University of California
Santa Barbara

Camera Coordination for Intruder Detection in 1D Environments

A Thesis submitted in partial satisfaction of the
requirements for the degree Master of Science
in Mechanical Engineering

by

Jeffrey R. Peters

Committee in charge:
Professor Francesco Bullo, Chair
Professor Brad Paden
Professor Bassam Bamieh

July 10, 2013

The thesis of Jeffrey R. Peters is approved.

Francesco Bullo,Chair

Brad Paden

Bassam Bamieh

August 2013

Abstract

Camera Coordination for Intruder Detection in 1D Environments

BY

Jeffrey R Peters

This work proposes surveillance trajectories for a network of active (pan-tilt-zoom) cameras to detect intruders. We consider *smart* intruders, which appear at arbitrary times and locations, are aware of the cameras configuration, and move to avoid detection for as long as possible. As performance criteria we consider the worst-case detection time and the average detection time. We focus on the case of a chain of cameras, and we obtain the following results. First, we characterize a lower bound on the worst-case and on the average detection time of smart intruders. Second, we propose a team trajectory for the cameras, namely *Equal-waiting trajectory*, with minimum worst-case detection time and with guarantees on the average detection time. Third, we design a distributed algorithm to coordinate the cameras along an Equal-waiting trajectory. Fourth, we design a distributed algorithm for cameras reconfiguration in the case of failure or network change. Finally, we illustrate the effectiveness and robustness of our algorithms via simulations and experiments on a camera network testbed.

Acknowledgements

This work was done in collaboration with Fabio Pasqualetti, Filippo Zanella, Markus Spindler, Ruggero Carli, and Francesco Bullo. I would like to thank all of these people for their contributions to the work, and extend a special thanks to Fabio Pasqualletti and Francesco Bullo for their support of my academic endeavors.

Contents

List of Figures

11 In Fig. 11(a) we show the detection times for our second experiment, in which smart intruders appear at *worst case* times and locations. The detection times of this experiment are depicted by a solid blue line. Notice that the detection times are smaller than the upper bound predicted in Section 2. In Fig. 11(b) we show the detection times for each trial of our third experiment, in which smart intruders appear at *random* times and locations. The detection times of this experiment are depicted by a solid blue line. The solid black line corresponds to the average of the experimental detection times. For the considered configuration of cameras, the lower bound ADT* in (5) (dashed green line), ADT(X^{eq}) as in (6) (dotted red line), and the worst case upper bound (dashed light blue line), which is calculated by multiplying the lower bound on ADT* by the quantity $\frac{\tau^{\max}+\tau^{\min}}{2\tau^{\min}}$ from (7) are reported. 47

12 Simulation of REC with $n = 5$ cameras with maximum speed $v^{\max} = 0.67$ m/s and patrolling windows constraints. In Fig. 12(a) we show the cameras trajectories starting from random positions. The dashed lines refer to the trajectories of the active boundaries. In Fig. 12(b) we report the dynamics of the longest patrolling time $\hat{\tau}_i^{\max}$. Notice that $\hat{\tau}_i^{\max}$ converges to the optimal value $\tau^* = 6.2023$ s (dash-dot line). 52

List of Tables

1 Introduction

Coordinated teams of mobile agents have recently been used for many tasks requiring continuous execution, including the monitoring of oil spills [1], the detection of forest fires [2], the tracking of border changes [3], and the patrolling of environments [4]. The use of mobile agents has several advantages with respect to the classic approach of deploying a large number of static sensors, such as improved situation awareness and fast reconfigurability.

Decreasing hardware costs, along with ever increasing efficiency of numerical algorithms has garnered increased interest in research problems relating specifically to the use of camera networks for patrolling tasks. Camera networks can be take a variety of forms. However, camera networks usually consist of a set of IP or network cameras which are utilized in order to gain visual data about a specific region of interest [5]. These networks differ from other sensor networks in the sense that cameras are directional sensors, whose sensing range is very suceptable to visual obstructions in the region of interest. This, combined with the high amounts of uncertainty and large computational costs that are inherent in image processing applications, makes for a unique class of challenging research problems.

Since surveillance areas of interest can be very large, camera networks often contain many cameras. With the high computational complexity that is involved with processing image streams in real time (generally 30 FPS), centralized algorithms are not always feasible. Therefore, there is significant interest in development of distributed strategies for accomplishing tasks in large camera networks. Along with the advantage of faster processing, distributed algorithms are often advantageous in

camera networks by eliminating the need to transmit data to a central processor, which might be impossible in certain cases.

Camera network applcations usually contain both computer vision and control aspects. Control theory is generally used to determine what is the best way to obtain data, and how to react to the conclusions that are drawn from the data. Computer vision tools are generally used to analyze visual data, and draw conclusions despite the large amounts of uncertainty. Although almost all camera network problems contain aspects from both of these fields, most research literature focuses on one of the two. In this thesis, we will focus on the specific control problem of coordinating camera motion to detect intruders in a specific type of surveillance region. We develop a distributed strategy which will allow for an architecture that is robust to failures and has performance guarantees.

1.1 Problem Description

We consider a network of identical Pan-Tilt-Zoom (PTZ) cameras for video surveillance, and we focus on the development of distributed and autonomous surveillance strategies for the detection of moving intruders. We make combined assumptions on the environment to be monitored, the cameras, and the intruders. We assume the environment to be one dimensional, in the sense that it can be completely observed by a chain of cameras by using linear motion only (the perimeter surveillance problem is a special case of this framework). We assume the cameras to be subject to physical constraints, such as limited field of view (f.o.v.) and speed, and to be equipped with a low-level routine to detect intruders that fall within their f.o.v.. We

assume intruders to be *smart*, in the sense that they have access to the cameras configuration at every time instant, and schedule their trajectory to avoid detection for as long as possible. Since the probability of success of an intrusion increases with the time an intruder remains undetected in the environment [6], we propose cameras trajectories and control algorithms to minimize the *worst case detection time* and the *average detection time* of smart intruders.

1.2 Related Work

Of relevance to this work are the research areas of robotic patrolling and video surveillance. In a typical robotic patrolling setup, the environment is represented by a graph on which the agents motion is constrained, and the patrolling performance is given by the *worst-case* detection time of *static* events. In [7, 8, 9] performance evaluations of certain patrolling heuristics are performed. In [4] and [2], an efficient and distributed solution to the (worst-case) perimeter patrolling problem for robots with zero communication range is proposed. In [10] the computational complexity of the patrolling problem is studied as a function of the environment topology, and optimal strategies as well as constant-factor approximations are proposed. With respect to these works, we consider smart intruders, as opposed to static ones, and we study also the average detection time, as opposed to the worst case detection time only. In the context of camera networks the perimeter patrolling problem is discussed in [11, 12], where distributed algorithms are proposed for the cameras to partition a one-dimensional environment and to coordinate along a trajectory with minimum worst-case detection time of static intruders. Graph partitioning and intruder detec-

tion with minimum worst-case detection time for two-dimensional camera networks are studied in [13]. We improve the results along this direction by showing that the strategies proposed in [11, 12] generally fail at detecting smart intruders, and by studying the average detection time of smart intruders. Complementary approaches based on numerical analysis and game theory for the surveillance of two dimensional environments are discussed in [14, 15]. Finally, preliminary versions of this work were presented in [16, 17], and a journal version of this work is presented in [18].

In the context of video surveillance most approaches consider the case of static cameras, where the surveillance problem reduces to an optimal sensor placement problem. In [19, 20], sensor placement problems are considered in a static camera network with the goals of maximizing the observability of a set of aggregate tasks that occur in a dynamic environment, and of visual tagging, respectively. In [21], a resource aware scheme for coverage and task assignment is considered. In the case of surveillance in active (PTZ) camera networks, there have been many attempts to formulate feasible approaches for camera control in order to detect and track targets, adapt sensor coverage, and achieve high image resolution; see for instance [22]. In [23], the authors consider an image-based control scheme in a setup containing a "master" camera for detection and tracking. In [24, 25], methodologies for obtaining high-resolution images in camera networks containing both static and active cameras are developed and tested in a virtual environment. In [26] a similar problem of opportunistic visual sensing is considered. In [27], the problem of coordinating camera motion is addressed using a game-theoretic approach with the assumption that the entire environment is covered at all times. Finally, the problem of context-aware

anomaly detection is studied in [28]. We depart from the aforementioned works with PTZ camera networks in the following ways. First, we focus on the problem of intruder detection, rather than tracking or scene analysis, and we define appropriate performance metrics for this problem. Second, we do not make the assumption that cameras can cover the whole environment at all times, and we develop coordination methods for the cameras to surveil the environment. Third and finally, we do not require a source of global information for our algorithms, so that our methods are fully distributed.

1.3 Contributions

The contributions of this work are as follows.

First, we mathematically formalize the concepts of cameras trajectory and smart intruder, and we propose the trajectory design problem for video surveillance. We formalize the worst-case detection time and average detection time criteria, and we characterize lower bounds on both performance criteria.

Second, we propose the *Equal-waiting* cameras trajectory, which achieves minimum worst case detection time, and constant factor optimal average detection time (under reasonable assumptions). The Equal-waiting trajectory is easy to compute given a camera network, and it is amenable to distributed implementation. In fact, we develop a distributed coordination algorithm to steer the cameras along an equal-waiting trajectory. Our coordination algorithm converges in finite time, which we characterize, and it requires only local communication and minimal information to be implemented.

Third, we design a distributed reconfiguration algorithm for the cameras to react to failures and to adapt to time-varying topologies. In particular, our reconfiguration algorithm takes advantage of gossip communication to continuously partition the environment and, at the same time, coordinate the motion of the cameras to optimize the detection performance.

Fourth, we illustrate our findings through simulations and experiments. Our simulations verify our results for different network configurations. Our experiments validate our modeling framework and assumptions, and show that our methods are robust to cameras failure, model uncertainties, and sensor noise.

We finally note that our algorithms are applicable beyond the domain of camera networks. For instance, we envision applicability to real-time scheduling for manufacturing, where tasks with spatial and temporal constraints are allocated, and robots need to complete these tasks while satisfying the given constraints [29, 30, 31].

1.4 Organization

The organization of the remainder of this thesis is as follows. In Section 2, we formalize the problem by introducing our performance metrics, the *worst case* and *average detection times*. In Section 3, we propose the *Equal-Waiting Trajectory*, and provide bounds on its performance. In this section, we also provide a distributed algorithm to steer the cameras into the Equal Waiting trajectory, and provide a proof of its convergence. Section 4 provides the proof of the performance bounds introduced in the previous section. In Section 5, we provide the results of our simulations and hardware experiments. Section 6 provides a distributed algorithm for cameras re-

configuration, which can be used to automatically adjust camera patrolling windows for optimumal performance. Section 7 provides a brief summary and some ideas for future research. Finally, the actual code used for hardware implementations is provided in the Appendix.

2 Problem Formulation

In this section we describe the one-dimensional surveillance problem under consideration, and we present some useful definitions and mathematical tools for its analysis.

2.1 Problem setting and notation

Consider a set of $n \in \mathbb{N}$ identical active cameras installed along a one dimensional open path (boundary) Γ of length L (see Fig 1). For ease of notation and without affecting generality, we represent Γ with the segment $[0, L]$, and we label the cameras in increasing order from c_1 to c_n according to their physical position on Γ. We make the following assumptions:

(A1) the f.o.v. of each camera is represented by a point on Γ,

(A2) the speed v_i of the i-th camera satisfies $|v_i| \in \{0, v_i^{\max}\}$, with $v_i^{\max} \in \mathbb{R}_{>0}$.

For assumption (A2) to be satisfied, we let each camera be equipped with a low-level controller that maintains the speed of its f.o.v. at 0 or $v_i^{\max}$.[1] Let $v^{\max} = \max\{v_1^{\max}, \dots, v_n^{\max}\}$.

Let $x_i : \mathbb{R}_{\geq 0} \to \Gamma$ be a map, such that $x_i(t)$ specifies the position on Γ of the i-th f.o.v. at time t. We define the *patrolling window* $A_i = [\ell_i, r_i] \subseteq \Gamma$ of camera c_i as the segment of Γ containing the f.o.v. of camera c_i at all times, where ℓ_i and r_i denote the start and end points of the segment A_i, respectively. We assume the patrolling windows to be given and constant in time (except for our analysis in Section 6).

[1] For instance, in order to move the camera f.o.v. along its panning direction, the controller may set the panning velocity of the i-th camera to $\dot{\alpha}_i = v_i^{\max}/(a_i \sec^2(\alpha))$, where α denotes the panning angle, and a_i is the distance of the i-th camera from Γ. See Section 5.2 for a related example.

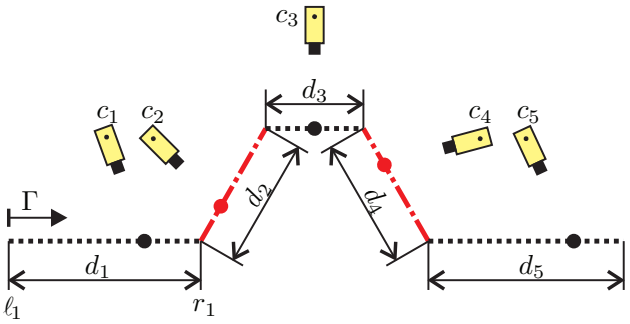


Figure 1: This figure shows five cameras installed along a one dimensional open path. The field of view of each cameras is a point on the path. Cameras coordinate their motion to detect smart moving intruders along the path.

We additionally assume that $\ell_1 = 0$, $r_n = L$, and $\ell_i = r_{i-1}$, with $i = 2, \ldots, n$, so that $\{A_1, \ldots, A_n\}$ is in fact a partition of Γ.[2] Finally, let d_i be the length of A_i, let $d^{\max} = \max\{d_1, \ldots, d_n\}$ and $d^{\min} = \min\{d_1, \ldots, d_n\}$, and define the longest cameras *sweeping time* as $\tau^{\max} = \max\{\tau_1, \ldots, \tau_n\}$, where $\tau_i = d_i / v_i^{\max}$ is the sweeping time of camera c_i.

A *cameras trajectory* is an array $X = \{x_1, \ldots, x_n\}$ of n continuous functions describing the motions of the cameras f.o.v. on Γ. We focus on *periodic* cameras trajectories, for which there exists a duration $T \in \mathbb{R}_{\geq 0}$ such that $X(t+T) = X(t)$ or, equivalently, $x_i(t+T) = x_i(t)$ for all $i \in \{1, \ldots, n\}$. We say that a cameras trajectory is *synchronized* if there exists a time $t_i \in [0, T]$ such that $x_i(t_i) = r_i = l_{i+1} = x_{i+1}(t_i)$ for each pair of neighboring cameras c_i and c_{i+1}.

Remark 1. ***(Two dimensional environment and f.o.v.)*** *Our assumptions of one dimensional environment and point-wise f.o.v. do not limit applicability of our*

[2] As discussed below, this assumption ensures detectability of intruders (see equation (4)).

results to practical cases. To see this, assume that the f.o.v.s of the cameras are two dimensional surfaces on the ground plane. Assume that cameras sweep the environment, and that f.o.v.s of neighboring cameras intersect at some locations. Partition the environment into patrolling windows, in a way that each camera can entirely sweep its assigned region. Let τ_i be the time needed by camera c_i to sweep its region, and let $d_i = \tau_i v_i$, where v_i is the speed of camera c_i. It is now clear that (i) synchronization of the cameras depend only upon τ_i, v_i, and d_i, as captured by our simplified framework, (ii) our synchronization algorithm is applicable for the realistic case of cameras with two dimensional f.o.v.s, since it only requires the parameters τ_i, d_i, and v_i to be implemented, and (iii) the detection performance obtained on our simplified camera model is a conservative bound for the performance with cameras with two dimensional f.o.v.s, because at each time a camera with point-wise f.o.v. can only detect intruders contained in a region of zero area. □

2.2 Model of intruder and performance functions

We consider the problem of detecting intruders appearing at random times and moving on Γ. We model an intruder as an arbitrarily fast point on Γ, and we let the continuous map $\mathcal{I}_{t_0,p_0} : \mathbb{R}_{\geq t_0} \to \Gamma$ be defined such that $\mathcal{I}_{t_0,p_0}(t)$ describes the position of the intruder at a time t, where t_0 and $p_0 = \mathcal{I}_{t_0,p_0}(t_0)$ are the time and location at which the intruder appears, respectively. We focus on *smart* intruders, which have full knowledge of the cameras trajectory and select their motion to avoid detection for as long as possible. More formally, given an initial time $t_0 \in \mathbb{R}_{\geq 0}$, an initial point

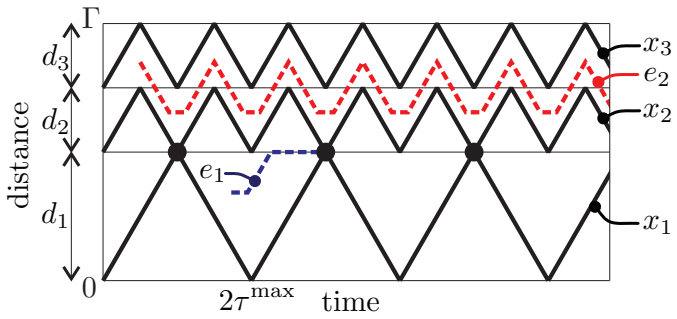


Figure 2: This figure shows a $2\tau^{\max}$-periodic cameras trajectory in which cameras c_1 and c_2 are synchronized ($x_1(t) = x_2(t)$ for $t = k\tau^{\max}$ with $k \in \mathbb{N}_{>0}$; see Section 2.1), while cameras c_2 and c_3 are not synchronized. Notice that, because of the synchronization among cameras, intruder e_1, and in fact any smart intruder appearing between cameras c_1 and c_2, is detected at time $k\tau^{\max}$, for some $k \in \mathbb{N}_{>0}$. Consequently, the worst-case detection time for intruders appearing between cameras c_1 and c_2 is $2\tau^{\max}$. Intruder e_2, and in fact any smart intruder appearing between cameras c_2 and c_3, may avoid detection by properly choosing its trajectory.

$p_0 \in \Gamma$, and a cameras trajectory X, the trajectory $\mathcal{I}^*_{t_0,p_0}$ of a smart intruder satisfies

$$T_{\text{det}}(\mathcal{I}^*_{t_0,p_0}, X) = \max_{\mathcal{I}_{t_0,p_0}} T_{\text{det}}(\mathcal{I}_{t_0,p_0}, X) - t_0,$$

where $T_{\text{det}}(\mathcal{I}_{t_0,p_0}, X)$ is the time at which the intruder is detected by the cameras, that is,

$$T_{\text{det}}(\mathcal{I}_{t_0,p_0}, X) = \min\{\{t \ : \ \mathcal{I}_{t_0,p_0}(t) \in X(t)\}, \infty\}.$$

Notice that the trajectory $\mathcal{I}^*_{t_0,p_0}$ is in general not unique.

We consider two criteria for the detection performance of a T-periodic cameras trajectory, namely the *worst-case detection time* (WDT), and the *average detection*

time (ADT). These two criteria are formally defined as

$$\mathrm{WDT}(X) = \sup_{t_0,p_0} \; T_{\mathrm{det}}(\mathcal{I}^*_{t_0,p_0}, X) - t_0, \tag{1}$$

and

$$\mathrm{ADT}(X) = \frac{1}{TL} \int_0^T \int_\Gamma T_{\mathrm{det}}(\mathcal{I}^*_{t,p}, X) - t \; dpdt. \tag{2}$$

In other words, the WDT criterion measures the longest time that a smart intruder may remain in the environment without detection, while the ADT criterion measures the average time that a smart intruder may remain in the environment without detection, over the boundary Γ and the periodicity T.

The worst case detection time criterion for static intruders, namely WDTs, is defined in [32] as

$$\mathrm{WDTs}(X) = \sup_{t_0,p_0} \{t - t_0 \; : \; t \geq t_0, p_0 \in X(t)\}, \tag{3}$$

and it corresponds to the longest time for the cameras to detect a static intruder, or simply event, along Γ. We next informally discuss the relation between WDT and WDTs, and we refer the reader to [10, 11, 32] for a proof of these results. Let

$$\mathrm{WDT}^* = \min_X \; \mathrm{WDT}(X), \text{ and } \mathrm{WDTs}^* = \min_X \; \mathrm{WDTs}(X).$$

Clearly, $\mathrm{WDT}(X) \geq \mathrm{WDTs}(X)$ for every cameras trajectory X, as static intruders do not move to avoid camera f.o.v.s. For instance, as shown by the example in Fig.

2, if a cameras trajectory X is not synchronized but covers every location of Γ, then $\mathrm{WDTs}(X) < \infty$ and $\mathrm{WDT}(X) = \infty$. Additionally, because the patrolling windows define a partition of Γ, it can be easily verified that the static worst case detection time satisfies

$$\mathrm{WDTs}^* = 2\tau^{\max},$$

and that any $2\tau^{\max}$-periodic cameras trajectory achieves minimum static worst-case detection time. Similarly, any synchronized $2\tau^{\max}$-periodic cameras trajectory X satisfies (see Fig. 2)

$$\mathrm{WDT}(X) = 2\tau^{\max}. \tag{4}$$

In fact, since the f.o.v.s of neighboring cameras intersect at least once in any interval of length $2\tau^{\max}$, intruders cannot avoid detection for more than $2\tau^{\max}$. Thus, any synchronized $2\tau^{\max}$-periodic cameras trajectory achieves minimum worst-case detection time (WDT and WDTs). This discussion motivates us to restrict our attention to periodic and synchronized cameras trajectories.

Problem 1. ***(Design of cameras trajectories)*** *Consider an open path partitioned among a set of n cameras, and let $\tau^{\max}$ be the longest cameras sweeping time. Design a cameras trajectory X^* satisfying*

$$\mathrm{ADT}(X^*) = \mathrm{ADT}^* = \min_{X \in \Omega} \mathrm{ADT}(X),$$

where Ω is the set of all synchronized $2\tau^{\max}$-periodic cameras trajectories.

Remark 2. ***(Optimal patrolling windows)*** *We assume that the patrolling windows are given and form a partition of the path Γ. With these assumptions, the worst-case detection time satisfies $\mathrm{WDT}^* \geq \mathrm{WDTs}^* \geq 2\tau^{\max}$, and any synchronized $2\tau^{\max}$-periodic cameras trajectory achieves the lower bound (see the above discussion).*

If the patrolling windows are not given but are still required to be a partition of Γ, then the longest cameras sweeping time, and hence the worst-case detection performance, can be minimized by solving a min-max graph partitioning problem [10, 32, 17]. We will discuss this aspect in Section 6, where we develop an algorithm to simultaneously partition the environment and coordinate the motion of the cameras to optimize the detection of intruders.

If the patrolling windows are not required to be a partition of Γ, then the bound WDTs^* may be smaller than $2\tau^*$. We refer the reader interested in this case to [6, Conjecture 1] and [33]. □

A second focus of this paper is the design of distributed algorithms to coordinate the cameras along a desired trajectory. We consider a distributed scenario in which cameras c_i and c_j are allowed to communicate at time t only if $|j-i| = 1$ (neighboring cameras) and $x_i(t) = x_j(t)$. Although conservative, this assumption allows us to design algorithms implementable with many low-cost communication devices; see Section 5.2. Notice that additional communications cannot decrease the performance of our algorithms.

Problem 2. ***(Distributed coordination)*** *For a set of n cameras on a one-dimensional*

open path, design a distributed algorithm to coordinate the cameras along a trajectory with minimum average detection time of smart intruders.

3 Equal-Waiting Cameras Trajectory and Coordination Algorithm

In this section we present our results for Problems 1 and 2. In particular, we propose a cameras trajectory with performance guarantees for the average detection time, and a distributed algorithm for the cameras to converge to such a trajectory. We remark that, in general, cameras trajectories with minimum average detection time can be designed via standard, yet computationally intensive, optimization techniques. Such an approach is adopted for instance in [34], where the problem of designing robots' strategies is cast into an optimal control framework and a gradient-based algorithm is used to compute a locally optimal solution, and in [35, Chapter 7], where optimal cameras trajectories are explicitly computed for environments satisfying $d^{\max} < 2d^{\min}$ and for cameras moving at unit speed. The approach taken in this Section is different, as our objective is to gain a comprehensive insight into the cameras surveillance problem, and to design surveillance strategies that are easily implementable and reconfigurable.

3.1 Equal-waiting trajectory

The cameras trajectory that we propose can informally be described as follows:

(Equal-waiting trajectory) Each camera continuously sweeps its patrolling window at maximum speed, and it stops for some waiting time when its f.o.v. reaches an extreme of its patrolling window. For each camera, the waiting times at its two extremes are equal to each other. Additionally, the waiting times of each camera are

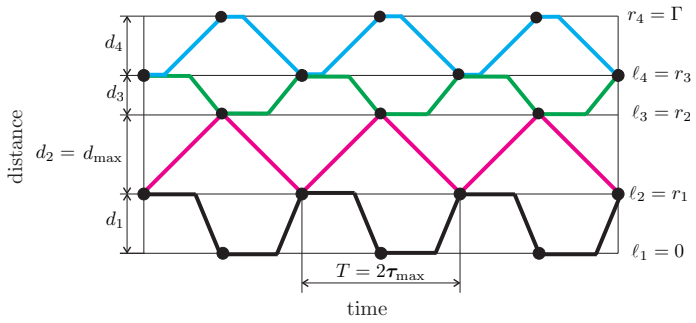


Figure 3: This figure shows the Equal-waiting trajectory for 4 cameras. Notice that (i) the cameras are synchronized, (ii) the trajectory is $2\tau_{\max}$-periodic, and (iii) the waiting time of each camera is the same at both its boundaries.

chosen so that the resulting cameras trajectory is synchronized and periodic. See Fig. 3 for an illustrative explanation. □

Because we let each camera wait for the same amount of time at the two extremes of its patrolling window, we call this cameras trajectory *Equal-waiting trajectory*. An example of Equal-waiting trajectory is in Fig. 3, and a formal description is in Trajectory 1.

As discussed in Section 2, the Equal-waiting cameras trajectory is optimal with respect to the worst-case detection time criterion. Indeed, the Equal-waiting cameras trajectory is synchronized and $2\tau^{\max}$-periodic. We now characterize the average detection time performance of the Equal-waiting trajectory. A proof of this result is postponed to the Section 4.

Theorem 3.1. ***(Performance of Equal-waiting trajectories)*** *Let X^{eq} be the Equal-waiting trajectory defined in Trajectory 1. Then,*

Trajectory 1: *Equal-waiting trajectory (camera c_i)*

Input : $\tau^{\max}, r_i, \ell_i, \tau_i, v_i^{\max}$;
Set : $\omega_i(k) = (k+1)\tau^{\max} - \tau_i$, $k = -1, 0, \ldots$;

if *i is odd* **then**

$$
\begin{aligned}
&x_i(t) = r_i && \text{for } \omega_i(k-1) + \tau_i \le t \le \omega_i(k) \\
&x_i(t) = -v_i^{\max}(t - \omega_i(k)) + r_i && \text{for } \omega_i(k) \le t \le \omega_i(k) + \tau_i \\
&x_i(t) = \ell_i && \text{for } \omega_i(k) + \tau_i \le t \le \omega_i(k+1) \\
&x_i(t) = v_i^{\max}(t - \omega_i(k+1)) + \ell_i && \text{for } \omega_i(k+1) \le t \le \omega_i(k+1) + \tau_i
\end{aligned}
$$

else if *i is even* **then**

$$
\begin{aligned}
&x_i(t) = \ell_i && \text{for } \omega_i(k-1) + \tau_i \le t \le \omega_i(k) \\
&x_i(t) = v_i^{\max}(t - \omega_i(k+1)) + \ell_i && \text{for } \omega_i(k) \le t \le \omega_i(k) + \tau_i \\
&x_i(t) = r_i && \text{for } \omega_i(k) + \tau_i \le t \le \omega_i(k+1) \\
&x_i(t) = -v_i^{\max}(t - \omega_i(k)) + r_i && \text{for } \omega_i(k+1) \le t \le \omega_i(k+1) + \tau_i
\end{aligned}
$$

1. *The average detection time of a smart intruder satisfies the lower bound:*

$$\mathrm{ADT}^* \ge \frac{1}{L} \sum_{i=1}^{n} v_i^{\max} \tau_i^2, \tag{5}$$

2. *The Equal-waiting trajectory X^{eq} satisfies*

$$\mathrm{ADT}(X^{\mathrm{eq}}) = \frac{1}{2} \left(\tau^{\max} + \frac{1}{L} \sum_{i=1}^{n} v_i^{\max} \tau_i^2 \right), \tag{6}$$

3. *The Equal-waiting trajectory X^{eq} satisfies*

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} \le \min \left\{ \frac{\tau^{\max} + \tau^{\min}}{2\tau^{\min}}, \frac{(n+1)d^{\max}}{2d^{\min}} \right\}, \tag{7}$$

4. If $v_i^{\max} = 1$ *for all* $i \in \{1, \ldots, n\}$*, then the Equal-waiting trajectory* X^{eq} *satisfies*

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} \leq \min\left\{\frac{d^{\max} + d^{\min}}{2d^{\min}}, \frac{3+\sqrt{n}}{4}\right\}. \tag{8}$$

The following comments are in order. First, the average detection time of the Equal-waiting trajectory is within a constant factor of the optimal if either $\tau^{\max}/\tau^{\min}$ or n are constant. Second, if all patrolling windows have the same sweeping time, that is $\tau^{\max} = \tau^{\min}$, then our Equal-waiting trajectory is an optimal solution to Problem 1 (it achieves minimum worst-case and average detection times). Moreover, our lower bound (5) is tight and holds with equality if $\tau^{\max} = \tau^{\min}$. Third, the lower bounds in Theorem 3.1 are independent of the ordering of the patrolling windows. Fourth, if

1. all cameras move at unit speed,
2. there exists an index $h \in \{1, \ldots, n\}$ such that $d_h > d_i$ for all $i \in \{1, \ldots, n\} \backslash \{h\}$, and
3. for all $i \in \{1, \ldots, n\} \setminus \{h\}$ the patrolling windows satisfy

$$d_i = \frac{d^{\max}}{1+\sqrt{n}}, \tag{9}$$

then (see the proof of Theorem 3.1 and Fig. 4(b))

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} = \frac{3+\sqrt{n}}{4}. \tag{10}$$

Fifth and finally, different cameras speeds can be taken into account in our bound (8). In fact, if $v_i^{\max}/v_j^{\max} \leq C$ for all $i, j \in \{1, \ldots, n\}$ and for some $C \in \mathbb{R}$, then (see the proof of Theorem 3.1)

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} \leq \min\left\{\frac{C\left(d^{\max}+d^{\min}\right)}{2d^{\min}}, \frac{2+C\left(1+\sqrt{n}\right)}{4}\right\}.$$

3.2 Distributed Algorithm for Camera Coordination

We now design a distributed feedback algorithm that steers the cameras towards an Equal-waiting trajectory. Our algorithm to coordinate the cameras along an Equal-waiting trajectory is described in Algorithm 2, where for convenience we set $x_0(t) = \ell_1$ and $x_{n+1}(t) = r_n$ at all times.

Algorithm 2: *Distributed camera coordination along an Equal-waiting trajectory (camera c_i)*

Input : $\tau^{\max}, r_i, \ell_i, \tau_i, v_i^{\max}$;
Set : $\omega_i = \tau^{\max} - \tau_i$, $x_0(t) = \ell_1$ and $x_{n+1}(t) = r_n$ $\forall t$;
1 Move towards ℓ_i with $|v_i(t)| = v_i^{\max}$;
while *True* **do**
2 **if** $x_i(t) = x_{i-1}(t)$ *or* $x_i(t) = x_{i+1}(t)$ **then**
3 Wait until time $t + \omega_i$;
4 Move towards the opposite boundary with $|v_i(t)| = v_i^{\max}$;

An informal description of Algorithm 2 follows.

(Distributed coordination) Camera c_i moves to ℓ_i (line 1) and, if $i > 1$, it waits until the f.o.v. of its left neighboring camera c_{i-1} occupies the same position (line 2). Then, camera c_i stops as specified in Trajectory 1 (line 3), and finally move to r_i

(line 4). Camera c_1 (resp. c_n) moves to r_2 (resp. ℓ_{n-1}) as soon as its f.o.v. arrives at ℓ_1 (resp. r_n). □

It should be observed that, by construction, cameras sweep their patrolling windows (lines 1 and 4), and that the cameras trajectory obtained via Algorithm 2 is synchronized and Equal-waiting. Moreover, since cameras wait until the f.o.v. of a neighboring camera occupies the same position (line 3), our coordination algorithm is robust to cameras failures and motion uncertainties. A related example is in Section 5. Regarding the implementation of Algorithm 2, notice that each camera is required to know the endpoints of its patrolling window, its sweeping time and the maximum sweeping time in the network, and to be able to communicate with neighboring cameras. The following theorem characterizes the convergence properties of Algorithm 2, where we write $X(t \geq \bar{t})$ to denote the restriction of the trajectory $X(t)$ to the interval $t \in [\bar{t}, \infty)$.

Theorem 3.2. ***(Convergence of Algorithm 2)*** *For a set of n cameras with sweeping times $\tau_1, \ldots, \tau_n$, let $X(t)$ be the cameras trajectory generated by Algorithm 2. Then, $X(t \geq n\tau^{\max})$ is an Equal-waiting trajectory.*

Proof. Notice that the f.o.v. of camera c_1 coincides with the f.o.v. of camera c_2 within time $\max\{2\tau_1, \tau_2\} \leq 2\tau^{\max}$. Then, the f.o.v. of camera c_i coincides with the f.o.v. of camera c_{i+1} within time $(i+1)\tau^{\max}$. Hence, within time $n\tau^{\max}$ the cameras trajectory coincides with an Equal-waiting trajectory in Trajectory 1. The claimed statement follows. □

Notice that our cameras trajectory and coordination algorithm are easy to com-

pute, valid for every number of cameras and environment configuration, and their performance are guaranteed to be within a bound of the optimum.

4 Proof of Main Theorem

This section contains a proof of Theorem 3.1. We start with a lower bound for the average detection time.

Lemma 4.1. ***(Lower bound on the average detection time)*** *For a set of n cameras with maximum velocities $v_1^{\max}, \ldots, v_n^{\max}$ and sweeping times $\tau_1, \ldots, \tau_n$, the average detection time of a $2\tau^{\max}$-periodic cameras trajectory X satisfies the lower bound*

$$\mathrm{ADT}(X) \geq \frac{1}{L} \sum_{i=1}^{n} v_i^{\max} \tau_i^2.$$

Proof. Since a smart intruder moves away from the camera f.o.v., the detection time of a smart intruder appearing at time t and at location $p \in [\ell_i, r_i]$ satisfies the lower bound

$$\int_{\ell_i}^{r_i} T_{\mathrm{det}}(\mathcal{I}_{t,p}) - t \; dp = \int_{0}^{d_i} T_{\mathrm{det}}(\mathcal{I}_{t,p}) - t \; dp \geq T_{\mathrm{up}},$$

where $d_i = r_i - \ell_i$ and T_{up} equals

$$\int_{0}^{x_i(t)} \frac{x_i(t) + 2(d_i - x_i(t))}{v_i^{\max}} \; dp + \int_{x_i(t)}^{d_i} \frac{d_i - x_i(t)}{v_i^{\max}} \; dp,$$

if the camera c_i first detects intruders appearing at locations $p \geq x_i(t)$. Analogously,

if the camera c_i first detects intruders appearing at locations $p \leq x_i(t)$

$$\int_0^{d_i} T_{\text{det}}(\mathcal{I}_{t,p}) - t \; dp \geq T_{\text{down}},$$

where T_{down} equals

$$\int_0^{x_i(t)} \frac{x_i(t)}{v_i^{\max}} \; dp + \int_{x_i(t)}^{d_i} \frac{2x_i(t) + (d_i - x_i(t))}{v_i^{\max}} \; dp.$$

To see this, consider the first case. The detection time of every intruder appearing at location $p > x_i(t)$ is at least $(d_i - x_i(t))/v^{\max}$ (time needed by camera c_i to reach r_i starting from $x_i(t)$). Likewise, the detection time of every intruder appearing at location $p < x_i(t)$ is at least $(x_i(t) + 2(d_i - x_i(t)))/v^{\max}$ (time needed by c_i to reach r_i and ℓ_i starting from $x_i(t)$). The other case follows similarly.

It can be verified by simple manipulation that $T_{\text{up}} = T_{\text{down}} = d_i^2/v^{\max}$. Finally, it follows from (2) and $\tau_i = d_i/v_i^{\max}$ that

$$\text{ADT}(X) = \frac{1}{TL} \int_0^T \int_\Gamma T_{\text{det}}(\mathcal{I}^*_{t,p}) - t \; dpdt \geq \frac{1}{TL} \int_0^T \sum_{i=1}^n \frac{d_i^2}{v_i^{\max}} \; dt = \frac{1}{L} \sum_{i=1}^n v_i^{\max} \tau_i^2.$$

□

It should be observed that the bound in Lemma 4.1 is tight for the case of a single camera, and it is conservative otherwise (see Fig. 4(a) and 5). We now characterize the average detection time of the Equal-waiting trajectory.

Lemma 4.2. ***(Equal-waiting trajectory performance)*** *For a set of n cameras with sweeping times $\tau_1, \ldots, \tau_n$, let X^{eq} be the Equal-waiting trajectory defined in*

Trajectory 1. Then

$$\mathrm{ADT}(X^{\mathrm{eq}}) = \frac{1}{2}\left(\tau^{\max} + \frac{1}{L}\sum_{i=1}^{n} v_i^{\max}\tau_i^2\right). \tag{11}$$

Proof. Observe that the function $\mathrm{ADT}(X^{\mathrm{eq}})$ can be written as

$$\mathrm{ADT}(X^{\mathrm{eq}}) = \frac{1}{2\tau^{\max}L}\sum_{i=1}^{n}\int_0^{2\tau^{\max}}\int_{\ell_i}^{r_i}(T_{\mathrm{det}}(\mathcal{I}_{t,p}) - t)\ dpdt.$$

Let i be odd and recall the description of $x_i(t)$ given in Trajectory 1. Due to symmetry, it can be verified that

$$\int_0^{2\tau^{\max}}\int_{\ell_i}^{r_i}(T_{\mathrm{det}}(\mathcal{I}_{t,p}) - t)\ dpdt = 2\int_0^{\tau^{\max}}\int_{\ell_i}^{r_i}(T_{\mathrm{det}}(\mathcal{I}_{t,p}) - t)\ dpdt$$

Let $0 \le t \le \tau^{\max} - \tau_i$, and notice that $x_i(t) = r_i$. Observe that $T_{\mathrm{det}}(\mathcal{I}_{t,p}) = \tau^{\max}$ for all $p \in [\ell_i, r_i)$, and $T_{\mathrm{det}}(\mathcal{I}_{t,p}) = 0$ for $p = r_i$. Then

$$\int_0^{\tau^{\max}-\tau_i}\int_{\ell_i}^{r_i}(T_{\mathrm{det}}(\mathcal{I}_{t,p}) - t)\ dpdt = \frac{(\tau^{\max})^2 - \tau_i^2}{2}\,d_i\,. \tag{12}$$

Let $\tau^{\max} - \tau_i \le t \le \tau^{\max}$. Observe that $T_{\mathrm{det}}(\mathcal{I}_{t,p}) = \tau^{\max}$ for $p \in [\ell_i, x_i(t))$, $T_{\mathrm{det}}(\mathcal{I}_{t,p}) = 0$ for $p = x(t)$, and $T_{\mathrm{det}}(\mathcal{I}_{t,p}) = 2\tau^{\max}$ for $p \in (x_i(t), r_i]$. Thus,

$$\begin{aligned}
&\int_{\tau^{\max}-\tau_i}^{\tau^{\max}}\int_{\ell_i}^{r_i}(T_{\mathrm{det}}(\mathcal{I}_{t,p}) - t)\ dpdt \\
&= \int_{\tau^{\max}-\tau_i}^{\tau^{\max}}\int_{\ell_i}^{x_i(t)}(\tau^{\max} - t)\ dp + \int_{x_i(t)}^{r_i}(2\tau^{\max} - t)\ dpdt.
\end{aligned}$$

Since $x_i = r_i + (t - (\tau^{\max} - \tau_i)v_i^{\max})$ (see Trajectory 1), it follows from the above expression that

$$\int_{\tau^{\max}-\tau_i}^{\tau^{\max}} \int_{\ell_i}^{r_i} (T_{\det}(\mathcal{I}_{t,p}) - t) \; dpdt = \int_{\tau^{\max}-\tau_i}^{\tau^{\max}} (2\tau^{\max} - t)d_i - \tau^{\max}(\tau^{\max} - t)v_i^{\max} \, dt$$
$$= \frac{1}{2}\tau^{\max}\tau_i^2 v_i^{\max} + \frac{1}{2}d_i\tau_i^2. \tag{13}$$

The statement follows by combining (12) and (13). □

We now conclude with a proof of Theorem 3.1.

Proof of Theorem 3.1: From Lemma 4.1 and 4.2 we have

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} = \frac{1}{2} + \frac{L\tau^{\max}}{2\sum_{i=1}^n v_i^{\max}\tau_i^2} = \frac{1}{2} + \frac{L\tau^{\max}}{2\sum_{i=1}^n d_i\tau_i}$$
$$\leq \frac{1}{2} + \frac{L\tau^{\max}}{\tau^{\min}2\sum_{i=1}^n d_i} = \frac{\tau^{\max} + \tau^{\min}}{2\tau^{\min}},$$

where we have used $L = \sum_{i=1}^n d_i$ and $\tau^{\min} \leq \tau_i$ for all $i \in \{1, \ldots, n\}$. To show the second bound notice that

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} = \frac{1}{2} + \frac{L\tau^{\max}}{2\sum_{i=1}^n d_i\tau_i} = \frac{1}{2} + \frac{L}{2\sum_{i=1}^n d_i\frac{\tau_i}{\tau^{\max}}} \leq \frac{1}{2} + \frac{L}{2d^{\min}},$$

where the last inequality is obtained by letting $\tau_i/\tau^{\max} \to 0$ for all i except for one segment ($\tau_i/\tau^{\max} = 1$ for some i, and $d_i \geq d^{\min}$). Since $L \leq nd^{\max}$ we conclude that

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} \leq \frac{1}{2} + \frac{nd^{\max}}{2d^{\min}} \leq \frac{(n+1)d^{\max}}{2d^{\min}}.$$

We now show the last part of the Theorem. Assume that all cameras move at unit speed and, without affecting generality, that $d_1 = d^{\max}$. Notice that

$$\frac{\mathrm{ADT}(X^{\mathrm{eq}})}{\mathrm{ADT}^*} = \frac{1}{2} + \frac{L\tau^{\max}}{2\sum_{i=1}^n d_i\tau_i} = \frac{1}{2} + \frac{\sum_{i=1}^n d_1 d_i}{2\sum_{i=1}^n d_i^2} = \frac{1}{2}\left(1 + \frac{1+\sum_{i=2}^n y_i}{1+\sum_{i=2}^n y_i^2}\right),$$

where $y_i = d_i/d_1$. Consider the minimization problem

$$\begin{array}{ll} \min\limits_{\{K,x_2,\ldots,x_n\}} & K, \\ \text{subject to} & 1+\sum_{i=2}^n y_i \leq K\left(1+\sum_{i=2}^n y_i^2\right), \end{array} \tag{14}$$

and the associated Lagrangian function [36]

$$\mathcal{L} = K + \lambda\left(1 - K + \sum_{i=2}^n y_i - K y_i^2\right).$$

Following standard optimization theory, necessity optimality conditions for the minimization problem (14) are

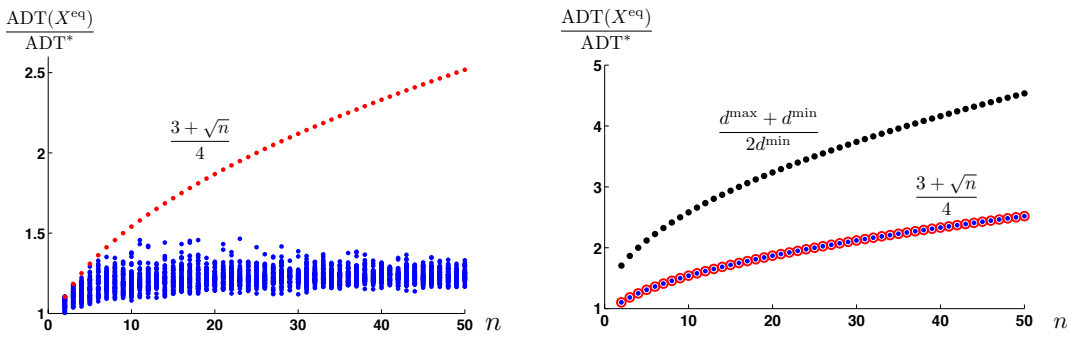
$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial K} = 0 \implies 1 - \lambda\left(1 + \sum_{i=2}^n y_i^2\right) = 0, \\ &\frac{\partial \mathcal{L}}{\partial y_i} = 0 \implies \lambda(1 - 2Ky_i) = 0, \text{ for } i \in \{2,\ldots,n\}, \\ &\text{Complementary slackness: } \lambda\left(1 - K + \sum_{i=2}^n y_i - K y_i^2\right) = 0. \end{aligned}$$

From the first and second equations we obtain $\lambda \neq 0$ and $y_i = 1/(2K)$. Then, the third equation yields $K = (1 \pm \sqrt{n})/2$. Since $y_i > 0$, the statement follows. □

5 Simulations and Experiments

In this section we report the results of our simulation studies and experiments. Besides validating our theory, these results show that our models are accurate, that our algorithms can be implemented on real hardware, and that our algorithms are robust to sensor noise and model uncertainties.

5.1 Simulations



(a) (b)

Figure 4: In Fig. 4(a) we report the ratio $\mathrm{ADT}(X^{\mathrm{eq}})/\mathrm{ADT}^*$ as a function of the number of cameras n (blue dots), and the bound $(3+\sqrt{n})/4$ in Theorem 3.1 (red dots). For the considered configurations, the bound $(d^{\max}+d^{\min})/(2d^{\min})$ is much larger than the experimental data, and it is not considered here. The lengths d_i of the patrolling windows are uniformly distributed in the interval $(0,1]$, with $d_1 = d^{\max} = 1$. We assume that cameras have unit speed. In Fig. 5 we report the ratio $\mathrm{ADT}(X^{\mathrm{eq}})/\mathrm{ADT}^*$ as a function of the number of cameras n (blue dots), and the bounds in Theorem 3.1 (black dots and red circles). The lengths d_i of the patrolling windows are chosen as $d_1 = d^{\max} = 1$ and $d_i = (1+\sqrt{n})^{-1}$ for all $i \in \{2,\dots,n\}$. As predicted by equation (10), the performance bound in equation (8) is tightly achieved.

Three simulation studies are presented in this section. For our first simulation

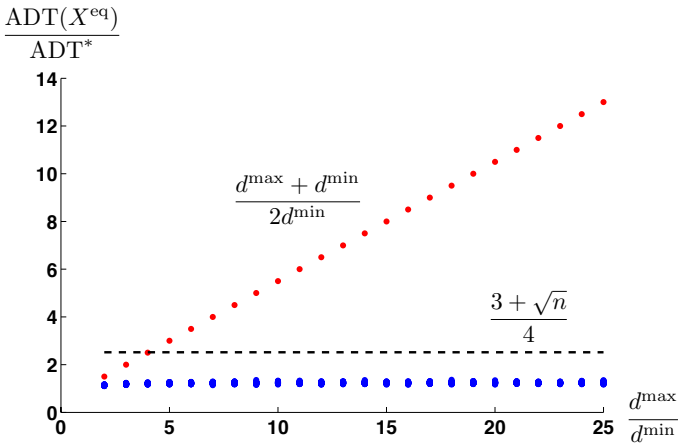


Figure 5: In this figure we report the ratio $\mathrm{ADT}(X^{\mathrm{eq}})/\mathrm{ADT}^*$ as a function of $d^{\max}/d^{\min}$ (blue dots), and the bounds in Theorem 3.1. For each value of $d^{\max}/d^{\min}$, the lengths of the patrolling windows are uniformly distributed in the interval $[d^{\min}/d^{\max}, 1]$, with $d_1 = d^{\max} = 1$. Notice that the theoretical bounds are compatible with the experimental data.

study, we let the number of cameras n vary from 2 to 50. For each value of n, we generate 50 sets of patrolling window with lengths $\{d_1, \ldots, d_n\}$, where $d_1 = d^{\max} = 1$ m, and d_i is uniformly distributed within the interval $(0, 1]$ m , for all $i \in \{2, \ldots, n\}$. For each configuration we let $v_i^{\max} = 1$ m/s for all cameras, we design the Equal-waiting trajectory X^{eq}, and we evaluate the cost $\mathrm{ADT}(X^{\mathrm{eq}})$ and the lower bound ADT^* from equation (5). We report the result of this study in Fig. 4(a). Notice that, when the number of cameras is large and the lengths of the patrolling windows are uniformly distributed, the bound in (8) is conservative. On the other hand, if the lengths of the patrolling windows are chosen as in (9), then the bound in (8) is tightly achieved (Fig. 4(b)).

For our second simulation study, we let the number of cameras be fixed (50 cameras), and we vary the value $d^{\max}/d^{\min}$ between 2 and 25. Specifically, we let

$d_1 = d^{\max} = 1$ m, and d_i, with $i = 2, \ldots, 50$ be uniformly distributed within the interval $[d^{\min}/d^{\max}, 1]$ m. For each value of $d^{\max}/d^{\min}$ we generate 50 sets of patrolling windows with lengths $\{d_1, \ldots, d_n\}$, compute the Equal-waiting trajectory X^{eq}, evaluate the cost $\text{ADT}(X^{\text{eq}})$, and compute the lower bound in equation (5). The results of this simulation study are reported in Fig. 5, where we observe that the theoretical bounds derived in Theorem 5 are compatible with the experimental data.

In our third simulation study we validate the effectiveness of our coordination algorithm. We consider a set of 4 cameras with pre-specified patrolling windows and unit speed. The cameras trajectory generated by Algorithm 2 is reported in Fig. 6. Observe that our coordination algorithm drives the cameras towards an Equal-waiting trajectory, and it is robust to failures and motion uncertainty. In particular, (i) coordination is achieved for cameras starting at random initial positions, (ii) the algorithm is robust to temporary cameras failure, and (iii) the average detection time degrades gracefully in the presence of motion uncertainties.

5.2 Experiments

In this section we detail the experiments we have conducted to validate our theoretical findings and simulation results. In order to perform the experiments, it it is necessary to utilize a variety of tools from computer vision and computer science in order to practically implement the code. We start by introducing these concepts, and then move on to describe the experimental results

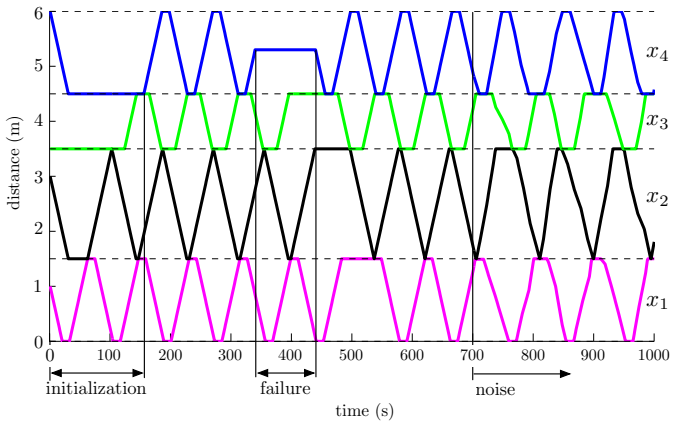


Figure 6: In this figure we validate Algorithm 2 for a set of 4 cameras with unit speed. Cameras start at random positions inside their patrolling window and achieve coordination at time 150. Notice that the algorithm recovers from the temporary failure of camera c_4 between time 340 and 440. Moreover, the coordination performance of the algorithm degrade gracefully in the presence of noise affecting the cameras motion of the cameras (time 700). In this simulation the cameras motion noise is assumed to be normally distributed with mean 0.2 and unit standard deviation.

5.2.1 Programming and Computer Vision Tools

Multi-Threading Since the algorithm that we are trying to implement is distributed, it is necessary to find a way to simulate multiple, independent processes on the same machine. The tool that allows us to do this is called multi-threading.

In multithread programming, we are able to spawn multiple threads from the same master thread. This allows us to more efficiently use the computer's processing power. Each of these threads is completely independent, and has its own set of variables associated to it. For the purposes of the problem at hand, this is useful because we can assign each camera involved in the experiment its own thread, while still being able to control the program and receive information about the state of the system through the master computer.

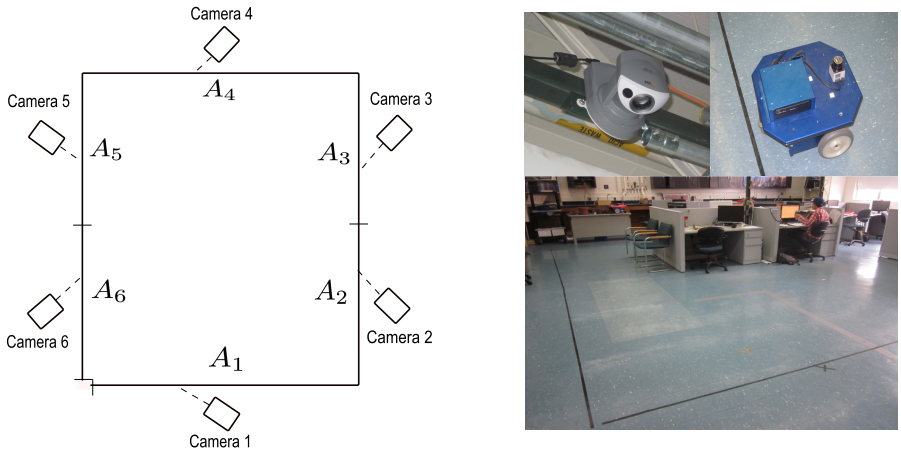


Figure 7: Illustration of the experimental set-up and photos of the hardware.

Object Oriented Programming Object oriented programming is a technique which allows us to create an entity called an object, which can be thought of as a type of data structure. Each object has a variety of values and functions associated to it, and is able to pass and receive information to the other objects. This provides an alternative way of viewing a program, as a collection of interacting objects, rather than a list of tasks.

One instance when this type of programming scheme is useful is in a scenario where it is necessary to run very similar codes multiple times. In our experiments, object oriented programming is very useful, as it allows us to define a data structure (object) called a *camera*, to which we can associate the various values that are crucial to the program, such as urls necessary to send commands to the camera, camera position and timestep data, and current state information.

Queuing Queuing is the mechanism which allows us to pass information between

the multiple threads that are running on each camera. The queuing functionality allows us to assign to each object (camera) a "queue", in which values or commands can be placed. The associated camera can then draw from its queue in order to receive the information it contains. Items that are placed in the queue are ordered, and are drawn out of the queue in a "first in, first out" fashion.

This functionality is extremely useful when it is necessary to control the rate or timing at which commands are sent to an object. In the context of our experiement, we utilize the queues associated to each camera in order to relay information about when a camera has finished sweeping its assigned region. That is, when a camera has reached an extreme point of its assigned partition, it places a message in its neighboring camera's queue and simultaneously checks its own queue to see if it has received a message from the neighboring camera. If a message is there, the camera knows that the neighboring camera is also at the extreme point, and hence it proceeds with the execution of the Equal-waiting trajectory algorithm. If no message is there, the camera knows that the neighboring camera has not finished its sweep, and thus waits while continously checking its queue until a message is received. This method ensures that the trajectory of the cameras stays synchronized and robust to failures.

Histogram Back-Projection

Histogram Back-Projection is an image processing technique in which the degree of similarity between two images can be quantified. From each image, a hue image is generated, in which each pixel in the image is given a numerical value based on its color. From this, computer vision techniques are used to compare the two images and essentially calculate the probability that the two images are showing the same

feature.

For our experiments, we utilize histogram back-projection in the following way: We start with an image of our robot intruder to use as a reference image. Then, for each frame that a camera captures, we utilize histogram back-projection functions in OpenCV in order to calculate the similarity of the frame with the reference image (the image of the intruder). If the similarity score exceeds a given threshold, we know that the current frame is showing an intruder and a flag is raised.

We remark that in a real scenario the user will not know what the intruder will look like, and therefore a histrogram back-projection algorithm will not work. In this case, more advanced computer vision algorithms for object detection must be employed. However, since the focus of our problem is on the control aspects of this problem, a simple histogram back-projection algorithm is sufficient.

5.2.2 Experimental Results

For our experiments we use a network of six AXIS 213 PTZ (Pan-Tilt-Zoom) network cameras mounted along a square perimeter. In order to simulate a 1-dimensional environment, we assign each camera responsibility for surveilling a segment of the perimeter and assume that camera c_1 and camera c_6 have no left and right neighbors, respectively. Movement of the cameras is restricted to a panning motion and is controlled in such a way as to keep the center of the f.o.v. moving at constant speed. Each camera is equipped with a low-level detection algorithm in order to alert the user when an intruder enters its f.o.v.. All programming of the cameras is performed in Python, utilizing the OpenCV computer vision package for image processing. A

diagram of our camera network and a table with our experimental parameters are shown in Fig. 7 and Table 1.

Table 1: Relevant experimental parameters.

Camera	d_i(cm)	$v_i^{\max}$(cm/s)	τ_i (s)
c_1	624.3	20.8	30.0
c_2	290.3	18.0	16.1
c_3	291.0	20.6	14.1
c_4	619.3	21.1	29.0
c_5	331.5	19.0	17.4
c_6	232.7	17.3	13.5

In our first experiment, we validate our distributed coordination algorithm to control the motion of the cameras. Fig. 8 shows the results of our experiment. Notice that the algorithm steers the cameras into an equal waiting trajectory within time $6\tau^{\max}$ as predicted by Theorem 3.2. In fact, since the cameras are all starting the experiment at their left boundary, we see that the system reaches an equal waiting trajectory in only slightly longer than $5\tau^{\max} = 150$ s. This is consistent with Theorem 3.2, since delays in communication and network bandwidth limits cause some lagging in our experimental implementation. In order to demonstrate the behavior of the algorithm under a camera failure, camera c_4 is stopped at time $t = 600$ s. Notice that the algorithm continues to function despite this temporary hardware failure.

In our second experiment we focus on the worst-case detection time of intruders. We utilize an Erratic mobile robot from Videre Design to simulate a *smart* intruder.

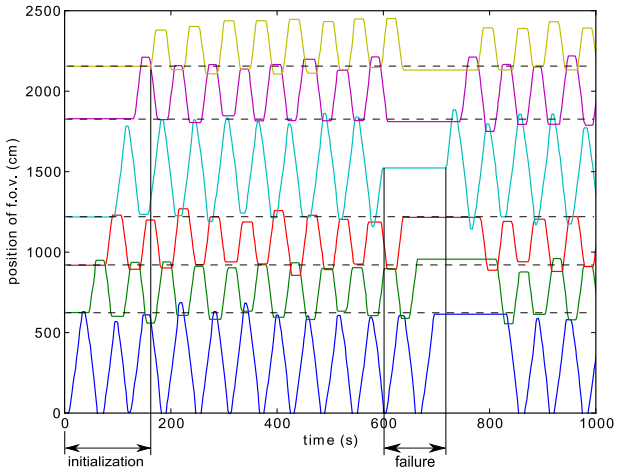


Figure 8: Cameras trajectories as obtained from our experimental implementation of Algorithm 2. See Table 1 for the cameras parameters. Notice that the trajectory is robust to noise, as well as small overshoots and undershoots introduced by hardware and network uncertainty. These inaccuracies in the individual camera trajectories do not significantly affect coordination. The cameras trajectory is also robust to momentary failures, as shown at time $t \approx 600$ s.

The robot is equipped with an on-board computer with Ubuntu Linux and uses Player/Stage in order to interface with the user and allow for manual steering. We assume that the cameras motion is controlled by Algorithm 2, and we run 40 trials where the Erratic robot enters the environment at specific times and locations (we let the Erratic robot move only along the first segment, that is, the segment with longest sweeping time), and it is manually driven to avoid detection for as long as possible. We report the results of our second experiment in Fig. 11(a), where we notice that the theoretical worst-case detection time is a relatively tight bound for the experimental worst-case detection time.

In our third experiment we focus on the average detection time of intruders. As

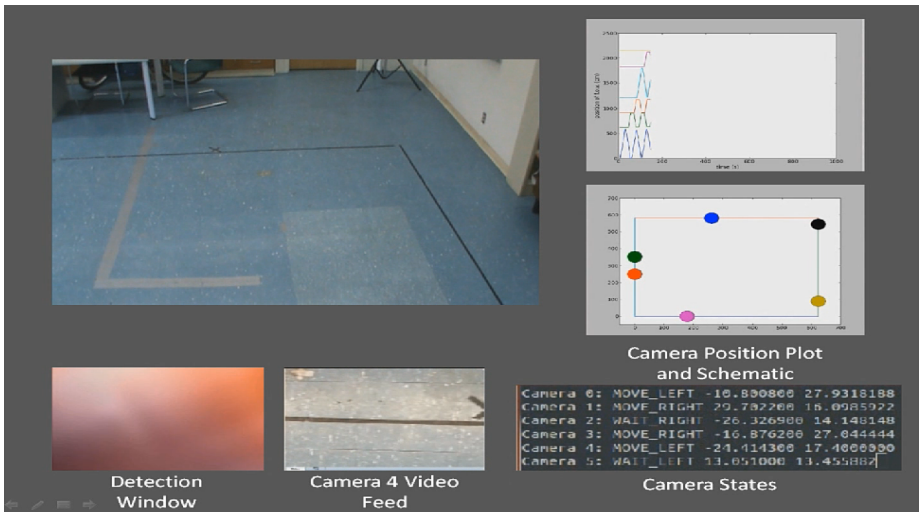


Figure 9: Screen shot of the GUI from our experiments, before an intruder has entered the environment

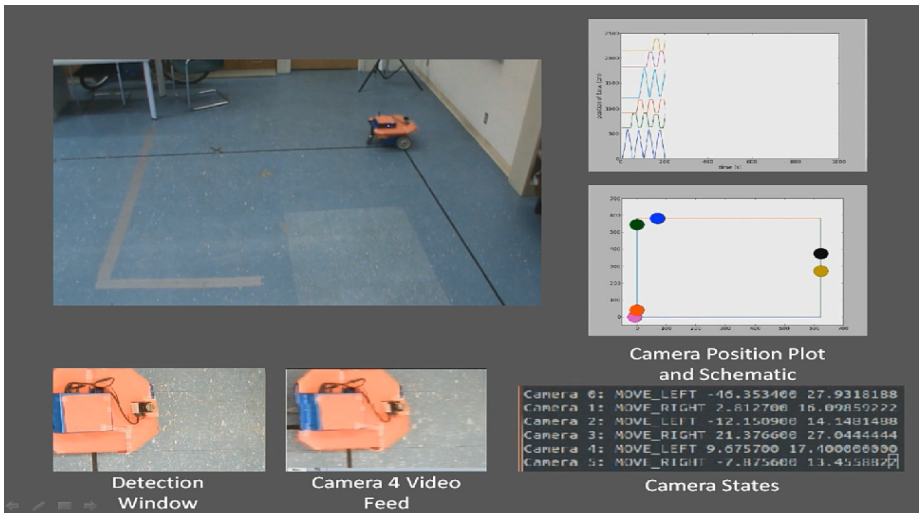
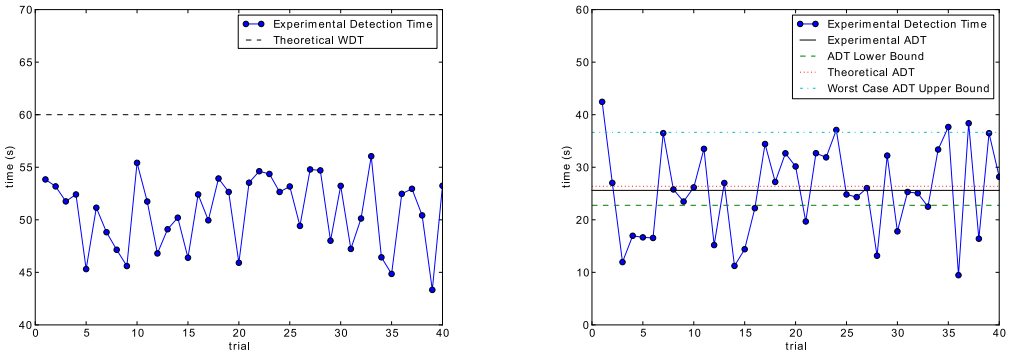


Figure 10: Screen shot of the GUI from our experiments, after an intruder has entered the environment

in our second experiment, we let the cameras motion be controlled by Algorithm 2, and we use an Erratic robot as an intruder. We run 40 trials where the Erratic robot enters the environment at random times and locations, and it is manually driven to avoid detection for as long as possible. We report the results of our second experiment in Fig. 11(b), where we notice that the theoretical bounds in Theorem 3.1 are compatible with the experimental data (the slight difference is due to the fact that the theoretical value is calculated by considering all possible intruder initial locations and times).

We remark that there is a small amount of uncertainty in the execution of the algorithm by the cameras, resulting in small overshoots and undershoots in the individual camera trajectories. As we see from Fig. 8, these small gaps, which are to be expected in practical applications, do not have a significant effect on the performance of the algorithm. We conclude that our experimental results validate our theory, our camera models, and our assumptions.



(a) (b)

Figure 11: In Fig. 11(a) we show the detection times for our second experiment, in which smart intruders appear at *worst case* times and locations. The detection times of this experiment are depicted by a solid blue line. Notice that the detection times are smaller than the upper bound predicted in Section 2. In Fig. 11(b) we show the detection times for each trial of our third experiment, in which smart intruders appear at *random* times and locations. The detection times of this experiment are depicted by a solid blue line. The solid black line corresponds to the average of the experimental detection times. For the considered configuration of cameras, the lower bound ADT* in (5) (dashed green line), ADT(X^{eq}) as in (6) (dotted red line), and the worst case upper bound (dashed light blue line), which is calculated by multiplying the lower bound on ADT* by the quantity $\frac{\tau^{\max}+\tau^{\min}}{2\tau^{\min}}$ from (7) are reported.

6 An Algorithm for Camera Reconfiguration

In this section we describe an algorithm to reconfigure the cameras patrolling windows to improve the detection performance, and to allow the camera network to recover from a permanent camera failure and autonomously adapt to the addition and removal of cameras. We consider a *symmetric gossip* communication protocol among cameras, where communication is allowed only among neighboring cameras, and where each camera updates its patrolling window only after communication with a neighboring camera. Our reconfiguration algorithm (REC) is described in Algorithm 3, where D_i represents cameras visibility constraints ($A_i \subseteq D_i$).

An informal description of Algorithm 3 follows.

(Cameras reconfiguration) Camera c_i sweeps back and forth at maximum speed its patrolling windows A_i (line 1), and it updates A_i upon communication with neighboring cameras (lines 2 to 5). The update of A_i is performed so that, as time progresses, the cameras patrolling windows form a partition of the boundary that minimizes the longest sweeping time (line 5). Following Algorithm 2, cameras stop for a certain waiting time when their f.o.v. reaches an extreme of their patrolling window. These waiting times ensure that (i) communication among neighboring cameras is maintained over time, and (ii) cameras trajectories are synchronized along an Equal-waiting trajectory. As previously mentioned, in an Equal-waiting trajectory the waiting times at the two extremes of the patrolling window are equally long. Finally, in order to achieve motion synchronization, the (time varying) maximum sweeping time $\tau^{\max}$ is propagated across cameras during the execution of the algorithm. In order to do so, the auxiliary variable q_i is used by the i-th camera to store

Algorithm 3: *Cameras Reconfiguration (Camera c_i)*

Input : $A_i = [\ell_i, r_i]$, $D_i = [\underline{\gamma}_i, \overline{\gamma}_i]$, $v_i^{\max}$;
Require : $\{A_1, \ldots, A_n\}$ is a partition of Γ;

Set $\hat{\tau}_i^{\max} = \tau_i$ and $q_i = c_i$;
1 Move according to Algorithm 2;
if *Communication between cameras c_i and c_{i+1}* **then**
2 Transmit ℓ_i, r_i, $\hat{\tau}_i^{\max}$, q_i to c_{i+1};
3 Receive ℓ_{i+1}, r_{i+1}, $\hat{\tau}_{i-1}^{\max}$, q_{i+1} from c_{i+1};
4 Compute $m = \left(\ell_i v_{i+1}^{\max} + r_{i+1} v_i^{\max}\right) / \left(v_i^{\max} + v_{i+1}^{\max}\right)$;
5 Update r_i and ℓ_{i+1} as:

$$r_i = \ell_{i+1} = \begin{cases} m, & \text{if} \quad m \in [\underline{\gamma}_{i+1}, \overline{\gamma}_i], \\ \overline{\gamma}_i, & \text{if} \quad m > \overline{\gamma}_i, \\ \underline{\gamma}_{i+1}, & \text{if} \quad m < \underline{\gamma}_{i+1}; \end{cases}$$

6 Compute new value for $\tau_i = (r_i - \ell_i)/v_i^{\max}$;
7 **case** $q_i \geq c_i$
8 **if** $q_{i+1} > c_{i+1}$ **then**
9 $\hat{\tau}_i^{\max} = \hat{\tau}_{i+1}^{\max} = \max\{\tau_i, \tau_{i+1}, \hat{\tau}_{i+1}^{\max}\}$;
10 $q_i = q_{i+1} = \arg \max_{c_i, c_{i+1}, q_{i+1}} \{\tau_i, \tau_{i+1}, \hat{\tau}_{i+1}^{\max}\}$;
11 **else if** $q_{i+1} \leq c_{i+1}$ **then**
12 $\hat{\tau}_i^{\max} = \hat{\tau}_{i+1}^{\max} = \max\{\tau_i, \tau_{i+1}\}$;
13 $q_i = q_{i+1} = \arg \max_{c_i, c_{i+1}} \{\tau_i, \tau_{i+1}\}$;
14 **case** $q_i < c_i$
15 **if** $q_{i+1} \leq c_{i+1}$ **then**
16 $\hat{\tau}_i^{\max} = \hat{\tau}_{i+1}^{\max} = \max\{\tau_i, \tau_{i+1}, \hat{\tau}_i^{\max}\}$;
17 $q_i = q_{i+1} = \arg \max_{c_i, c_{i+1}, q_i} \{\tau_i, \tau_{i+1}, \hat{\tau}_i^{\max}\}$;
18 **else if** $q_{i+1} > c_{i+1}$ **then**
19 $\hat{\tau}_i^{\max} = \hat{\tau}_{i+1}^{\max} = \max\{\tau_i, \tau_{i+1}, \hat{\tau}_i^{\max}, \hat{\tau}_{i+1}^{\max}\}$;
20 $q_i = q_{i+1} = \arg \max_{c_i, c_{i+1}, q_i, q_{i+1}} \{\tau_i, \tau_{i+1}, \hat{\tau}_i^{\max}, \hat{\tau}_{i+1}^{\max}\}$;

the information about the camera associated with $\tau^{\max}$ (lines 7 to 20). □

For the analysis of Algorithm 3, notice that the patrolling window A_i is updated

every time camera c_i communicates with a neighboring camera. Let $A_i(k)$ denote the i-th patrolling window after k communications of camera c_i, and let $d_i(k)$ be the length of $A_i(k)$. We say that a cameras trajectory X is *asymptotically T-periodic* if there exists a duration $T \in \mathbb{R}_{>0}$ satisfying

$$\lim_{t\to\infty} X(t+T) - X(t) = 0.$$

Theorem 6.1. ***(Convergence of REC)*** *Consider a set of n cameras installed along a one-dimensional open path Γ. Let $A_1, \ldots, A_n$ be the initial patrolling windows, with $A_i \subseteq D_i$ for all $i \in \{1, \ldots, n\}$. Let the cameras implement the Algorithm 3. Then,*

1. *for all iterations $k \in \mathbb{N}$ and for all $i \in \{1, \ldots, n\}$ the patrolling window $A_i(k)$ satisfies $A_i(k) \subseteq D_i$,*

2. *for all iterations $k \in \mathbb{N}$ the set $\{A_1(k), \ldots, A_n(k)\}$ is a partition of Γ, and*

$$\tau^* = \lim_{k\to\infty} \max_{i\in\{1,\ldots,n\}} \frac{d_i(k)}{v_i^{max}} = \min_{\mathcal{P}} \max_{i\in\{1,\ldots,n\}} \frac{d_i}{v_i^{max}},$$

   where $\mathcal{P}$ is the set of partitions $\{A_1, \ldots, A_n\}$ of Γ satisfying $A_i \subseteq D_i$ for all $i \in \{1, \ldots, n\}$ and d_i is the length of A_i, and

3. *the cameras trajectory generated by the REC algorithm is asymptotically $2\tau^*$-periodic, and it converges to an Equal-waiting trajectory.*

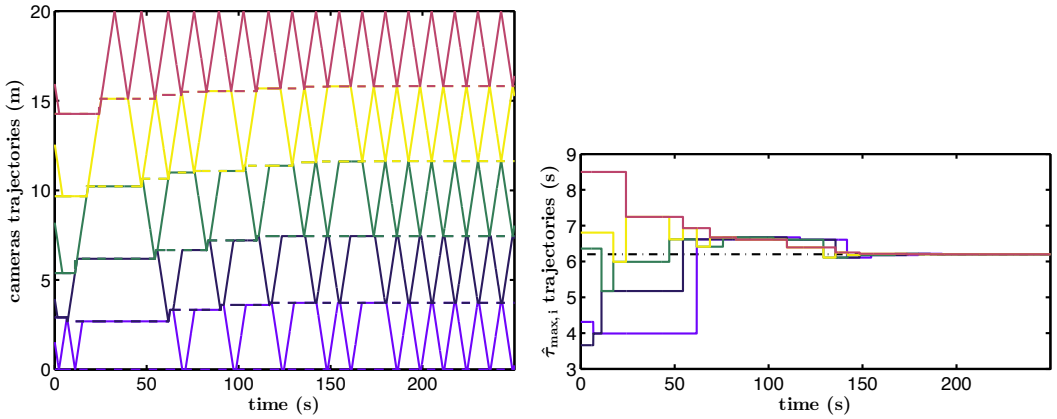
Proof. In the interest of space, we only sketch the proof. First, notice that the update of r_i and ℓ_i is such that A_i belongs to the constraint set D_i, so that statement (i) follows. Second notice that cameras persistently communicate over time. Indeed,

(i) each camera sweeps back and forth its assigned segment, (ii) cameras wait at their boundaries until communication with a neighboring camera takes place, and (iii) cameras 1 and n do not stop at ℓ_1 and r_n, respectively. In particular, it can be shown that any two neighboring cameras communicate within an interval of finite length. Then, statement (ii) follows from [32, Theorem IV.1]. Third, because of the persistence of communication among cameras, the value $\tau^{\max}(k)$, which is decreasing in k, propagates in some time T_{prop} to every camera, for every iteration k. Let $\bar{t}$ be such that $\tau^{\max}(\bar{t}) = \tau^* + \varepsilon$, for some $\varepsilon \in \mathbb{R}_{>0}$. Then, after time $\bar{t} + T_{\text{prop}}$, the period T_i of c_i is within 2ε of $2\tau^*$, for all $i \in \{1, \ldots, n\}$. Statement (iii) follows by letting ε tend to zero. □

As stated in Theorem 6.1, Algorithm 3 drives the cameras towards an Equal-waiting trajectory. Then, the detection performance of the cameras trajectory generated by our reconfiguration algorithm are as in Theorem 3.1 with $\tau^{\max} = \tau^*$.

We now validate our reconfiguration algorithm via simulation. We consider two scenarios with 5 cameras. All cameras start their trajectory from some initial point in their patrolling window. In the first scenario (Fig. 12) cameras have the same maximum speed $v^{\max} = 0.67$ m/s, and they are not subject to patrolling windows constraints. Relevant parameters for this simulation study are reported in Table 2. Observe from Fig. 12 that the cameras trajectory converges to an Equal-waiting trajectory, and that the length of the largest patrolling window $\tau^{\max}$ is decreasing and converges to $\tau^* = 6.2023$ s.

In the second scenario, cameras have different maximum speeds ($v_1 = 0.61$, $v_2 = 0.57$, $v_3 = 0.47$, $v_4 = 0.68$, $v_5 = 0.68$ m/s), and they are subject to patrolling



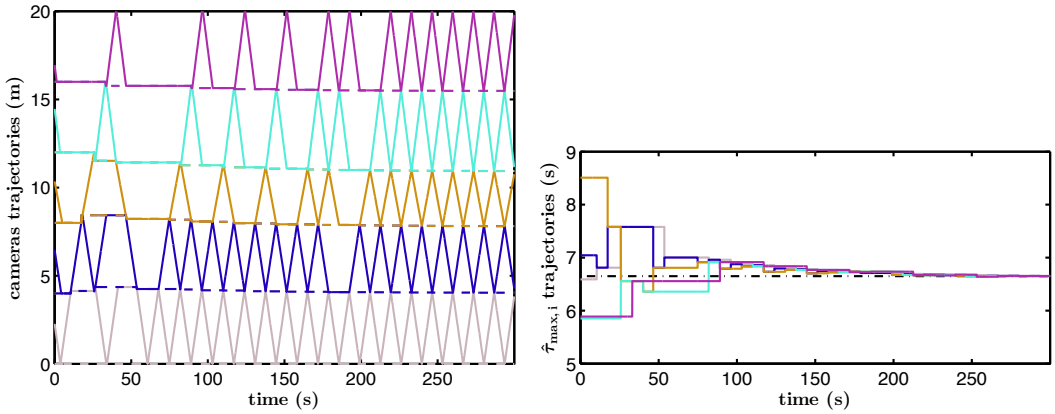
(a) (b)

Figure 12: Simulation of REC with $n = 5$ cameras with maximum speed $v^{\max} = 0.67$ m/s and patrolling windows constraints. In Fig. 12(a) we show the cameras trajectories starting from random positions. The dashed lines refer to the trajectories of the active boundaries. In Fig. 12(b) we report the dynamics of the longest patrolling time $\hat{\tau}_i^{\max}$. Notice that $\hat{\tau}_i^{\max}$ converges to the optimal value $\tau^* = 6.2023$ s (dash-dot line).

Table 2: Parameters and results for uniform cameras speed.

	c_1	c_2	c_3	c_4	c_5
D_i	[0 4.68]	[1.14 7.45]	[3.32 12.09]	[7.26 18.41]	[10.12 20]
$A_i(0)$	[0 2.91]	[2.91 5.38]	[5.38 9.67]	[9.67 14.26]	[14.26 20]
$A_i(\infty)$	[0 3.72]	[3.72 7.45]	[7.45 11.63]	[11.63 15.82]	[15.82 20]

windows constraints. Relevant parameters for this simulation study are reported in Table 3. As shown in Fig. 13, the cameras trajectory converges to an Equal-waiting trajectory, and that the length of the largest patrolling window $\tau^{\max}$ is decreasing and converges to $\tau^* = 6.65$ s.



(a) (b)

Figure 13: Simulation of REC with $n = 5$ cameras with non-uniform maximum speeds $v_i^{\max} \sim \mathcal{U}[0.45, 0.75]$ m/s and no patrolling windows constraints. In Fig. 13(a) we show the cameras trajectories starting from random positions. The dashed lines refer to the trajectories of the active boundaries. In Fig. 13(b) we report the dynamics of $\hat{\tau}_i^{\max}$. Notice that $\hat{\tau}_i^{\max}$ converges to the optimal value $\tau^* = 6.65$ s (dash-dot line).

Table 3: Parameters and results for non-uniform cameras speed.

	c_1	c_2	c_3	c_4	c_5
D_i	[0 20]	[0 20]	[0 20]	[0 20]	[0 20]
$A_i(0)$	[0 4]	[4 8]	[8 12]	[12 16]	[16 20]
$A_i(\infty)$	[0 4.04]	[4.04 7.81]	[7.81 10.94]	[10.94 15.48]	[15.48 20]

7 Conclusions and Future Work

This work studies the problem of coordinating a team of autonomous cameras along a one-dimensional open path to detect moving intruders. We propose mathematical models of cameras and intruders, and we define the worst-case and average detection times as performance criteria. We propose cameras trajectories with performance guarantees, and distributed algorithms to coordinate the motion of the cameras. Finally, we validate our theoretical findings and show effectiveness of our algorithms via simulations and experiments.

Several extensions to this work are of interest. First, we envision extension to more general situations, such as tree-like and cyclic environments. Second, additional performance metrics can be defined to compare different strategies, capturing robustness of the coordination algorithm and predictability of the surveillance strategy. Third, the possibility of having cameras f.o.v.s with different and non-constant velocity profiles. Finally, the problem of jointly estimating the cameras positions and developing a sweeping strategy to prevent intruders from entering and exiting the environment at strategic locations identified by suitable probability density functions.

8 Appendix: Working Code

In this section, we provide the python code used to conduct the experiments on our hardware.

```
#Import Packages

import threading
import Queue
import urllib2
import numpy
import math
from math import pi
import time
import curses
import curses.ascii
import pdb
import pylab
import cv
import sys

#Defining Object Classes

HIST_BINS = 16

class Enum(object):
  def __init__(self, names):
    self._names = names
    for index, name in enumerate(names):
      setattr(self, name, index)

  def name(self, value):
    return self._names[value]

states = Enum(['WAIT_LEFT', 'WAIT_RIGHT',
```

```
                'MOVE_LEFT', 'MOVE_RIGHT',
                'STALL'])

messages = Enum(['LEFT_READY', 'RIGHT_READY',
                  'STALL', 'RESUME'])

def HueImage(frame):
    frame_hsv = cv.CreateImage((frame.width, frame.height), 8, 3)
    cv.CvtColor(frame, frame_hsv, cv.CV_BGR2HSV)
    frame_hue = cv.CreateImage((frame.width, frame.height), 8, 1)
    cv.Split(frame_hsv, frame_hue, None, None, None)
    return frame_hue

#Defining Camera Object Class

class Camera(object):
  def __init__(self, number, d, a1, a2, d_max, url1, url2,
  url3, time_initial,speed1,height,correction_factor, hist):
    self._l_neighbor = None
    self._r_neighbor = None
    self.history = []
    self.timehistory = []
    self.teardown = False
    self._number = number
    self._d=d
    self._a1=a1
    self._a2=a2
    self._d_max=d_max
    self._url1=url1
    self._url2=url2
    self._url3=url3
    self._time_initial=time_initial
    self._timewait = 0
    self._speed1 = speed1
    self._height = height
    self._correction_factor=correction_factor
    self._flag=1
    self._last=0
```

```
  self._resume=None
  self._hist=hist

  self._state = states.WAIT_LEFT
  self._queue = Queue.Queue()
  self._l_ready = False
  self._r_ready = False

#Image Processing Commands

def _HueImage(self,frame):
  frame_hsv = cv.CreateImage((frame.width, frame.height), 8, 3)
  cv.CvtColor(frame, frame_hsv, cv.CV_BGR2HSV)
  frame_hue = cv.CreateImage((frame.width, frame.height), 8, 1)
  cv.Split(frame_hsv, frame_hue, None, None, None)
  return frame_hue

#Defining Neighbors

def setNeighbors(self, l_neighbor, r_neighbor):
  self._l_neighbor = l_neighbor
  self._r_neighbor = r_neighbor

#Updating States

def _handleMessage(self, message):
  if message == messages.LEFT_READY:
    self._l_ready = True
    self._timewait = time.time()-self._time_initial
  elif message == messages.RIGHT_READY:
    self._r_ready = True
    self._timewait = time.time()-self._time_initial
  elif message == messages.STALL:
    self._resume = self._state
    self._state = states.STALL
  elif message == messages.RESUME:
    self._state = self._resume
```

```
#Position Query

def _findPosition(self):
  pos=urllib2.urlopen(self._url2)
  for line in pos:
    name,value=line.strip().split('=')
    if name=='pan':
      position=float(value)
  return position

#Time Query

def _findTime(self):
  t=time.time()-self._time_initial
  return t

#State Query

def state(self):
  return self._state

#Movement Commands

def _move(self, position, direction):

  speed1 = 180*(self._speed1/(self._height
  *math.pow(numpy.tan(pi*position/180),2)+self._height))/pi

  if self._number==0 or self._number==3:
    if (position<=self._a1 and direction==states.MOVE_LEFT)
    or (position>=self._a2 and direction==states.MOVE_RIGHT):
      urllib2.urlopen(self._url1 % 0)
      return True
    else:
      if direction==states.MOVE_RIGHT:
        urllib2.urlopen(self._url1 % speed1)
        return False
```

```
      if direction==states.MOVE_LEFT:
        urllib2.urlopen(self._url1 % -speed1)
        return False

  if self._number!=0 and self._number!=3:
    if (position>=self._a1 and direction==states.MOVE_LEFT)
    or (position<=self._a2 and direction==states.MOVE_RIGHT):
      urllib2.urlopen(self._url1 % 0)
      print time.time()-self._time_initial
      return True
    else:
      if direction==states.MOVE_RIGHT:
        urllib2.urlopen(self._url1 % -speed1)
        return False

      if direction==states.MOVE_LEFT:
        urllib2.urlopen(self._url1 % speed1)
        return False

#Commands for Collecting Initial Data

def initialData(self):
  position = self._findPosition()
  t = self._findTime()
  if self._number==0 or self._number==3:
    self.history.append(numpy.tan(pi*position/180)
    *self._height+self._correction_factor)
  else:
    self.history.append(-1*numpy.tan(pi*position/180)
    *self._height+self._correction_factor)
  self.timehistory.append(t)

#Last Position Query

def lastPos(self):
  return self._last
```

```
#Defining Stall Commands for Simulating Failures

def toggleStall(self):
  if self._state==states.STALL:
    self._queue.put(messages.RESUME)
  else:
    self._queue.put(messages.STALL)

#Defining the Tick commands, This will be looped on each camera.

def _tick(self):

  #Image Processing

  capture = cv.CreateFileCapture(self._url3)
  frame = cv.QueryFrame(capture)
  frame_hue = self._HueImage(frame)
  back_project=cv.CreateMat(frame_hue.height,
  frame_hue.width,cv.CV_8UC1)
  cv.CalcBackProject([frame_hue],back_project,self._hist)
  cv.Threshold(back_project,back_project,
  70,255,cv.CV_THRESH_BINARY)
  contours = cv.CreateMemStorage(0)
  seq = cv.FindContours(back_project,contours)
  A=[]
  A.append(cv.ContourArea(seq))
  while seq!=None:
    A.append(cv.ContourArea(seq))
    seq=seq.h_next()
  n=numpy.max(A)

  threshhold=[10000, 10000, 10000, 10000, 10000, 10000]
  if n>=threshhold[self._number]:

     t=time.time()-self._time_initial
     cv.NamedWindow("frame%f"%t,cv.CV_WINDOW_AUTOSIZE)
     cv.ShowImage("frame%f"%t,frame)
     cv.WaitKey(100)
```

```
# Empty message queue

while True:
  try:
    message = self._queue.get_nowait()
  except Queue.Empty:
    break
  self._handleMessage(message)

# Check position and time
position = self._findPosition()
self._last=position
t = self._findTime()
if self._number==0 or self._number==3:

  self.history.append(numpy.tan(pi*position/180)
  *self._height+self._correction_factor)
else:
  self.history.append(-1*numpy.tan(pi*position/180)
  *self._height+self._correction_factor)
self.timehistory.append(t)

# Handle states
if self._state == states.WAIT_LEFT:
  if self._l_ready and self._flag==1:
    self._timewait=time.time()-self._time_initial
    self._flag=0
  if self._l_ready and (t-self._timewait)>=(self._d_max-self._d):
    self._state = states.MOVE_RIGHT
    if self._l_neighbor:
      self._l_ready = False
elif self._state == states.WAIT_RIGHT:
  if self._r_ready and self._flag==0:
    self._timewait=time.time()-self._time_initial
    self._flag=1
  if self._r_ready and (t-self._timewait)>=(self._d_max-self._d):
    self._state = states.MOVE_LEFT
```

```
        if self._r_neighbor:
          self._r_ready = False
    elif self._state == states.MOVE_LEFT
    or self._state == states.MOVE_RIGHT:
      complete = self._move(position, self._state)
      if complete:
        if self._state == states.MOVE_LEFT:
          self._state = states.WAIT_LEFT
          if self._l_neighbor:
            self._l_neighbor._queue.put(messages.RIGHT_READY)
        else:
          self._state = states.WAIT_RIGHT
          if self._r_neighbor:
            self._r_neighbor._queue.put(messages.LEFT_READY)
    elif self._state == states.STALL:
      urllib2.urlopen(self._url1 % 0)

  def _loop(self):
    while not self.teardown:
      self._tick()

  #Multi-threading commands

  def start(self):
    p = threading.Thread(target=self._loop)
    p.start()

def main(stdscr):

  #Defining Variables

  speed=[8.8,7.1,8.1,9.0,7.5,6.8]
  height=[110.5, 112.0, 113.0, 114.0, 112.0, 112.0]

  d=[245.8,114.3,114.6,243.4,130.5,91.5]
  d_new=[]

  for index, i in enumerate(d):
```

```
  d_new.append(i/speed[index])

d_max=numpy.max(d_new)
print(d_max)

correction_factor = [142.5, 315.8, 407.4, 605.4, 787.5, 874.4]

for i in range(len(d)):
  d[i]=d[i]/speed[i]

a1=[-50.2,27.5,14,-43.3,26,7]
a2=[34,-13.7,-22.5,37.2,-23,-26.3]
tilt=[-3,-3,-2,-3, -3, -3]

s=[50.0,50.0,50.0,50.0,50.0,50.0]

time_initial=time.time()
capture = cv.CreateFileCapture(sys.argv[1])

# Get initial box
frame = cv.QueryFrame(capture)

# Calculate histogram for reference image
frame_hue = HueImage(frame)
hist=cv.CreateHist([HIST_BINS], cv.CV_HIST_ARRAY, [(0, 180)], 1)
cv.CalcHist([frame_hue], hist)
cv.NormalizeHist(hist, 255)

#Define a Camera Object for each Camera with relevant parameters

cameras = []
cameras.append(Camera(0, d[0], a1[0], a2[0], d_max,
'http://192.168.1.21/axis-cgi/com/ptz.cgi?continuouspantiltmove=%d,0',
'http://192.168.1.21/axis-cgi/com/ptz.cgi?query=position',
'http://192.168.1.21/jpg/image.jpg',
time_initial,s[0],height[0],correction_factor[0], hist))
cameras.append(Camera(1, d[1], a1[1], a2[1], d_max,
'http://192.168.1.23/axis-cgi/com/ptz.cgi?continuouspantiltmove=%d,0',
```

```
'http://192.168.1.23/axis-cgi/com/ptz.cgi?query=position',
'http://192.168.1.23/jpg/image.jpg',
time_initial,s[1],height[1],correction_factor[1], hist))
cameras.append(Camera(2, d[2], a1[2], a2[2], d_max,
'http://192.168.1.22/axis-cgi/com/ptz.cgi?continuouspantiltmove=%d,0',
'http://192.168.1.22/axis-cgi/com/ptz.cgi?query=position',
'http://192.168.1.22/jpg/image.jpg',
time_initial,s[2],height[2],correction_factor[2], hist))
cameras.append(Camera(3, d[3], a1[3], a2[3], d_max,
'http://192.168.1.26/axis-cgi/com/ptz.cgi?continuouspantiltmove=%d,0',
'http://192.168.1.26/axis-cgi/com/ptz.cgi?query=position',
'http://192.168.1.26/jpg/image.jpg',
time_initial,s[3],height[2],correction_factor[3], hist))
cameras.append(Camera(4, d[4], a1[4], a2[4], d_max,
'http://192.168.1.24/axis-cgi/com/ptz.cgi?continuouspantiltmove=%d,0',
'http://192.168.1.24/axis-cgi/com/ptz.cgi?query=position',
'http://192.168.1.24/jpg/image.jpg',
time_initial,s[4],height[4],correction_factor[4], hist))
cameras.append(Camera(5, d[5], a1[5], a2[5], d_max,
'http://192.168.1.25/axis-cgi/com/ptz.cgi?continuouspantiltmove=%d,0',
'http://192.168.1.25/axis-cgi/com/ptz.cgi?query=position',
'http://192.168.1.25/jpg/image.jpg',
time_initial,s[5],height[5],correction_factor[5], hist))

cameras[0].setNeighbors(None,cameras[1])
cameras[1].setNeighbors(cameras[0],cameras[2])
cameras[2].setNeighbors(cameras[1],cameras[3])
cameras[3].setNeighbors(cameras[2],cameras[4])
cameras[4].setNeighbors(cameras[3],cameras[5])
cameras[5].setNeighbors(cameras[4],None)

cameras[0]._l_ready=True
cameras[5]._r_ready=True

for camera in cameras:
  if camera._l_neighbor:
    camera._l_neighbor._queue.put(messages.RIGHT_READY)
```

```
#Initialize Camera Position

urllib2.urlopen('http://192.168.1.21/axis-cgi/com/ptz.cgi
?pan=%d'%-52.2)
urllib2.urlopen('http://192.168.1.23/axis-cgi/com/ptz.cgi
?pan=%d'%32)
urllib2.urlopen('http://192.168.1.22/axis-cgi/com/ptz.cgi
?pan=%d'%22.7)
urllib2.urlopen('http://192.168.1.26/axis-cgi/com/ptz.cgi
?pan=%d'%-48.9)
urllib2.urlopen('http://192.168.1.24/axis-cgi/com/ptz.cgi
?pan=%d'%31.8)
urllib2.urlopen('http://192.168.1.25/axis-cgi/com/ptz.cgi
?pan=%d'%13.1)

urllib2.urlopen('http://192.168.1.21/axis-cgi/com/ptz.cgi
?tilt=%d'%tilt[0])
urllib2.urlopen('http://192.168.1.23/axis-cgi/com/ptz.cgi
?tilt=%d'%tilt[1])
urllib2.urlopen('http://192.168.1.22/axis-cgi/com/ptz.cgi
?tilt=%d'%tilt[2])
urllib2.urlopen('http://192.168.1.26/axis-cgi/com/ptz.cgi
?tilt=%d'%tilt[3])
urllib2.urlopen('http://192.168.1.24/axis-cgi/com/ptz.cgi
?tilt=%d'%tilt[4])
urllib2.urlopen('http://192.168.1.25/axis-cgi/com/ptz.cgi
?tilt=%d'%tilt[5])

urllib2.urlopen('http://192.168.1.21/axis-cgi/com/ptz.cgi
?zoom=6000')
urllib2.urlopen('http://192.168.1.23/axis-cgi/com/ptz.cgi
?zoom=6000')
urllib2.urlopen('http://192.168.1.22/axis-cgi/com/ptz.cgi
?zoom=6000')
urllib2.urlopen('http://192.168.1.26/axis-cgi/com/ptz.cgi
?zoom=6000')
urllib2.urlopen('http://192.168.1.24/axis-cgi/com/ptz.cgi
?zoom=5900')
```

```
urllib2.urlopen('http://192.168.1.25/axis-cgi/com/ptz.cgi
?zoom=6000')


while time.time()-time_initial<=3:
  for camera in cameras:
    camera.initialData()

#Start Each Thread

for camera in cameras:
  camera.start()

#Listen for stall or teardown command

curses.halfdelay(10)
teardown=False
while not teardown:
  for index, camera in enumerate(cameras):
    state = states.name(camera.state())
    last_pos = camera.lastPos()
    d_max = camera._d
    stdscr.addstr(index, 0, 'Camera %d: %s %f %f' %
    (index, state, last_pos,d_max))

  c = stdscr.getch()
  if c>= 0:
    if c == ord('x'):
      teardown = True
      for camera in cameras:
        camera.teardown = True
    elif c == ord('t'):
      print time.time()-time_initial
    elif curses.ascii.isdigit(c):
      index = int(c)-ord('0')
      if index < len(cameras):
        cameras[index].toggleStall()
T=[]
```

```
  History=[]

  for camera in cameras:
    T.append(numpy.array(camera.timehistory))
    History.append(numpy.array(camera.history))

  #Plot results

  History[:]=[history*2.54 for history in History]
  pylab.ion()
  pylab.hold(True)
  for index, camera in enumerate(cameras):
    pylab.plot(T[index],History[index])
  pylab.xlabel('time (s)')
  pylab.ylabel('position of f.o.v. (cm)')
  pylab.axis([0,1000,0,2500])
  pylab.savefig('fig.eps')

if __name__ == '__main__':
  curses.wrapper(main)
```

References

[1] J. Clark and R. Fierro, "Mobile robotic sensors for perimeter detection and tracking," *ISA Transactions*, vol. 46, no. 1, pp. 3–13, 2007.

[2] D. B. Kingston, R. W. Beard, and R. S. Holt, "Decentralized perimeter surveillance using a team of UAVs," vol. 24, no. 6, pp. 1394–1404, 2008.

[3] S. Susca, S. Martínez, and F. Bullo, "Monitoring environmental boundaries with a robotic sensor network," vol. 16, no. 2, pp. 288–296, 2008.

[4] Y. Elmaliach, A. Shiloni, and G. A. Kaminka, "A realistic model of frequency-based multi-robot polyline patrolling," in *International Conference on Autonomous Agents*, Estoril, Portugal, May 2008, pp. 63–70.

[5] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2000.

[6] J. Czyzowicz, L. Gąsieniec, A. Kosowski, and E. Kranakis, "Boundary patrolling by mobile agents with distinct maximal speeds," in *European Symposium on Algorithms*, Saarbrücken, Germany, Sep. 2011, pp. 701–712.

[7] A. Machado, G. Ramalho, J. D. Zucker, and A. Drogoul, "Multi-agent patrolling: An empirical analysis of alternative architectures," in *Multi-Agent-Based Simulation II*, 2003, pp. 155–170.

[8] Y. Chevaleyre, "Theoretical analysis of the multi-agent patrolling problem," in *IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, Beijing, China, Sep. 2004, pp. 302–308.

[9] A. Gusrialdi, R. Dirza, T. Hatanaka, and M. Fujita, "Improved distributed coverage control for robotic visual sensor network under limited energy storage," *International Journal of Imaging & Robotics*, vol. 10, no. 2, pp. 58–74, 2013.

[10] F. Pasqualetti, A. Franchi, and F. Bullo, "On cooperative patrolling: Optimal trajectories, complexity analysis and approximation algorithms," vol. 28, no. 3, pp. 592–606, 2012.

[11] M. Baseggio, A. Cenedese, P. Merlo, M. Pozzi, and L. Schenato, "Distributed perimeter patrolling and tracking for camera networks," Atlanta, GA, USA, Dec. 2010, pp. 2093–2098.

[12] R. Carli, A. Cenedese, and L. Schenato, "Distributed partitioning strategies for perimeter patrolling," San Francisco, CA, USA, Jun. 2011, pp. 4026–4031.

[13] D. Borra, F. Pasqualetti, and F. Bullo, "Continuous graph partitioning for camera network surveillance," Jul. 2012, submitted.

[14] S. M. Huck, N. Kariotoglou, S. Summers, D. M. Raimondo, and J. Lygeros, "Design of importance-map based randomized patrolling strategies," in *Complexity in Engineering (COMPENG)*, Jun. 2012, pp. 1–6.

[15] D. M. Raimondo, N. Kariotoglou, S. Summers, and J. Lygeros, "Probabilistic certification of pan-tilt-zoom camera surveillance systems," Orlando, FL, USA, Dec. 2011, pp. 2064–2069.

[16] M. Spindler, F. Pasqualetti, and F. Bullo, "Distributed multi-camera synchronization for smart-intruder detection," Jun. 2012, pp. 5120–5125.

[17] F. Zanella, F. Pasqualetti, R. Carli, and F. Bullo, "Simultaneous boundary partitioning and cameras synchronization for optimal video surveillance," Santa Barbara, CA, USA, Sep. 2012, pp. 1–6.

[18] F. Pasqualetti, F. Zanella, J. R. Peters, M. Spindler, R. Carli, and F. Bullo, "Camera network coordination for intruder detection," Jan. 2013, submitted.

[19] R. Bodor, A. Drenner, P. Schrater, and N. Papanikolopoulos, "Optimal camera placement for automated surveillance tasks," *Journal of Intelligent and Robotic Systems*, vol. 50, pp. 257–295, 2007.

[20] J. Zhao, S.-C. Cheung, and T. Nguyen, "Optimal camera network configurations for visual tagging," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 4, pp. 464–479, 2008.

[21] B. Dieber, C. Micheloni, and B. Rinner, "Resource-aware coverage and task assignment in visual sensor networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 10, pp. 1424–1437, 2011.

[22] C. Micheloni, B. Rinner, and G. L. Foresti, "Video analysis in pan-tilt-zoom camera networks," *IEEE Signal Processing Magazine*, vol. 27, no. 5, pp. 78–90, 2010.

[23] S.-N. Lim, A. Elgammal, and L. S. Davis, "Image-based pan-tilt camera control in a multi-camera surveillance environment," in *Int. Conf. on Multimedia and Expo*, vol. I, Baltimore, MD, USA, Jul. 2003, pp. 645–648.

[24] F. Z. Qureshi and D. Terzopoulos, "Proactive PTZ camera control," in *Distributed Video Sensor Networks*, 2011, pp. 273–287.

[25] ——, "Surveillance camera scheduling: A virtual vision approach," *Multimedia Systems*, vol. 12, no. 3, pp. 269–283, 2006.

[26] C. Ding, B. Song, A. Morye, J. A. Farrell, and A. K. Roy-Chowdhury, "Collaborative sensing in a distributed PTZ camera network," *IEEE Transactions on Image Processing*, vol. 21, no. 7, pp. 3282–3295, 2012.

[27] C. Soto, B. Song, and A. K. Roy-Chowdhury, "Distributed multi-target tracking in a self-configuring camera network," in *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, Jun. 2009, pp. 1486–1493.

[28] Y. Zhu, N. M. Nayak, and A. K. Roy Chowdhury, "Context-aware activity recognition and anomaly detection in video," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 91–101, 2013.

[29] R. D. Quinn, G. C. Causey, F. L. Merat, D. M. Sargent, N. A. Barendt, W. S. Newman, V. B. Velasco Jr, A. Podgurski, L. S. Jo, J.-Y. Sterling, and Y. Kim,

"An agile manufacturing workcell design," *IIE Transactions on Design and Manufacturing*, vol. 29, no. 10, pp. 901–909, 1997.

[30] T. Borangiu, P. Gilbert, N.-A. Ivanescu, and A. Rosu, "An implementing framework for holonic manufacturing control with multiple robot-vision stations," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 4, pp. 505–521, 2009.

[31] S. S. Nestinger, B. Chen, and H. H. Cheng, "A mobile agent-based framework for flexible automation systems," *IEEE/ASME Transactions on Mechatronics*, vol. 15, no. 6, pp. 942–951, 2010.

[32] R. Alberton, R. Carli, A. Cenedese, and L. Schenato, "Multi-agent perimeter patrolling subject to mobility constraints," Jun. 2012, pp. 4498–4503.

[33] A. Kawamura and Y. Kobayashi, "Fence patrolling by mobile agents with distinct speeds," in *Algorithms and Computation*, ser. Lecture Notes in Computer Science, 2012, pp. 598–608.

[34] C. G. Cassandras, X. Lin, and X. C. Ding, "An optimal control approach to the multi-agent persistent monitoring problem," 2013, to appear.

[35] M. Spindler, "Distributed multi-camera synchronization for smart-intruder detection," Master's thesis, University of Stuttgart, Sep. 2011.

[36] S. Boyd and L. Vandenberghe, *Convex Optimization*, 2004.