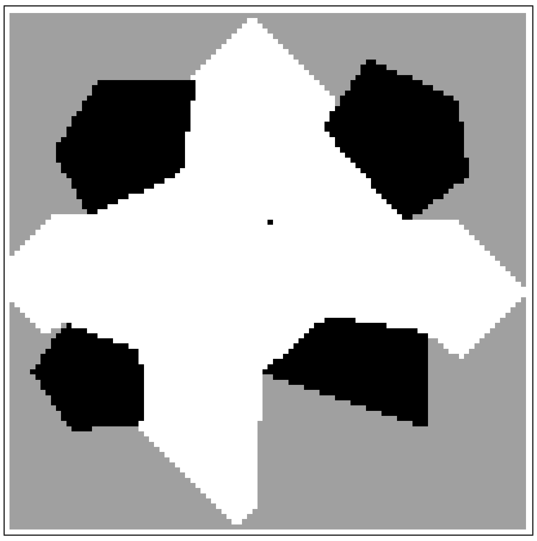


FIGURE 5.9: Decision map (evader), 4-connected neighborhood.

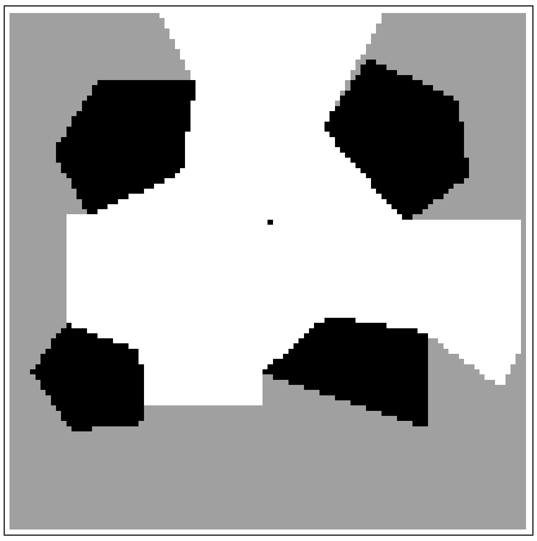


FIGURE 5.10: Decision map (evader), 8-connected neighborhood.

Figure 5.11.

TABLE 5.1: Average runtime for Algorithm 1 vs. grid size

Grid size	Runtime (hh:mm:ss)
50×50	00:00:01
60×60	00:00:02
75×75	00:00:06
100×100	00:00:23
120×120	00:00:52
150×150	00:02:27
200×200	00:09:24
300×300	01:06:18
400×400	06:47:57

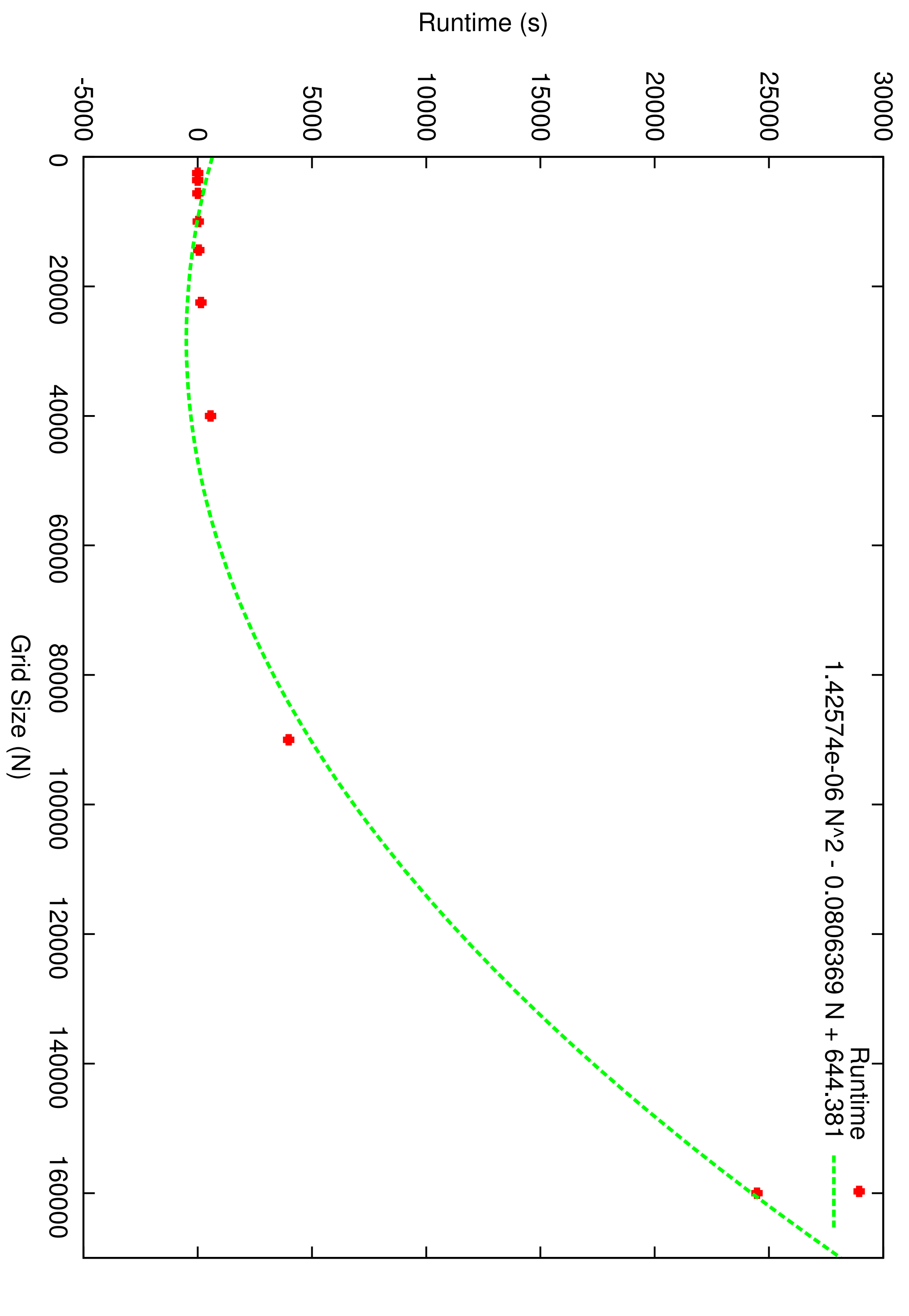


FIGURE 5.11: Average runtimes fit a quadratic model as predicted by Lemma 5.5.

Chapter 6

Extensions and Applications of Visibility Induction

In this chapter, we build upon the efficient solution we introduced for the discretized pursuit-evasion game model. Recall that the output of the visibility induction algorithm was a matrix encoding of the Bad function, which returns win-lose decisions for game configurations defined by the locations of both players.

We start by taking the Bad function a step further and using it for trajectory planning, beyond simply answering decision queries. Further objectives can also be defined as needed and an optimal trajectory can then be chosen. In particular, we are interested in optimal escape trajectories for a winning evader minimizing the visibility time. Other objectives relating to the distance between players, the distance traveled or speed of maneuvering can also be used. Next, we prove the optimality of the evader trajectories computed using an extension of the Bad matrix as being the fastest way for such an evader to break line of sight visibility and win the game. Finally, we present our own variant of the visibility-based pursuit-evasion game where the pursuer is allowed to lose sight of the evader for no more than a limited number of steps, as a relaxation to the original problem. We develop an algorithm to decide the relaxed game and show how all techniques previously developed for the original game still apply to this new one.

6.1 Extensions to Guaranteed Tracking

The computed Bad function can be used directly in basic trajectory planning for the winning player. A valid trajectory must maintain the winning state by only moving via cells with guaranteed win regardless of the strategy followed by the opponent. A higher level plan may then choose any of the valid neighbors, which are guaranteed to exist for the winner. To unify the notation used below, we define:

$$\begin{aligned} Lose(a,b) =& (Pursuer(a) \wedge Bad(a,b)) \vee \\ & (Evader(a) \wedge \neg Bad(b,a)) \end{aligned} \tag{6.1}$$

Algorithm 2 first discards invalid neighbors a winner must not move to, then chooses one of the remaining neighbors. Typically, a distance function is used for tie breaking. For example, a pursuer would generally prefer to move closer to the evader and keep it away from bad cells.

Algorithm 2: Generic trajectory planning for winners.

```
Input: $Bad(.,.)$, current state $(player, opponent)$.
1 begin
2     $\mathcal{N}^* = \{\}$
3     foreach $n \in \mathcal{N}(player)$ do
4         if $\neg Lose(n, n')\ \forall n' \in \mathcal{N}(opponent)$ then
5             $\mathcal{N}^* = \mathcal{N}^* \cup n$
6         end
7     end
8     Move to any neighbor in $\mathcal{N}^*$.
9 end
```

6.2 Optimal Trajectory Planning

If the pursuer can keep the evader in sight, there is not much the evader can do as far as we are concerned. On the other hand, if the evader can win the game, it is particularly important to minimize the time taken to break the line of sight to the pursuer. The losing pursuer must also maximize that time by not making suboptimal moves that allow the evader to escape faster. We seek an extension of the Bad function to help with this task, namely by returning the time left before visibility can be broken. We call the enhanced Bad function $J(p, e)$ as it gives the time left for visibility, which corresponds to the value of the game as in [48]. Equation 6.2 shows the modified initialization and recurrence.

To compute these optimal trajectories, we modify Algorithm 1 to store the time index i, at which the game got decided, into $M[p, e]$. In lines 5 and 10, we use 0 and i, respectively, instead of just 1, and only make the assignment once for the smallest i. The matrices are initialized to ∞ to indicate the absence of an escape strategy for the evader and the condition in line 10 is modified accordingly.

$$J(p,e,0) = \begin{cases} \infty & e \text{ is visible to } p \\ 0 & \text{otherwise} \end{cases}$$
$$J(p,e,i+1) = \min\left(J(p,e,i), 1 + \min_{e' \in \mathcal{N}(e)} \max_{p' \in \mathcal{N}(p)} J(p',e',i)\right) \tag{6.2}$$

Theorem 6.1. (Time-Optimal Trajectories) $J(p,e)$ *gives the time left before visibility is broken, assuming both players move optimally.*

Proof. Trivially, $J(p,e,i) = 0\ \forall i \Leftrightarrow e$ is not initially visible to p and the game ends immediately in 0 turns. Otherwise, if $J(p,e,i) = \infty \Leftrightarrow \forall e' \exists p' J(p',e',i-1) = \infty$ and the evader has no escape strategy of length $\leq i$ turns. Finally, $J(p,e,i) = \infty \wedge J(p,e,i+1) = k < \infty \Leftrightarrow \exists e' \exists p' J(p',e',i) = k-1$. Both players at p and e scan their neighbors to minimize (evader) and maximize (pursuer) the value of J. As the evader moves first, it follows the strategy e' achieving the minimum value over all corresponding strategies of the pursuer at p. Consequently, the pursuer has to follow the strategy p' achieving the maximum corresponding value. By the induction hypothesis, if $J(p',e',i)$ is optimal, both players are able to correctly identify the minimum and maximum strategies and the optimality of $J(p,e,i+1)$ follows. $\square$

At each step, the players will be moving to neighbors p' and e', maximizing $J(p',e)$ and minimizing $J(p,e')$, respectively. The obtained $J(p,e)$ can be processed further to have these neighbors precomputed into separate functions S_p and S_e, encoding the trajectories for each player. However, as storing the time index i increases the required storage, the algorithm can be modified to compute S_p and S_e directly. This allows reducing the storage by describing the neighbor relative to the current position of the player. The neighborhoods can be indexed unambiguously which allows the precomputed S functions to store just as many bits as necessary per entry, which is as small as $\lceil \log_2 \kappa \rceil$.

6.3 Tolerating Blind Interruptions

It might be desirable to relax the hard requirement of maintaining visibility at all times, possibly by tolerating limited durations where line of sight visibility is interrupted. This would be particularly helpful for situations where the pursuer is able to quickly regain visibility as the evader maneuvers around a corner. This could also help an evader guarantee a stronger win.

We still assume that each player knows the location of the opponent, even when line of sight visibility is broken. One can imagine that the evader is fitted with a location tracking device that reports its location to the pursuer. By tolerating relatively short interruptions to visibility, of length d, the evader would not have enough time to disable or get rid of the device and is forced to proceed with the game.

Building upon the visibility maintaining recurrence 5.1, we define a new recurrence for regaining visibility 6.3 and rename Bad to Bad_d. Notice how we exchange quantifiers, with the pursuer being the interesting player searching for the right move to regain visibility while the evader is running out of hiding options. In addition, we use zeros where the original recurrence 5.1 uses ones, as we test for non-bad neighbors and unmark cells previously considered bad.

$$\begin{aligned} Bad_d(p,e,i+1) = 0 \quad & \forall(p,e)\exists p' \in \mathcal{N}(p) \text{ s.t.} \\ & \forall e' \in \mathcal{N}(e)\ Bad_d(p',e',i) = 0 \end{aligned} \tag{6.3}$$

All we need is a simple modification to initialization step in Algorithm 1. After marking cells corresponding to initially invisible players as bad, we unmark those who are able regain visibility within d steps. This is very similar to the main induction loop, but works backwards from non-bad to bad cells per the new recurrence 6.3. With all such cells initialized as non-bad, the algorithm can now proceed with the induction to decide if the pursuer can keep the evader in sight with interruptions not exceeding d steps, using the same recurrence in 5.1.

Algorithm 3 differs from Algorithm 1 in the added induction loop at Line 7, which corresponds to attempts of regaining visibility. The loop need only be executed d times as any pursuer is only allowed up to d blind steps before it loses the relaxed game. As d is typically

Algorithm 3: Decides the game for a given map with interruptions of length $\leq d$.

```
Input  : A map of the environment, Maximum tolerable interruption $d$.
Output: The $Bad_d$ function encoded as a bit matrix.
Data: Two $N \times N$ binary matrices $M$ and $M'$.
1  begin
2      Discretize the map into a uniform $grid$ of $N$ cells.
       // Visibility initialization
3      Initialize $M$ and $M'$ to 0.
4      foreach $(p, e) \in grid \times grid$ do
5          if $e$ not visible to $p$ then  $M[p, e] = 1$
6      end
       // Induction loop: Regaining Visibility
7      while $M' \neq M \wedge d > 0$ do
8          $M' = M$
9          foreach $(p, e) \in grid \times grid$ do
10             if $\exists p' \in \mathcal{N}(p)$ s.t.$\forall e' \in \mathcal{N}(e)$ $M'[p', e'] = 0$ then  $M[p, e] = 0$
11         end
12         Decrement $d$
13     end
       // Induction loop: Pursuit-Evasion
14     while $M' \neq M$ do
15         $M' = M$
16         foreach $(p, e) \in grid \times grid$ do
17             if $\exists e' \in \mathcal{N}(e)$ s.t.$\forall p' \in \mathcal{N}(p)$ $M'[p', e'] = 1$ then  $M[p, e] = 1$
18         end
19     end
20     return $M$
21 end
```

small, i.e., $d \ll N$, the main induction loop which is common between the two algorithm still dominates the runtime. The complexity analysis presented for the original visibility induction algorithm in 5.2.2 applies here as well. We conclude that Algorithm 3 is also $\mathcal{O}(\kappa^2 N^3)$.

Note that it might be the case that multiple interruptions occur during the course of the game, but as long as each interruption spans no more than d steps, the evader does not win. It is also possible that the pursuer manages to regain visibility after the first interruption, but fails to do so on the second and ends up losing. Interestingly, the same trajectory planning algorithms presented above still applies after this change. The modified algorithm is described above in Algorithm 3.

6.4 Experimental Results

We show a selection of tracking scenarios in Figures 6.1 to 6.5. Figure 6.1 shows an environment with non-polygonal obstacles as an example of the flexibility of our proposed method. In Figure 6.2, we modified the objective function of the purser to maximize the distance to the evader as long as it can see it. We can see that even though it initially moved away, it was guaranteed to keep the evader in sight and the computed trajectory achieved that.

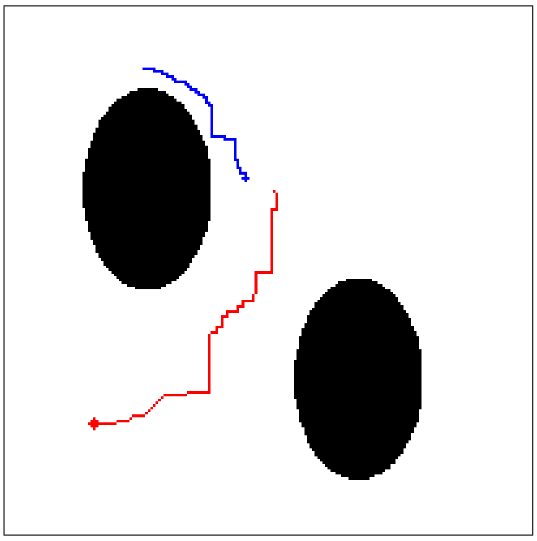


FIGURE 6.1: Example of a winning evader (blue; top).

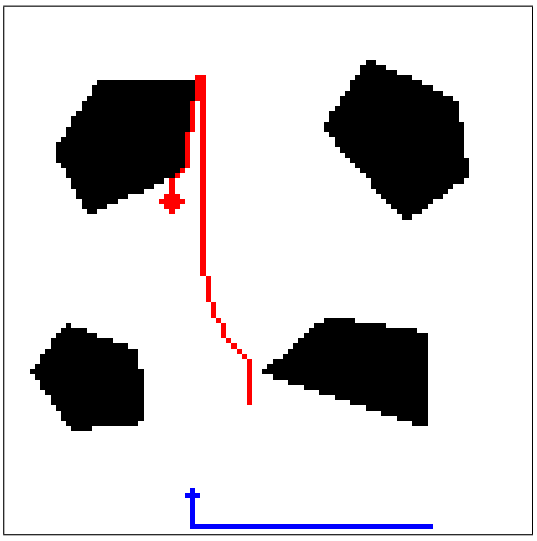


FIGURE 6.2: Example of a winning pursuer (red; top).

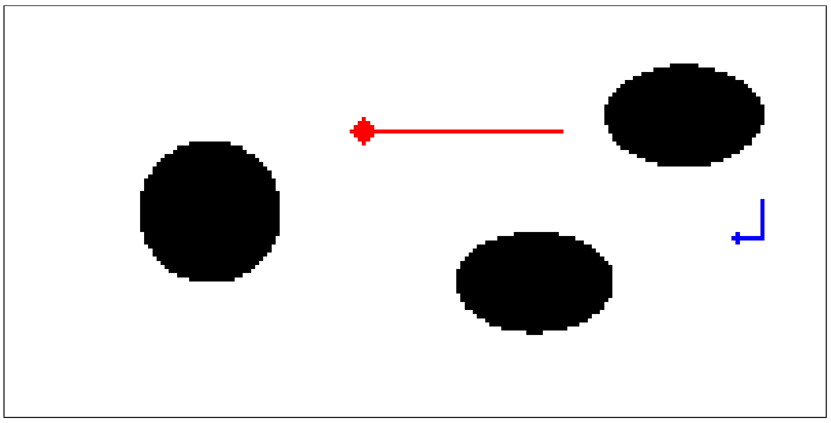


FIGURE 6.3: A winning evader on a rectangular grid with non-polygonal obstacles.

Finally, we show examples of regaining visibility around an infinite corner in Figures 6.6A and 6.6B. For these two sample cases, the evader was not even initially visible to the pursuer.

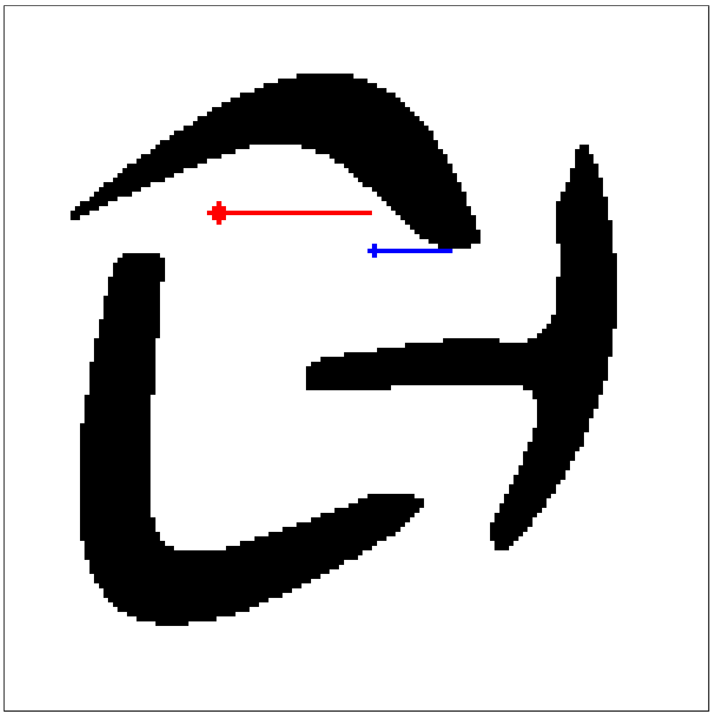


FIGURE 6.4: A winning evader in a complex maze.

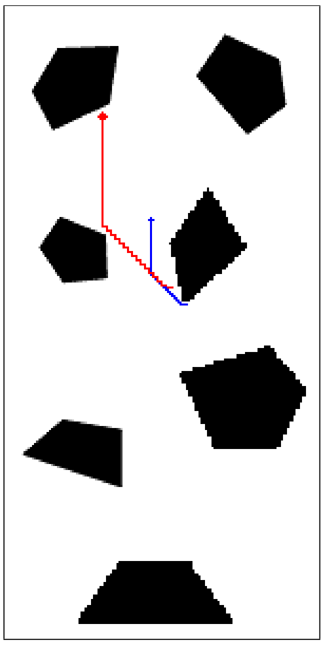


FIGURE 6.5: A winning evader on a rectangular grid.

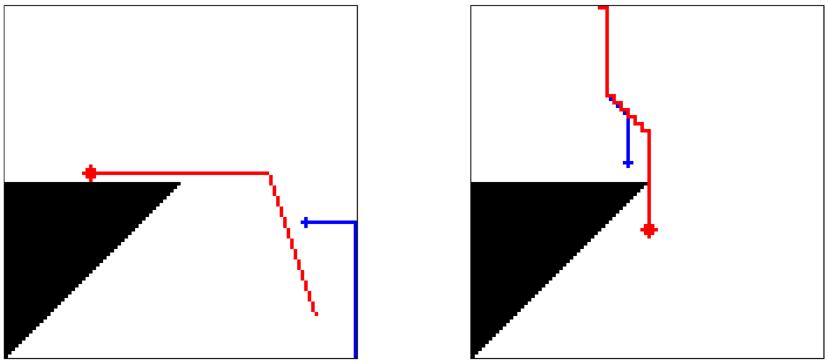

(A) (B)

FIGURE 6.6: Example of a pursuer regaining visibility around an infinite corner.

The pursuer correctly maneuvers around the corner and proceeds with the pursuit, minimizing the distance to the pursuer as a secondary objective.

Chapter 7

Conclusion and Future Work

7.1 Coverage Maximization in Multi-Camera Surveillance

We have addressed the problem of covering as many targets as possible, moving through obstacles, using a set of PTZ cameras (or directional sensors). Since an optimal solution to this problem is computationally expensive, we have compared several suboptimal heuristics and designed two new ones. Experimental results have shown that our heuristics supersede earlier approaches in terms of coverage rates. Moreover, our angular heuristics not only increase coverage but also have the advantage of providing continuous panning actions for cameras. This makes it more smooth in real world experimentation with PTZ cameras.

Future Work

In the future, we plan developing heuristics that would provide a more stable system by reducing the sum of the angles that the cameras have to pan to cover the targets. In real surveillance systems, we seek to obtain better views of targets (orientation facing camera, illumination) and to trigger camera panning by scene events (rather than by mere people motion). Plug-in applications (such as face/plate recognition) can benefit from the zoom-in view from the slave cameras, however these require a better iris focus specially at high zooms.

7.2 Visibility-based Pursuit-Evasion

We addressed the problem of maintaining an unobstructed view of an evader moving amongst obstacles by a pursuer. In particular, we developed an algorithm that decides the outcome of the game for any pair of initial positions of the players. By employing discretization to both a space and time, the solution becomes feasible in polynomial time, is independent of the geometry of the environment and does not require the use of heuristics. We give a detailed analysis for the correctness of the algorithm, derive its space and time complexities and present and verify several approaches to reduce its time and memory demands. We extended our algorithm to compute tracking or escape trajectories for both players in real time and verified their optimality. We tested our method on different tracking scenarios and environments.

Future Work

To model the real world more accurately, we consider hexagonal mesh discretization, with several interesting properties, and study decision errors for a given resolution. We already implemented limited range sensors and an extension to a limited field of view is systematic. Other constraints on players' motion can be considered such as restricted areas where the pursuer is not allowed to go into. Efficiently updating the computed strategies to adapt to changes in the environments is also an interesting direction for real world scenarios. Finally, we envision efficient ways to extend this approach to the case of more players.

Bibliography

[1] Tomi D Raty. Survey on contemporary remote surveillance systems for public safety. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(5):493–515, 2010.

[2] Aswin C. Sankaranarayanan, Ashok Veeraraghavan, and Rama Chellappa. Object detection, tracking and recognition for multiple smart cameras. *Proceedings of the IEEE*, 96(10):1606–1624, 2008.

[3] Hazem El-Alfy, David Jacobs, and Larry Davis. Assigning cameras to subjects in video surveillance systems. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'09)*, pages 837–843, Kobe, Japan, May 2009.

[4] David Hsu, Wee Sun Lee, and Nan Rong. A point-based pomdp planner for target tracking. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2644–2650. IEEE, 2008.

[5] Sourabh Bhattacharya and Seth Hutchinson. Approximation schemes for two-player pursuit evasion games with visibility constraints. *Proceedings of Robotics: Science and Systems IV*, 2008.

[6] Rafael Murrieta-Cid, Teja Muppirala, Alejandro Sarmiento, Sourabh Bhattacharya, and Seth Hutchinson. Surveillance strategies for a pursuer with finite sensor range. *The International Journal of Robotics Research*, 26(3):233–253, 2007.

[7] Steven Michael LaValle. *Planning algorithms*. Cambridge university press, 2006.

[8] Brian P Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. *The International Journal of Robotics Research*, 25(4):299–315, 2006.

[9] Andreas Kolling and Stefano Carpin. Multi-robot pursuit-evasion without maps. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3045–3051. IEEE, 2010.

[10] Kyle Klein and Subhash Suri. Complete information pursuit evasion in polygonal environments. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI)*, pages 1120–1125, 2011.

[11] Richard Borie, Craig Tovey, and Sven Koenig. Algorithms and complexity results for graph-based pursuit evasion. *Autonomous Robots*, 31(4):317–332, 2011.

[12] Steven M LaValle, Hector H González-Banos, Craig Becker, and J-C Latombe. Motion strategies for maintaining visibility of a moving target. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pages 731–736. IEEE, 1997.

[13] Rafael Murrieta-Cid, Raul Monroy, Seth Hutchinson, and J-P Laumond. A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2657–2664. IEEE, 2008.

[14] Hazem El-Alfy and Amr Kabardy. A new approach for the two-player pursuit-evasion game. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on*, pages 396–397. IEEE, 2011.

[15] Vikram P. Munishwar and Nael B. Abu-Ghazaleh. Scalable target coverage in smart camera networks. In *Fourth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'10)*, pages 206–213. ACM, 2010.

[16] Yanli Cai, Wei Lou, Minglu Li, and Xiang-Yang Li. Target-oriented scheduling in directional sensor networks. In *26th IEEE International Conference on Computer Communications (INFOCOM'07)*, pages 1550–1558, 2007.

[17] Jing Ai and Alhussein A. Abouzeid. Coverage by directional sensors in randomly deployed wireless sensor networks. *Journal of Combinatorial Optimization*, 11(1):21–41, 2006.

[18] David W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, 2007.

[19] Fatih Porikli and Ajay Divakaran. Multi-camera calibration, object tracking and query generation. In *Proc. International Conference on Multimedia and Expo (ICME'03)*, pages 653–656, Baltimore, MD, USA, July 2003.

[20] Robert T. Collins, Alan J. Lipton, Hironobu Fujiyoshi, and Takeo Kanade. Algorithms for cooperative multisensor surveillance. *Proceedings of the IEEE*, 89(10):1456–1477, October 2001.

[21] Yong Rui, Anoop Gupta, Jonathan Grudin, and Liwei He. Automating lecture capture and broadcast: technology and videography. *Multimedia Systems*, 10(1):3–15, 2004.

[22] Dmitry Rudoy and Lihi Zelnik-Manor. Posing to the camera: automatic viewpoint selection for human actions. 6495:307–320, 2011.

[23] Eric Sommerlade and Ian Reid. Probabilistic surveillance with multiple active cameras. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'10)*, pages 440–445, Anchorage, AK, USA, May 2010.

[24] Prabhu Natarajan, Trong Nghia Hoang, Kian Hsiang Low, and Mohan Kankanhalli. Decision-theoretic approach to maximizing observation of multiple targets in multi-camera surveillance. In *Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, Valencia, Spain, June 2012.

[25] Noriko Takemura and Jun Miura. View planning of multiple active cameras for wide area surveillance. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'07)*, pages 3173–3179, Roma, Italy, April 2007.

[26] Loren Fiore, Duc Fehr, Robot Bodor, Andrew Drenner, Guruprasad Somasundaram, and Nikolaos Papanikolopoulos. Multi-camera human activity monitoring. *Journal of Intelligent & Robotic Systems*, 52(1):5–43, 2008.

[27] Faisal Z. Qureshi and Demetri Terzopoulos. Planning ahead for ptz camera assignment and handoff. In *Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'09)*, pages 1–8, Como, Italy, September 2009.

[28] Y. Li and B. Bhanu. Utility-based camera assignment in a video network: A game theoretic framework. *Sensors Journal, IEEE*, (99):1–1, 2010.

[29] Giordano Fusco and Himanshu Gupta. Selection and orientation of directional sensors for coverage maximization. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON'09. 6th Annual IEEE Communications Society Conference on*, pages 1–9. IEEE, 2009.

[30] David Lichtenstein. Planar formulae and their uses. *SIAM journal on computing*, 11 (2):329–343, 1982.

[31] Matthew P Johnson and Amotz Bar-Noy. Pan and scan: Configuring cameras for coverage. In *INFOCOM, 2011 Proceedings IEEE*, pages 1071–1079. IEEE, 2011.

[32] Vikram P Munishwar and Nael B Abu-Ghazaleh. Coverage algorithms for visual sensor networks.

[33] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.

[34] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[35] Héctor H González-Banos, David Hsu, and Jean-Claude Latombe. Motion planning: Recent developments. *Autonomous Mobile Robots: Sensing, Control, Decision-Making and Applications*, 2006.

[36] Geňa Hahn. Cops, robbers and graphs. In *Graphs' 04: proceedings of the Czech-Slovak Conference on Graphs, Vyšné Ružbachy, May 23-28, 2004*, volume 36, pages 163–176. Mathematical Institute, Slovak Academy of Sciences, 2007.

[37] Geňa Hahn and Gary MacGillivray. A note on k-cop, l-robber games on graphs. *Discrete mathematics*, 306(19):2492–2497, 2006.

[38] Toru Ishida and Richard E. Korf. Moving-target search: a real-time search for changing goals. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(6):609–619, 1995.

[39] Carl H Fitzgerald. The princess and monster differential game. *SIAM Journal on Control and Optimization*, 17(6):700–712, 1979.

[40] Brian H Bowditch. The angel game in the plane. *Combinatorics Probability and Computing*, 16(3):345–362, 2007.

[41] Rufus Isaacs. Differential games: A mathematical theory with applications to warfare and pursuit, control and optimization. 1965.

[42] Shmuel Gal. Search games with mobile and immobile hider. *SIAM Journal on Control and Optimization*, 17(1):99–122, 1979.

[43] Jon Pitchford. Applications of search in biology: Some open problems. In *Search Theory*, pages 295–303. Springer, 2013.

[44] Steve Alpern and Thomas Lidbetter. Mining coal or finding terrorists: The expanding search paradigm. *Operations Research*, 61(2):265–279, 2013.

[45] Zhengyu Yin, A Jiang, M Johnson, Milind Tambe, Christopher Kiekintveld, Kevin Leyton-Brown, Tuomas Sandholm, and John Sullivan. Trusts: Scheduling randomized patrols for fare inspection in transit systems. In *Proc. of the 24th Conference on Innovative Applications of Artificial Intelligence, IAAI*, 2012.

[46] Sourabh Bhattacharya, Salvatore Candido, and Seth Hutchinson. Motion strategies for surveillance. *Proceedings of Robotics: Science and Systems, Atlanta, GA, USA*, 2, 2007.

[47] Sourabh Bhattacharya, Seth Hutchinson, and Tamer Basar. Game-theoretic analysis of a visibility based pursuit-evasion game in the presence of obstacles. In *American Control Conference, 2009. ACC'09.*, pages 373–378. IEEE, 2009.

[48] Sourabh Bhattacharya and Seth Hutchinson. On the existence of nash equilibrium for a two player pursuit-evasion game with visibility constraints. In *Algorithmic Foundation of Robotics VIII*, pages 251–265. Springer, 2009.

[49] Ryo Takei, Richard Tsai, Zhengyuan Zhou, and Yanina Landa. An efficient algorithm for a visibility-based surveillance-evasion game. Technical report, DTIC Document, 2012.

[50] Carsten Moldenhauer and Nathan R Sturtevant. Evaluating strategies for running from the cops. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 584–589. Morgan Kaufmann Publishers Inc., 2009.

[51] Carsten Moldenhauer and Nathan R Sturtevant. Optimal solutions for moving target search. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1249–1250. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[52] Sven Koenig, Maxim Likhachev, and Xiaoxun Sun. Speeding up moving-target search. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 188. ACM, 2007.

[53] Alejandro Isaza, Jieshan Lu, Vadim Bulitko, and Russell Greiner. A cover-based approach to multi-agent moving target pursuit. In *Artificial Intelligence and Interactive Entertainment Conference (AIIDE)*, 2008.

[54] P. Raghavan and C.D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[55] N.E. Young. Randomized rounding without solving the linear program. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 170–178. Society for Industrial and Applied Mathematics, 1995.

[56] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, pages 233–235, 1979.

[57] M. Gairing. Covering games: Approximation through non-cooperation. *Internet and Network Economics*, pages 184–195, 2009.

[58] M Powell. Jmonkey engine. *Avaliable in: http://www. jmonkeyengine. com*, 2008.

[59] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing Surveys (CSUR)*, 38(4):13, 2006.

[60] Wilhelm Burger and Mark James Burge. *Principles of Digital Image Processing*, volume 2. Springer, 2009.

[61] Rhys Hill, Christopher Madden, Anton van den Hengel, Henry Detmold, and Anthony Dick. Measuring latency for video surveillance systems. In *Digital Image Computing: Techniques and Applications, 2009. DICTA'09.*, pages 89–95. IEEE, 2009.

[62] Vivotek, Inc. Vivotek IP Network PTZ Camera data sheet, PZ-7112, PZ-7131, 2012. URL `http://www.vivotek.com`.

[63] R.Y. Tsai. An efficient and accurate camera calibration technique for 3d machine vision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1986*, 1986.

[64] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

الملخص

تختص دراسة نظم المراقبة البصرية بالسبل المثلى للاستفادة من الكاميرات الرقمية الحديثة لتحقيق أهداف المراقبة المطلوبة. وبفضل توافر الكاميرات الحديثة منخفضة التكلفة وعالية الكفاءة، صار هناك انتشار واسع فى تبنى هذه الأنظمة كما يرى في المطارات والبنوك ومحطات القطار. كذلك تلعب الكاميرات الأكثر دقة دورا حاسما في الروبوتات ذاتية الحركة مثل الطائرات بدون طيار. في هذه الأطروحة، نقوم بدراسة نوعين مختلفين من مشكلة المراقبة البصرية.

في الجزء الأول، ندرس السبل المثلى للتحكم في كاميرات المراقبة القادرة على الدوران وتغيير مجال الرؤية من أجل تغطية أكبر عدد من الأهداف بدقة عالية. تعتبر هذه المشكلة نظيرا هندسيا لمشكلة التغطية الكلاسيكية فى علم التحسين التوافقي، وقد تم إثبات صعوبة الحصول على حل مثالى لها بطريقة حسابية. نقوم بتطوير اثنين من البدائل التقريبية، ثم نقارنها بالحلول المعروفة للمشكلة وإظهار تفوقها عن طريق المحاكاة. وعلاوة على ذلك، نقوم ببناء وتقييم نظام مراقبة الحقيقي لتتبع المشاة بداخل عدد من مباني الكلية باستخدام الطرق المقترحة.

في الجزء الثاني، ندرس أحد صور لعبة المطاردة الكلاسيكية في علم المباريات، والذى يتعلق بالتحكم فى الروبوت المطارد لكى يحافظ على العميل الهارب فى مجال رؤيته. نقوم بتصميم خوارزمية فعالة تحدد إذا كان الهارب يمكنه أن يتخذ أي استراتيجية تمكنه من الاختباء من المطارد. وإذا وجد أنه يمكنه الهروب بالفعل، نقوم بحساب مسار الهروب المثالي بالإضافة لمسار المطارد المثالي. كما نقوم بتحليل الخوارزمية واستنتاج العديد من التحسينات والتطبيقات، ثم نستعرض نتائج تنفيذ الخوارزمية لعدد من خرائط المحاكاة.

يحتوي الباب الأول على مقدمة ودوافع البحث المقترح في الرسالة وتلخيص المساهمات المقترحة. ثم يستعرض الباب الثاني الخلفية العلمية لكل من المشكلتين المطروحتين للدراسة بالإضافة للحلول التي تم اقتراحها من قبل لكل مشكلة وبعض المشكلات الشبيهة. تبدأ دراسة المشكلة الأولى في الباب الثالث حيث يتم اقتراح حلين. يعتمد الحل الأول على تعريف دالة جديدة متصلة للتغطية بينما يقدم الحل الثاني طريقة أكثر كفاءة في اختيار مجال الرؤية لكل كاميرا. ويتضمن الباب مقارنة الحلول الجديدة بعدد من الحلول المعروفة عن طريق المحاكاة. ويلخص الباب الرابع التجارب العملية التي أجريت لتطبيق طرق التغطية المقترحة على نظام مراقبة حقيقي ويتم شرح الأساليب الإضافية المطلوبة لتعقب الأهداف ومواصفات الكاميرات المستخدمة. ننتقل لدراسة المشكلة الثانية في الباب الخامس حيث نقم بتطوير وتحليل الحل المقترح والذي يعتمد على تعريف دالة منطقية لتوصيف امكانية هروب الهدف على شكل علاقة تكرارية كما يتم عرض أمثلة تطبيقه على عدد من خرائط المحاكاة. ثم يقدم الباب السادس عددا من التطبيقات المشتقة من الحل المقترح مثل حساب مسار الحركة وزمن الهروب المثالي بالإضافة إلى إمكانية استعادة الهدف في مجال الرؤية بعد هروبه. ويختتم الباب السابع بملخص للمساهمات والنتائج المقدمة ويطرح عددا من الاتجاهات المستقبلية لاستكمال دراسة مواضيع البحث.