A Negotiation Protocol with Conditional Offers for Camera Handoffs

by

Wiktor Starzyk

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Faculty of Graduate Studies (Computer Science)
University of Ontario Institute of Technology

Supervisor(s): Dr. Faisal Z. Qureshi

Abstract

A Negotiation Protocol with Conditional Offers

for Camera Handoffs

Wiktor Starzyk

Master of Science

Faculty of Graduate Studies

University of Ontario Institute of Technology

2014

This thesis explores the idea of conditional offers during camera handoff negotiations. In a departure from contract-net inspired negotiation models that have been proposed for camera handoffs, the current scheme assumes that each camera maintains the state of its neighbouring cameras. To this end, we develop a new short-term memory model for maintaining a camera's own state and the state of its neighbouring cameras. The fact that each camera is aware of its surrounding cameras is exploited to generate conditional offers during handoff negotiations. This can result in multiple rounds of negotiations during a single handoff, leading to successful handoffs in situations where one of the cameras that is being asked to take on one more task is unable to take on a new task without relinquishing an existing task. The results demonstrate the advantages of the proposed negotiation model over existing models for camera handoffs.

Acknowledgements

First, I would like to thank my supervisor Dr. Faisal Z. Qureshi for allowing me to work with him over the past six years, first as an undergrad and later as a masters student. I am very grateful for the guidance, motivation and opportunities that he gave me. I would also like to acknowledge my fellow lab members Mohamed Helala, Luis Zarrabeitia, Jordan Stadler, Nurjahan Parvin, Zheng Wang, Ryan Shanks, Ernesto Rodriguez Reina, Wesley Taylor and David Nemirovsky as well as all the others I have shared a lab with these past few years. They made my graduate experience enjoyable and rewarding.

Finally, I would like to thank by parents and sisters for supporting and encouraging me throughout these many years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The need for security in public spaces and the plummeting costs associated with camera installations are pushing the growth of video surveillance. Surveilling large environments necessitates the use of multiple cameras, as no single camera is able to observe the entire scene. As the number of cameras grows, it becomes impractical for a human operator to monitor all the video feeds (Figure 1.1). Consequently, over the last several years, there is much work on camera networks capable of providing video coverage of extended spaces with minimal human intervention. The computer vision community is busy developing algorithms for tracking [40], object counting [26, 25], traffic analysis [32, 3], event detection [7, 34], etc. to automate the task of monitoring video streams from a video surveillance installation. Concomitant with the advances in video analysis is the work on smart camera networks. Unlike traditional camera networks that rely upon a central unit to process and store videos captured by CCTV cameras, smart camera networks push processing and storage to individual cameras. Smart camera networks comprise visual sensing nodes (commonly referred to as smart cameras) with onboard processing and storage and the ability to communicate with other cameras in the vicinity.

Smart camera networks offer several advantages over traditional CCTV-based video surveillance systems: 1) the lack of a central processing unit suggests that smart camera networks are highly scalable; 2) the deployment and maintenance costs of these networks are forecasted to

Figure 1.1: A single operator monitoring a large number of surveillance cameras. Source [17].

be much less than those of traditional CCTV-based surveillance systems; 3) these networks are more robust as there is no single point of failure; and 4) these camera networks can be easily reconfigured by adding or removing camera nodes. Camera control and coordination are important research areas within smart camera networks. Specifically, how best to coordinate the available cameras to carry out the observation tasks? Ideally the control and coordination strategy should be distributed, as centralized schemes do not scale and defeat the *raison d'êter* of smart camera networks.

Consider the problem of observing individuals present in an area using a network of cameras. This task requires not only image analysis routines for detecting, tracking, and identifying pedestrians, but also techniques for controlling and coordinating these cameras to best observe the pedestrians present in the scene. Specifically, we need methods to assign each camera with the task of observing a subset of pedestrians at any given instance. This suggests that the problem of scheduling cameras to observe pedestrians is a special instance of a *job scheduling* problem [5]. The key challenge present in camera scheduling is the lack of information about the future actions of individuals under observation. For example, it is often not possible to determine how long an individual might stay within the field-of-view of a camera. Or when

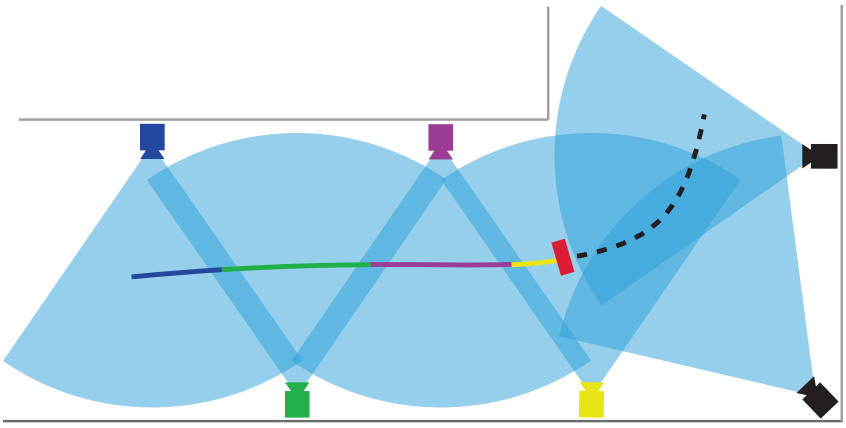


Figure 1.2: As a pedestrian moves through an area such as a hallway in a building, the tracking responsibility needs to be handed off from camera to camera. In this example, the pedestrian represented by a red rectangle moves from the viewing region of the blue camera to the green followed by the purple and then finally enters the yellow cameras viewing region.

an individual might leave the field-of-view of one camera. This observation has led to the development of, what might be referred to as, reactive approaches for camera scheduling, where camera assignments are constructed on-the-fly in response to changing observation goals and pedestrian locations.

Camera handoff is simply shifting the task of observing a pedestrian from one camera to another camera. Camera handoffs occur when an individual (or any other kind of object that is under observation) leaves the field-of-view of one camera and enters that of another camera (Figure 1.2). In this situation, the second camera needs to take on the responsibility for observing this person. For any camera network that is spread over a wide area and is observing a large number of individuals, camera handoffs occur at a high frequency. It is tedious—and for all intent and purposes, infeasible—for a human operator to manage camera handoffs manually, and there is a need to develop a camera handoff strategy capable of detecting and responding to individuals meandering through the field-of-views of different cameras. This thesis studies the problem of camera control within the context of smart camera networks and develops a new negotiation protocol for camera handoffs. The proposed model has two novel features:

- **Short-term memory model for storing the state of neighbouring cameras:** it describes a short-term memory model, which enables each camera to maintain the state

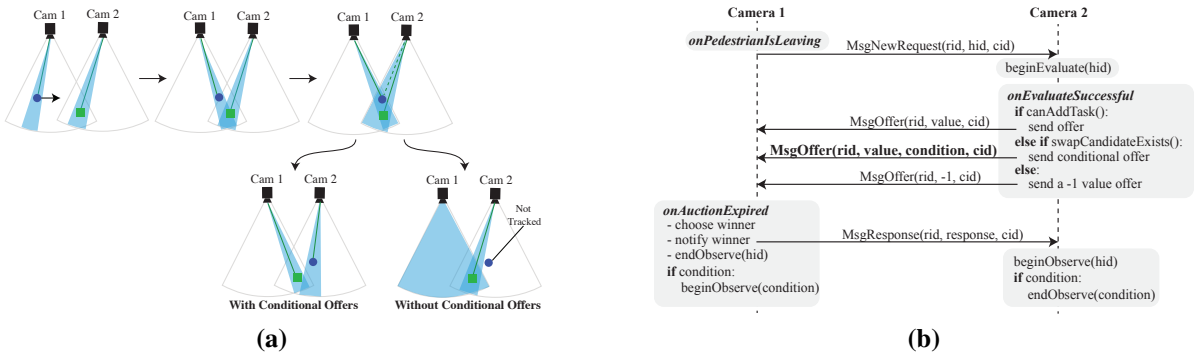


Figure 1.3: (a) Conditional offers allow handoffs to occur even if a camera is already at is tracking limit. They allow a camera to send an offer with a condition that ensures the auctioning camera takes over a task from the bidding camera. (b) Messages exchanged during the handoff operation. Notice that camera 2 proposes a conditional offer suggesting a task swap.

of its neighbouring cameras. Each camera is able to use the remembered state of its neighbouring cameras during handoffs and task assignments.

- **Counter-offers during handoff negotiations:** it implements a mechanism for conditional offers, which allows cameras to go through multiple rounds of negotiations during each handoff task. The ability to propose conditional offers leads to successful handoffs in situations where a camera that is being asked to take on a new task is unable to do so unless it first terminates an existing task.

1.1 Our Approach

The proposed approach is distributed. Camera handoffs are the result of local negotiations between two neighbouring cameras (i.e., cameras that are within the communication range of each other). The lack of a central controller suggests that the proposed technique can be scaled to large camera networks—the scalability properties of the proposed method have yet to be empirically evaluated. We evaluate the proposed negotiation model on a (simulated) network of uncalibrated active *pan/tilt/zoom* (PTZ) cameras. The negotiation model is used to perform handoffs between individual cameras, which are modelled as autonomous behaviour-based

agents. The handoffs ensure that the camera network is able to track multiple targets as these move in the area under observation, weaving in and out of the field-of-views of different cameras. We have also compared our technique against the camera handoff scheme that appeared in [12] and the initial results appear promising.

A Thought Experiment

Consider the scenario shown in Figure 1.3. Camera 1 is observing a pedestrian depicted as the blue circle and camera 2 is tracking an individual depicted as the green square. When camera 1 detects that the blue circle is about to leave its field-of-view, camera 1 requests camera 2 to take over the tracking (or observation) responsibilities for this pedestrian. Camera 2 is already at its tracking limit, i.e., camera 2 can only observe a single pedestrian at any given time.[1] Consequently, camera 2 is unable to take on the task of observing the blue circle. However, in the scheme proposed here, camera 2 is able to propose the following conditional offer to camera 1: camera 2 agrees to observe the blue circle if camera 1 agrees to observe the green square. Camera 2 comes up with this conditional offer by relying upon its internal model of camera 1. Camera 1 agrees and the two cameras swap their pedestrians, resulting in a successful handoff.

In the above example, we have used tracking limit as the reason why camera 2 is unable to take on a new task, which in turn triggered a conditional offer and the second round of negotiations. Tracking limit is but one of the reasons why a camera might be unsuitable for taking on a new task. It is possible that a camera is unable to carry out a particular combination of tasks simultaneously. For example, a *pan/tilt/zoom* (PTZ) camera

[1] The tracking limit of 1 is only used for illustrative purposes in this example. The proposed scheme does not assume that each camera can only track a single individual.

might be unable to view two pedestrians at the opposite ends of its field-of-view with the desired resolution. Such situations are also resolved through conditional offers.

1.1.1 Camera Network Simulation

Setting up a physical camera network of appropriate size and complexity is prohibitively costly for most computer vision researchers. This observation led to the development of virtual vision paradigm for camera networks research [30]. Virtual vision advocates the use of visually and behaviourally realistic environments for studying, designing, and evaluating camera networks. Initially we considered using the virtual vision simulator developed in [39] to evaluate our handoff strategy. We quickly realized that we lack the computational resources to use this simulator for our purposes. We instead decided to develop our own camera network emulator to carry out our work on camera handoffs. Our emulator is able to simulate smart camera networks comprising active PTZ and passive wide-FOV cameras in 2D. While it does not model imaging artifacts of physical cameras, it does support sensing characteristics due to occlusions and limited field-of-views of video cameras.

1.2 Contributions

This thesis makes the following contributions:

1) We build on the work of Qureshi [29] on collaborative sensing in smart camera networks by extending the negotiation protocol that allows cameras to collaborate on tasks.

2) We introduce a memory module to the smart camera node that keeps track of what neighbouring cameras are doing resulting in better decision making during the negotiation process.

3) We present a software framework which can be used to quickly simulate different camera networks making testing different scenarios quick and easy.

4) The work presented in this thesis has been accepted to appear in the Eighth ACM/IEEE

International Conference on Distributed Smart Cameras that will be held in November, 2014 in Venice, Italy.

1.3 Thesis Overview

The remainder of the thesis is organized as follows: the next chapter presents background information that is needed to understand the rest of this thesis and summarizes existing work on camera handoffs. A detailed description of our solution to the handoff problem is provided in Chapter 3. We demonstrate our proposed approach and show how it compares to another recent handoff approach in Chapter 4. Chapter 5 concludes this thesis by summarizing our work and suggesting possible directions for future research.

Chapter 2

Background Material

This chapter covers background material needed to understand the rest of this thesis. We also summarize existing work on camera selection and handoffs with a view to highlight the novel aspects of our handoff strategy.

2.1 Basic Concepts

2.1.1 Camera Networks

Any video surveillance system that spans an extensive space or even a small region divided by visual barriers needs more than one camera to provide adequate visual coverage; cameras have limited FOVs, and no single camera can observe the entire scene. Multiple cameras, and hence *camera networks*, are needed to provide the desired visual coverage in such situations.

A centralized camera network relies upon a central processing unit for processing and storing the video footage (Figure 2.1a). Here each camera simply pipes the captured video to this unit. The central unit is also responsible for controlling and coordinating these cameras to carry out the various observation tasks. Centralized camera networks are appropriate for smaller installations; however, these do not scale well. These networks also typically have very high bandwidth requirements, as each camera sends its video to the central unit. These

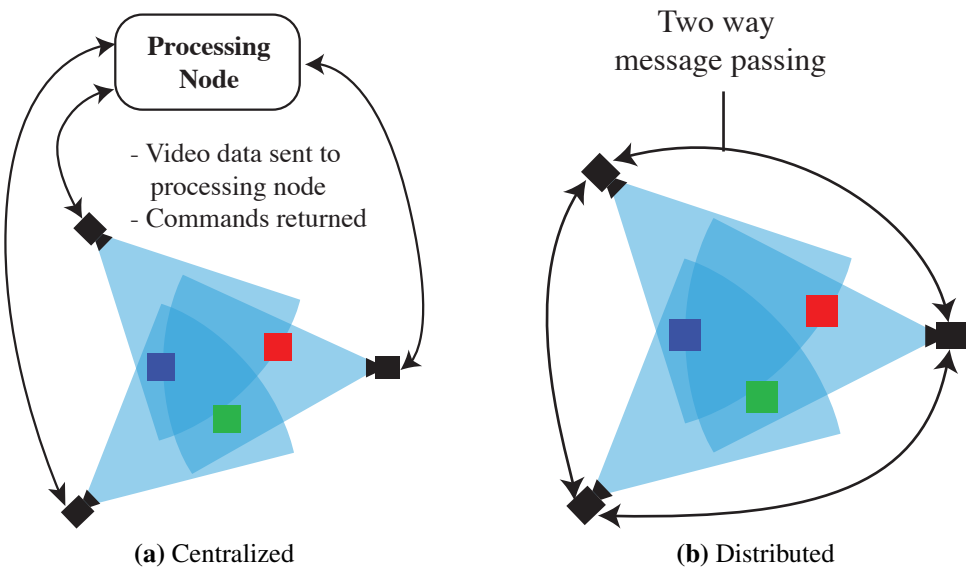
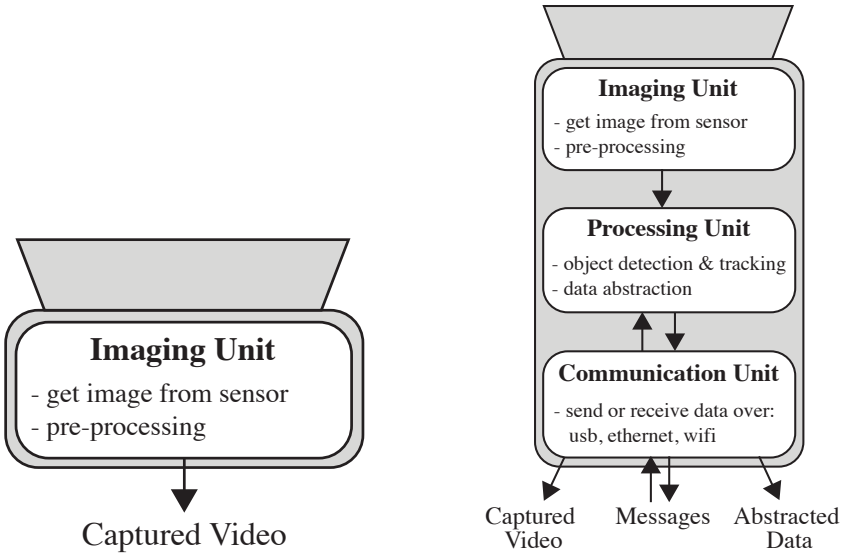


Figure 2.1: In a centralized camera network all cameras communicate with a central processing node. The central processing node is responsible for any coordination or control decisions. In a distributed camera network, the cameras are able to communicate directly with each other allowing them to make their own decisions.

centralized camera networks, however, have one key advantage: it is easy to implement and enforce "access" or "privacy" policies on these networks. These policy control how captured video is stored, archived, accessed, and used.

Distributed camera networks on the other hand do away with a central processing unit (Figure 2.1b). By necessity these networks require smart camera nodes, complete with on board processing and storage. Video processing and storage is distributed across the cameras. Many technical challenges—such as, in-network control and coordination of camera nodes, distributed sensing, energy-aware processing, implementation of operational policies, etc.—must be tackled before distributed camera networks become a reality. We envision that future camera networks will be by necessity distributed and that these networks will provide perceptive coverage of large areas with little or no human supervision. The work presented here is a step towards realizing this vision of distributed camera networks.



(a) A typical camera **(b)** A smart camera

Figure 2.2: A smart camera adds processing and communication modules to a typical camera.

2.1.2 Smart Cameras

Smart cameras combine sensing, processing, and communication in a single package (Figure 2.2). A mobile device with an embedded camera, for example, is a canonical example of a smart camera. Smart cameras are able to process video at source, saving bandwidth required to send raw video to a video processing unit. Within the camera networks community, there is a lot of interest to develop image processing, distributed sensing, and camera control and coordination methods for enabling smart cameras to organize themselves into ad hoc networks and provide visual coverage of extensive spaces. Our work relies upon the processing, storage and communication capabilities provided by smart cameras. Specifically, the proposed short-term memory model and camera handoff strategy is only relevant for smart cameras.

2.1.3 Tracking

For a camera to be fully autonomous, it must be able to detect and track objects. Tracking objects such as pedestrians poses several challenges. Pedestrians can be occluded by other objects or can blend into the background. Pedestrians move around and are very unpredictable

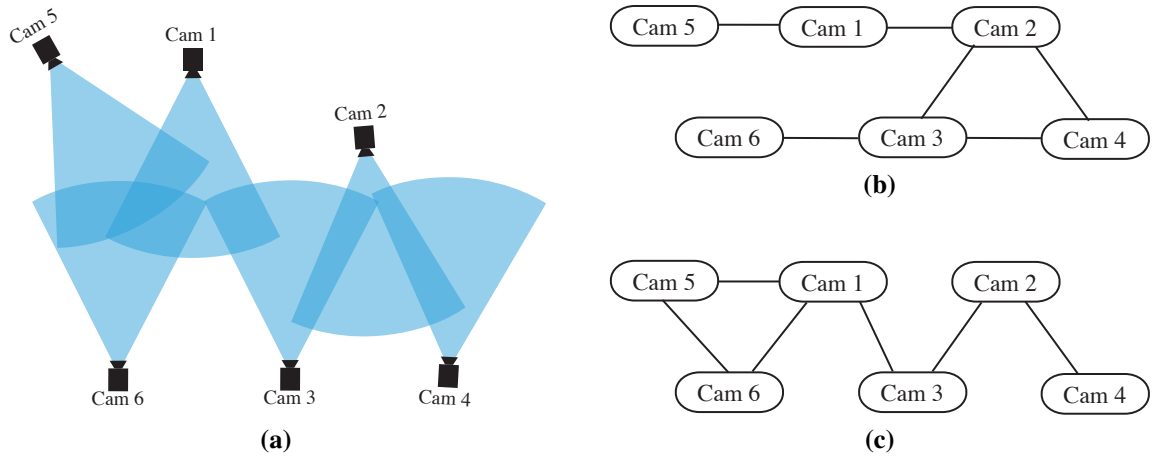


Figure 2.3: A communication graph can optimize the communication between cameras. (a) A sample camera network. (b) The communication graph for the camera network in (a). (c) The vision graph for the network in (a).

making it difficult for cameras to follow them. Multiple pedestrians can also have similar visual appearance, making it difficult for a camera to differentiate between multiple people. Many different approaches for tracking objects have been proposed, and we refer the interested reader elsewhere for a survey of tracking methods [40]. For the purposes of our work, we assume that our cameras are able to track pedestrians reliably. Our assumption is supported by the recent strides made within the computer vision community on object tracking.

2.1.4 Camera Network Topology

Spatial relationship between cameras plays an important role during handoffs. Handoffs are almost always performed between two neighbouring cameras. There are spacial cases, such as person re-identification after a extended observation gap, when a person might be handed off to a non-neighbouring camera. These cases are far and few between. We currently do not concern ourselves with these cases. Camera network topology encodes the spatial relationship between cameras. Given a camera, for example, it is possible to find its neighbours using camera network topology. Camera networks can be represented as a graph, where nodes represent individual cameras, and an edge between two nodes indicates that these two nodes are neighbours. Many schemes exist to construct a network topology for a given camera network. Some

approaches such as [4] work with overlapping camera FOVs, while others such as [10, 22] work with non-overlapping FOVs.

A *vision graph* encodes the observation-relationship between cameras; an edge between two nodes in a vision graph indicates that the two cameras have overlapping field-of-views (Figure 2.3c). A *communication graph* encodes the messaging-relationship between cameras; an edge between two nodes in the communication graph indicates that the two cameras are within the communication range of each other (Figure 2.3b). Esterle et al. proposes the use of ant-colony inspired pheromones to learn a communication graph [13]. Vision and communication graphs can be used during camera handoffs. Distributed handoff approaches, including ours, require that cameras communicate with their neighbours. Communication and vision graphs can be used to identify neighbours, and messaging are then restricted to these cameras only. Broadcast messages waste both bandwidth and processing power—every camera that receives a message ends up spending resources to send back a reply.

2.1.5 Calibration

In simple terms, camera calibration maps pixels to real-world coordinates. If the camera is calibrated, it is possible to find the 3D location of the observed pedestrian (under some assumptions). It is also possible to calibrate the entire network, meaning that the 3D locations of individual camera nodes are known within a common frame of reference. It is possible to estimate the location of every observed object within this frame of reference, which greatly simplifies object tracking and identification. Even if two individuals look identical, they cannot occupy the same physical space. A fact that can be exploited to disambiguate between the two persons. There is a great deal of existing literature about camera calibration [16, 6] and camera network calibration [8, 14]. A deeper discussion about camera calibration is beyond the scope of this work; however, we make the following observations:

> Maintaining camera and network calibration over the lifetime of a camera network is tedious. Particularly so for networks with mobile camera nodes

or PTZ cameras. Consequently, camera control methods that do not rely upon camera or network calibration are preferable to those that do need calibration information. Our method does not assume camera or network calibration. However, it is possible to extend the camera handoff strategy proposed here to use camera or network calibration information when available.

2.1.6 Camera Handoff

As stated earlier, handoff refers to shifting the observation responsibility of an object (in our case, pedestrians) from one camera to another. Even if we ignore the higher-level camera selection problem, camera handoff is challenging due to the vagaries of visual tracking and object recognition. In order for a camera to take over the observation responsibility of a pedestrian from another camera, the two cameras must agree upon the identity of the pedestrian in question. Recognizing a person as he moves from one camera to another is challenging: the same person might look different when viewed from different viewpoints, the two cameras might have different colour responses, etc. There is much work found in the computer vision literature that deals with the problem of *object re-identification* [9, 15]. This work, however, assumes that acceptable solutions exist for object identification problem. This thesis is concerned with the problem of camera selection, i.e., which of the available cameras should next observe the pedestrian. We discuss existing handoff approaches in the following section.

2.2 Existing Approaches for Camera Selection

Many different approaches have been proposed for solving the handoff problem. Some of the most relevant ones to our work are shown in Table 2.1. These various approaches have many different properties such as distributed or centralized, embedded or pc-based, calibrated or uncalibrated and active or static cameras. A comparison of some of these approaches is

Table 2.1: A Comparison Of Handover Approaches.

Approaches	**D**	**C**	**AC**	**RT**	**RD**	**NC**	**NP**	**O**
Javed & Khan [18]	No	No	No	Yes	Yes	2	2	Yes
Park et al. [27]	Yes	No	No	Yes	No	20	N/A	Yes
Jo & Han [19]	No	No	No	Yes	Yes	2	N/A	Yes
Quaritsch et al. [28]	Yes	No	No	Yes	Yes	2	1	No
Morioka et al. [24]	Yes	No	No	N/A	No	6	1	Yes
Li & Bhanu [20]	No	No	No	Yes	Yes	3	2	Yes
Qureshi & Terzopoulos [31]	Yes	Yes	Yes	No	No	16	100	Yes
Song et al. [38]	Yes	Yes	Yes	No	No	14	N/A	Yes
Song & Roy-Chowdhury [37]	Yes	No	No	No	Yes	7	9	No
Qureshi [29]	Yes	No	Yes	Yes	No	4	8	Yes
Esterle et al. [12]	Yes	No	No	Yes	Yes	4	31	No
Our Work	Yes	No	Yes	Yes	No	16	32	Yes

Legend: D–Distributed; C–Calibrated; AC–Active Camera; RT–Real Time; RD–Real Data; NC–Number of Cameras; NP–Number of Pedestrians; O–Overlapping FOVs;

discussed in [21].

One of the first handoff approaches was presented by Javed and Khan in 2000 [18]. They describe a way to discover the spatial relationship between the FOVs of different cameras by projecting the FOV lines of one camera onto another cameras view. The FOV lines can be learned by having a single person walk around to see where cameras viewing regions overlap. When the person is seen by more than one camera at the same time, a constraint point for a FOV line in a cameras image is added. Once there are at least two constraint points, a line can be drawn defining the overlapping region in a cameras image. Knowing the overlapping regions makes it possible to hand off a pedestrian from one camera to another. This approach does not require any calibration, but requires overlapping camera views.

Park et al. propose a solution to the handoff problem based on constructing a lookup table which encodes the suitability of a camera to observe a specific region [27]. This approach works by first calibrating the cameras to a global coordinate system. Once this is done, the viewing frustum of each camera is partitioned into sub-frustums of equal volume. Each sub-frustum is then given a location in world coordinates which is then sent to all neighbouring

cameras. A lookup table is then constructed which ranks cameras based on how well they can image a specific location. Once the lookup table is constructed, a simple lookup operation is performed when a handoff is needed.

Jo and Han introduce the concept of occurrence to co-occurrence to solve the handoff problem [19]. A handoff table is constructed by computing the ratio of co-occurrence to occurrence for all pairs of points in two views. The table maps a domain set of points in one view to a range set of points in another view. When a handoff is needed, the point an object is observed at is looked up in the handoff table to find the corresponding point in another cameras view. The approach does not require calibration and can be applied to a scene with non-planar ground.

Quaritsch et al. propose an approach that relies on a static vision graph that encodes migration regions [28]. The migration regions assign neighbouring cameras to specific areas in a cameras FOV. When an object enters a migration region which is represented as a polygon in image coordinates, the camera that corresponds to that migration region is notified and a handoff occurs. No information about how the migration regions are computed is provided which leads us to believe that they are defined manually.

Morioka et al. propose a fuzzy-based approach for handing over of tracking authority [24]. The camera selection decision is driven by fuzzy reasoning based on the previously selected camera and the tracking level of the object in each camera. Tracking level is defined by estimating the position measurement error in the monitoring area of each camera. The approach is tested using several simulations and does not require calibration.

Li and Bhanu propose a game theoretic approach to the handoff problem [20]. In this approach, cameras use bargaining to decide which camera to handoff a pedestrian to. When a pedestrian is visible in multiple cameras, the best camera is selected based on its expected utility. A number of criteria to construct the utility function is proposed, such as the number of pixels occupied by the person in the image. Their approach eschews spatial and geometric information.

Song et al. also present a game-theoretic approach to the handoff problem [38]. In their

approach, optimal global utility is achieved by having each camera optimize its local utility. PTZ cameras are used to track all targets in the area at an acceptable resolution and some at a high resolution. Negotiations are used to decide which pedestrians will be tracked at the higher resolution. The approach is tested in a simulated environment and requires calibration.

Qureshi and Terzopoulos propose a distributed camera coalition formation scheme for perceptive scene coverage and persistent surveillance in smart camera networks [31]. In the proposed approach, cameras form groups to complete observation tasks. Cameras are added to a group by the use of an auction which is initiated by a group leader. New cameras can be added and old cameras can be removed from the group as the pedestrian moves around the observation region. In cases where multiple groups require the same resources, the Constraint Satisfaction Problem (CSP) technique is used to resolve the conflict. The approach is tested using the Virtual Vision Train Station Simulator [35].

Qureshi proposes a solution to the handoff problem in his work on collaborative sensing via local negotiations [29]. In this work, Qureshi presents a solution to the handoff problem by introducing a negotiation protocol that allows cameras to collaborate on tasks. The camera nodes, which are modelled as behaviour based autonomous agents, are able to ask other cameras to take over a task when they are no longer able to meet tasks requirements. Neighbouring cameras evaluate their suitability to the task and send back an offer. The requesting camera node is then able to choose which camera can do the best job and a handoff is initiated. This thesis is an extension of this work.

In 2012, Esterle et al. present an approach to the handover problem based on self-interested autonomous agents that hand off tasks from one camera to another using a market mechanism [13]. Each camera tries to maximize their own utility by auctioning off tasks they are no longer able or no longer want to complete. The utility computation of a camera is based on how well a camera is able to complete its assigned tasks as well as the payments it has made and received from auctions. Upon receiving an auction initiation message, neighbouring cameras evaluate their suitability for the task and the effect it will have on their own utility. If adding

the task increases their utility, a bid is sent back and the camera with the highest bid wins the auction resulting in a handoff.

All of these approaches fall into two main groups: handoff function based approaches and negotiation based approaches. The handoff function or handoff table based approaches learn the spatial relationship between cameras. They can be used to quickly find out which cameras can see an object or where an object will appear after leaving a cameras field-of-view. This is useful when carrying out an investigation after an incident occurs because it allows the investigator to quickly find all of the video footage showing the person of interest. It is less useful in active camera systems doing real-time analysis. Such systems have a limited amount of resources which must be taken into account when making handoff decisions. Handoff function based approaches do not allow this which is why negotiation based approach are needed.

In negotiation based approaches, camera nodes communicate with each-other to make handoff decisions. This allows camera nodes to take resource limitations into account when agreeing on a handoff. The problem with a lot of these approaches is that they employ self-interested agents that try to maximize their own utility. This works great in some situations because the camera that can do the best job will probably be assigned to an observation task. Occasionally situations will arise where cameras need to help each other out to complete a handoff and to improve global utility. The self-interested approach to agent design will not allow this resulting in observation failures. We hope to solve this problem with our approach that employs agents willing to help each other out whenever possible.

2.3 Afterword

Table 2.1 juxtaposes the proposed method and existing schemes on camera handoffs. Our technique is novel insofar as it provides a mechanism for maintaining the state of neighbouring cameras and uses this state to generate counter-offers during handoff negotiations.

Chapter 3

Approach

We now describe our approach for camera handoffs. Consider an area under observation by n cameras $C = \{c_1, \cdots, c_n\}$. These cameras are tasked with observing the pedestrians present in the scene. Let $H = \{h_1, \cdots, h_m\}$ denote the set of pedestrians, and H_i denote the set of pedestrians currently assigned to camera c_i. Camera assignments evolve over time in response to 1) the arrival of new pedestrians, 2) the departure of individuals currently under observation, 3) the movement of a person, and 4) the changes in the overall observation tasks. Therefore, we need a mechanism to update camera assignments so as to satisfy the overall observation goal(s). Camera handoff problems typically arise when we seek a distributed mechanism for updating camera assignments. Camera handoff strategies often belong to the class of approaches where "global" camera assignments are managed locally at each camera through pairwise interactions between neighboring cameras.

3.1 Smart Camera Nodes

We model each camera as a highly capable behaviour-based autonomous agent. Each camera node has access to a repertoire of behaviours. Figure 3.2 shows the behaviour routines available to our smart cameras. These behaviours range from simple activities, such as turning on a sensing unit, to increasingly complex abilities, such as panning to keep a pedestrian in view. We

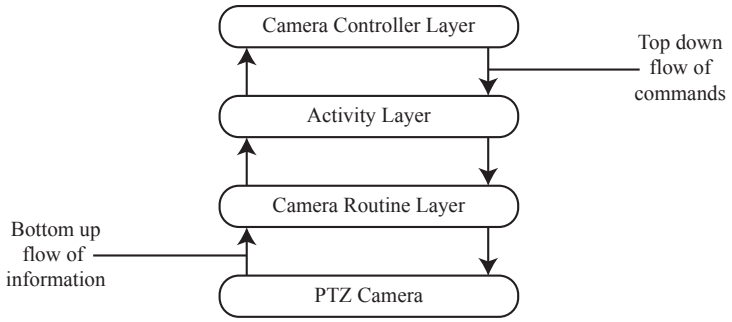


Figure 3.1: Our camera nodes implement a layered architecture with information flowing from the PTZ camera up to the Control Layer and commands flowing back down.

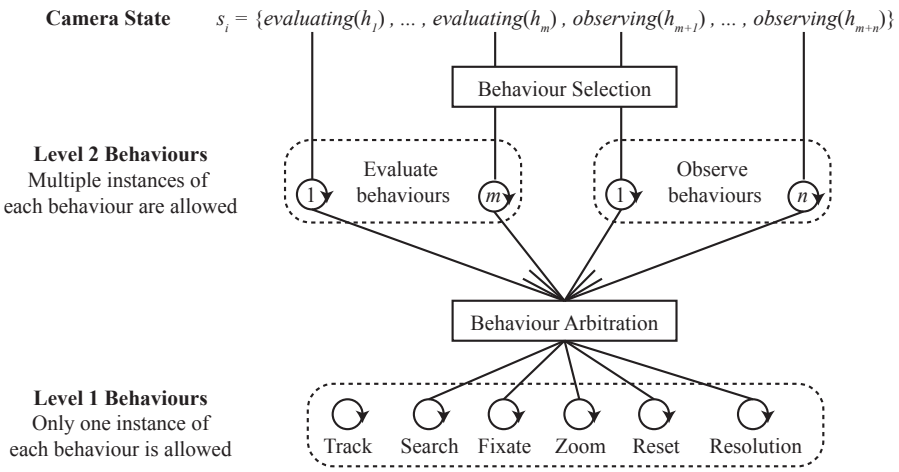


Figure 3.2: Levels of behaviours implemented by our camera nodes. Originally appeared in [29]

take the layered approach to behaviour design which was first popularized by the *subsumption* architecture [2]. Figure 3.1 provides an overview of our smart cameras. At the bottom of our layer architecture is any standard static wide-FOV or active PTZ camera. The second layer is the Camera Routine Layer which implements the level 1 behaviours shown in Figure 3.2. Next is the Activity Layer where the level 2 behaviours are implemented and where the behaviour arbitration takes place. The fourth and final layer in our layered architecture is the Camera Control Layer. This layer handles the high level decision making and communication between cameras, which is the focus of this thesis.

We make the following assumptions about the camera nodes:

- a camera is able to track pedestrians (targets) assigned to it;

- a camera is able to use appearance based signatures for acquiring a new pedestrian (target) for tracking;
- a camera is able to detect when it loses track of a pedestrian; and
- PTZ cameras are able to select appropriate values for pan, tilt and zoom settings to observe the assigned pedestrians.

These assumptions are motivated (and supported) by recent advances in pedestrian detection, recognition and tracking [11] and PTZ camera tracking [23]. Similar assumptions have been made by others [29, 12].

Each camera node is aware of the pedestrians (or targets) present in its FOV. This is easily accomplished by using a pedestrian detection routine. Algorithm 1 is used to maintain the set of pedestrians H_i seen by a camera c_i between times $t - t^{\text{forget}}$ and t, where t represent the current time. $ts(\cdot)$ is an operator that is used to set and retrieve the time-stamps for elements of H_i; it operates on sets. To make things concrete $ts(\mathcal{S}) = t$ sets the time-stamps of every element of set S equal to t; where as, $ts(S)$ assumes that $\|S\| \leq 1$ and returns the time-stamp of the element or the current time if the set is empty. $ts(S)$ for $\|S\| > 1$ is undefined. t_{forget} serves an important purpose by providing a mechanism for ignoring short-duration detection failures of pedestrians present in the scene. In the absence of t_{forget}, H_i might constantly change in response to the output of pedestrian detector.

At any given time, each camera may be engaged in several activities. A camera might be tracking multiple individuals, it might be evaluating its suitability for tracking a new target, or in case of PTZ cameras, a camera might be performing a visual search in the pan/tilt/zoom space to fixate and zoom in on a pedestrian. Ignoring the innards of a camera node, it is possible to keep track of these activities using a finite state machine proposed in [29] (see Figure 3.3). The state of a camera node represents the activities it is currently executing. Given this list of activities, Algorithm 2 maintains the activity set A for a camera c_i. Whether or not a camera is tracking an individual or evaluating its suitability for tracking a new individual is

Algorithm 1 Removing unseen pedestrians from H_i

Require: H_i ▷ The set of pedestrians seen by camera i between times $t - t^{\text{forget}}$ and now (t).
Ensure: Updated H_i

1: Capture a frame at time t
2: Use pedestrian detection (recognition) routines to construct a possibly empty set H_i^t of pedestrians found in frame I.
3: $H_i^{\text{existing}} = H_i^t \bigcap H_i$ ▷ The next four lines update the time-stamps of existing pedestrians in H_i
4: $H_i = H_i \setminus H_i^{\text{existing}}$
5: $ts(H_i^{\text{existing}}) = t$
6: $H_i = H_i \bigcup H_i^{\text{existing}}$
7: $H_i^{\text{new}} = H_i^t \setminus H_i^{\text{existing}}$
8: $ts(H_i^{\text{new}}) = t$
9: $H_i = H_i \bigcup H_i^{\text{new}}$ ▷ Add the previously unseen pedestrians into H_i
10: **for all** $h \in H_i$ **do**
11: **if** $t - ts(\{h\}) > t^{\text{forget}}$ **then**
12: $H_i = H_i \setminus \{h\}$ ▷ Pruning stale entries from H_i
13: **end if**
14: **end for**

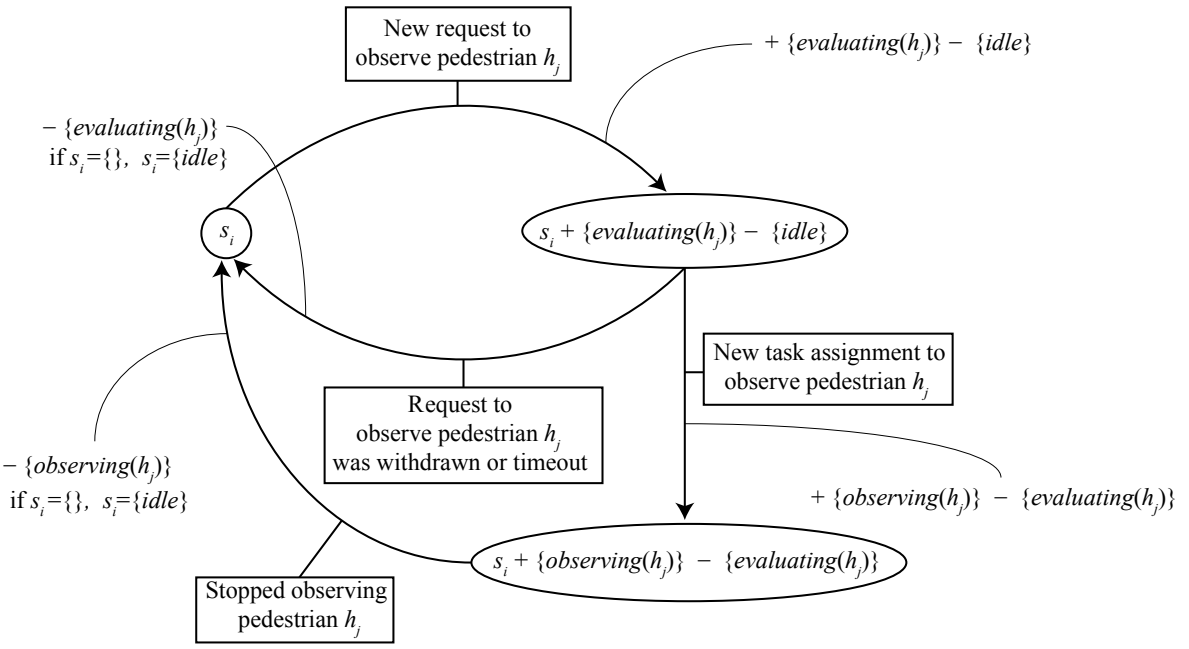


Figure 3.3: The state machine used by our Camera Controller. States are shown in ovals and transitions are shown in boxes. Additions and subtractions are shown next to each transition arc. *Courtesy of [29].*

Algorithm 2 Updating Activity Set

Require: The current activity set A.
Ensure: The updated activity set A.
1: **if** camera c_i is not engaged in activities at the moment **then return** $A = \Phi$
2: **end if**
3: **for all** $h \in H_i$ **do**
4: **if** Camera is tracking h **then**
5: $A = A \bigcup \{Observing(h)\}$
6: **end if**
7: **if** Camera is evaluating its suitability for tracking h **then**
8: $A = A \bigcup \{Evaluating(h)\}$
9: **end if**
10: **end for**

dictated by the tasks assigned to the camera. Other caveats are: 1) a camera cannot be both $Observing$ and $Evaluating$ the same individual and 2) an idle camera cannot be engaged in $Observing$ or $Evaluating$ any individual. Typically these tasks are assigned automatically through camera handoff negotiations described in the next section. These tasks can also be assigned by an operator manually or through heuristics of the form: track every individual that enters a specific region.

3.2 Memory Model

In contrast to existing models of smart camera nodes, we explicitly model short-term memory of a camera node. Each camera c_i uses the short-term memory to store its own state (the list of pedestrians in the near past plus the list of currently active tasks, i.e., H_i and A_i) and the state of its neighbouring camera nodes. Each camera manages its memory to keep it up-to-date. Stale items present in the memory are automatically removed. Presently, we assume that each camera has unbounded memory. Furthermore, we employ time-stamps to decide when to remove an item from the memory. Below we discuss how a camera manages the states of its neighbouring cameras. Say C_i represents the neighbours of c_i then c_i will store the current state s_j for each camera $c_j \in C_i$. Additionally, c_i will also keep track of previous states of

camera c_j. Remembered states are time-stamped and older states are automatically forgotten after some time.

Lets consider the following example to illustrate the memory model. Consider a camera c_2 that is currently tracking pedestrians h_2 and h_3. c_2 is also evaluating its suitability for tracking pedestrian h_4. Previously, c_2 was tracking h_1; however, h_1 is no longer tracked by c_2. In this scenario a neighbouring camera c_1 will store the following information about c_2 in its memory:

- $Observing(c_2, h_2, t)$;
- $Observing(c_2, h_3, t)$;
- $Evaluating(c_2, h_4, t)$; and
- $Know(c_2, h_1, t^-)$.

Notice that each memory item is time-stamped. $Know(c_2, h_1, t^-)$ indicates that camera c_1 believes that c_2 has seen the pedestrian h_1 at some previous time t^-.

Cameras share states voluntarily during negotiation messages or via a periodic state update message. In the above scenario, camera c_2 will periodically send H_2 and A_2 to the neighbouring camera c_1. c_1 breaks down the received state into memory items and compares these with the current c_2 items stored in its memory. One of the following four things can happen:

- if an item is already found in the memory, its time-stamp is updated;
- if an Observing item is received that matches with a stored Evaluating item, the Evaluating item is replaced with the Observing item;
- if no item matches with an already stored Observing or Evaluating item, the Observing (or Evaluating) item is replaced by a Know item; or
- if a received item does not match with a stored item (using the criteria implicit in the last three cases), the received item is time-stamped and inserted into the memory.

The following example clarifies these rules. Say c_1 receives the following information from c_2 at time $t^+ > t$:

$$A_2 = \{Observing(h_2), Observing(h_4)\}$$

and

$$H_2 = \{h_1, h_2, h_3, h_4\},$$

where $ts(h_1) = t^-$, $ts(h_2) = t^+$, $ts(h_3) = t$ and $ts(h_4) = t^+$. c_1 will update its information about c_2 as follows:

- $Observing(c_2, h_2, t^+)$;
- $Know(c_2, h_3, t)$;
- $Observing(c_2, h_4, t^+)$; and
- $Know(c_2, h_1, t^-)$.

If $t^+ - t^- > t^{\text{forget}}$ then item $Know(c_2, h_1, t^-)$ will be removed from the memory of c_1. There are many schemes for selecting t^{forget}; however, for the results presented in this paper, we use a single value of t^{forget} for all items.

In conclusion, the items stored in the short-term memory of a camera take one of the following forms:

- $Observing(c_i, h_j, t)$;
- $Evaluating(c_i, h_j, t)$;
- $Know(c_i, h_j, t)$.

Here $c_i \in C$, $h_j \in H$ and t refers to the time-stamp of a particular item. Notice that this form allows a camera to store both its own state and the states of its neighbouring cameras. Secondly, it is relatively easy to write queries involving cameras and pedestrians on the data stored in the short-term memory of a camera, such as (1) pick a neighbouring camera that is currently

observing pedestrian h_j, (2) how many neighbouring cameras are observing a particular pedestrian, (3) is their some neighbouring camera that knows about a particular pedestrians, etc. When considering the memory model discussed here, it is worthwhile to remember two things: (1) each camera only maintains the memory state of its first-hop neighbours, i.e., cameras that share an edge in the communication graph, and (2) we do not impose any guarantees that the state of the neighbouring camera is perfectly synchronized. The second item is of particular importance. The communication overhead to guarantee that each camera maintains a "perfectly synchronized copy of the state of its neighbours" is explosive and therefore unattainable in a real scenario.

Others, including [12], have developed techniques for generating vision and communication graphs within the context of camera networks. A side-effect of explicitly modelling the state of neighbouring cameras is that both communication and vision graphs are automatically learned. The goal is not to learn the overlaying communication graph or vision graph for the whole network. Rather, our goal is to learn a local snapshot at each node. For stationary nodes, we can assume the snapshot to be static; whereas, for non-stationary nodes, it is best to rely on the forgetting mechanism built into the memory model to maintain a time-varying snapshot.

3.3 Negotiations

Inter-camera negotiations allow observation tasks to be handed off from one camera to another as a person moves around an observation area. Generally speaking, however, a camera can initiate a negotiation whenever it wants another camera to take over an observation task:

- Typically a camera initiates a negotiation when it detects that the pedestrian in question is about to leave its field-of-view; or

- A camera can also initiate a negotiation when it wants to free up resources for a task requested by some other camera.

Existing negotiation schemes for camera handoff do not start negotiations when a camera wants to release resources that are currently being used by an observation task.

As stated earlier, a key difference between the current work and existing negotiation schemes for camera handoff is that here each camera maintains the state of neighbouring cameras. This allows a camera to be selective when considering neighbouring cameras for handoff. Consider the following scenario: camera 1 has the following information in its memory:

- $Observing(c_2, h_2, t^+)$;
- $Know(c_2, h_3, t)$;
- $Observing(c_2, h_4, t^+)$;
- $Know(c_2, h_1, t^-)$; and
- $Observing(c_3, h_6, t^+)$.

Since camera c_3 is only observing a single pedestrian h_6, camera c_1 first requests a handoff with c_3. If that fails, it can request c_2 to take over the task. Along similar lines, in a calibrated camera network, where each camera not only knows the state of neighbouring cameras, but also their observation constraints, cameras may be able to predict the outcome of negotiations without actually exchanging any messages.

3.3.1 Conditional Offers

Camera nodes are not modelled as *self-interested* agents and are always ready to takeover an observation task from a neighbouring camera. Still there are situations where a camera cannot accept a task, even if it can meet all task requirements. A camera node has a limited set of resources and is able to track at most o pedestrians at any given time. If a camera node is at this tracking limit, it cannot accept any new tasks. One potential solution to this problem is conditional offers. Conditional offers give camera nodes the ability to swap tasks or to terminate existing tasks in order to take on new, more pressing tasks.

Table 3.1: Events handled by the camera controller

Message Events	ReceivedRequestEvent
	ReceivedOfferEvent
	ReceivedResponseEvent
	ReceivedStatusUpdateEvent
Activity Manager Events	PedestrianLeavingEvent
	ObjectDetectedEvent
	EvaluateSuccessfulEvent
	EvaluateFailedEvent
Controller Events	EndAuctionEvent
	CleanupRequestsEvent
	SendStatusUpdateEvent
	UpdateMemoryEvent

3.4 Implementation Details

The camera controller is implemented using an event-based architecture. An event-based architecture makes it easy to manage all the things a camera node needs to do at any given time. There are three types of events that a camera node responds to. The first type are events triggered when a camera node receives a message from another camera. The second type are events triggered by the activity manager when a pedestrian is detected or an activity is completed. Finally, the third group are events triggered by the camera controller itself. This includes events set up to periodically update a camera nodes memory, send out status updates, and end auctions. The events that we currently use are listed in Table 3.1.

When a camera sends or receives a request to take over a task, it needs to keep track of the data associated with the request. This is achieved by the use of a Request Structure which is made up of the elements shown in Table 3.2. A Request Structure is created and added to a list of active requests when a camera receives an auction initiation message from another camera or when it decides to start its own auction. Each request has a unique identifier composed of the id of the camera initiating the request and the id of the pedestrian it involves. The Request Structure differs depending on if a request was sent by a camera or received from another camera. A sent request needs to keep track of the number of cameras notified of the request

Table 3.2: Data associated with a Request

	rid	unique request identifier
	src	id of source camera
	dst	id or ids of destination cameras
	hid	id of the pedestrian associated with the request
	$expiry$	the time at which the request expires and should be removed
	$parent$	the id of any parent request
Sent Request	$offers$	list of received offers
	num_sent	number of camera nodes notified about the auction
Received Request	$value$	the value of the offer sent back
	$condition$	the condition attached to the offer

and the offers/bids that have been received. A received request needs to keep track of the value and condition of any offer/bid that is sent back. The information stored in the Request Structure is used when responding to many of the events that can occur.

3.4.1 Message Events

Message events are triggered when a camera node receives a message from another camera. There is an event for every type of message that a camera can receive. The message events that we currently use are explained below.

Received Request Event

The received request event is triggered when a camera node receives a $MsgNewRequest$ message. This event has three parameters: rid, hid and sid. The rid is a unique identifier for the request, hid is a unique pedestrian identifier or descriptor and the sid is the id of the sending camera. Upon receiving this event, the camera node first checks to make sure the specified rid and hid are not already part of an active request. The camera node then begins an evaluate activity for the specified pedestrian. Lastly, a request object is created to keep track of all the information associated with a request and is added to a list of active requests.

Received Offer Event

The received offer event is triggered when a camera receives a $MsgOffer$ message. This event has four parameters: rid, $value$, $condition$ and sid. The $value$ parameter is the bid a camera is submitting. For our tests, it is the number of pixels enclosed by the bounding box of the pedestrian in a cameras image. The $condition$ is either a pedestrian identifier or Nil. Upon receiving this event, the number of received offers for the specified request is incremented and the offer is added to the list of offers. If every camera node that was notified replies back, the auction is ended prematurely and a winner is chosen.

Received Response Event

When a camera receives a response for an offer that was made, the received response event is triggered. This event has three parameters: rid, $response$, and sid. The response received is a Boolean value which is True if the camera won the auction and False otherwise. If a camera won the auction, the observation time for the associated task is extended and if the offer had a condition, the task associated with the condition is ended. The request object is also removed from the active request list. Finally, the camera's memory is updated and a status update is sent out to notify the other cameras of the change.

Received Status Update Event

The ReceivedStatusUpdateEvent is triggered when a camera receives a $MsgStatus$ message. This event has two parameters: $entries$ and sid. The $entries$ parameter is a list of entries that will be added to a cameras memory.

3.4.2 Activity Manager Events

Activity Manager Events are events triggered by the Activity Manager when it needs to notify the Camera Controller of a change at the activity layer. These events are triggered when activ-

ities complete successfully or unsuccessfully as well as when new objects are detected by the camera.

Pedestrian Leaving Event

The pedestrian leaving event is triggered when a camera notices that a person is about to leave a camera's viewing region. Upon receiving this event, the camera controller first checks to see if another camera is already observing the pedestrian. If the pedestrian is already observed by another camera, no further action is taken. If the person is not observed by another camera, the controller prepares to send out an auction initiation notification. The camera first gets a list of its neighbours which is constantly updated based on the communication graph. Next, the controller generates an rid for the request. After computing the expiry time based on the $kAuctionDuration$ constant, a request object is created to keep track of data associated with a request. The request object is then added to the list of active requests, and the auction initiation is then sent out to the neighbouring cameras. The final step enqueues an $EndAuctionEvent$ which will be triggered after the $kAuctionDuration$ time passes.

Evaluate Successful Event

When a camera receives an auction initiation request from another camera, it begins evaluating its suitability to observe the requested pedestrian. This evaluation can either end successfully, meaning the person has been found and all of the constraints can be met, or otherwise unsuccessfully. Upon receiving an $EvaluateSuccessfulEvent$, the camera controller must decide if it wants to submit an offer for the request. This decision is based on whether or not a camera is able to add a new task based on the resources that are available. If it is able to add a new task, an offer is sent to the requesting camera and a temporary observation task is initiated for the associated pedestrian. If the camera is not able to allocate enough resources for the task, it looks for a task that could potentially be swapped with the requesting camera or handed off to another camera. Handing off to a third camera requires a Child Request to be sent. Currently

a Child Request is only attempted once. If a Child Request is unsuccessful or handing off a pedestrian to a third camera still does not free up enough resources an offer with a -1 value is sent.

Algorithm 3 Handling request after a successful evaluation

Require: rid ▷ The request identifier of an active request
1: $request \leftarrow$ The Request Structure for the request with id rid
2: **if** Can add a new task **then**
3: Send an offer to requesting camera
4: Begin to temporarily observe the associated pedestrian
5: **else if** Handoff attempted by the use of a Child Request **then**
6: Send an offer with a -1 value to the requesting camera
7: **else**
8: Try finding a handoff candidate
9: **end if**

To find possible handoff candidates, a camera's memory is queried to find other cameras that know about the pedestrians observed by a camera. In our implementation, this results in a dictionary that maps the $hids$ of the pedestrians currently being observed by the camera to lists of camera ids that know of the pedestrian with the corresponding hid. If this dictionary is empty, it means that no handoffs are possible, and therefore an offer with a -1 value is sent. If the dictionary is not empty, a candidate hid and camera id are chosen. The candidate camera can either be the camera making the original handoff request or a third camera that needs to be added to the negotiation. If the candidate is the requesting camera, a conditional offer is sent and the requested pedestrian is temporarily observed. Otherwise, a new Child Request is created asking a third camera to take over a task.

The function for picking the best handoff candidate takes in a dictionary that maps $hids$ of observed pedestrians to lists of cameras that know about the person. It then scores the pedestrians based on how long they have been observed and their location in the cameras viewing region. Pedestrians who have been observed longest and are most likely to leave are scored higher. The winning candidate is the pedestrian with the highest combined score. Currently the camera chosen is the first camera in the list of cameras that know the winning

Algorithm 4 Handing off a task to free up resources

Require: $camID$ ▷ ID of the camera initiating handoff
Find possible handoff candidates
if No candidates found **then**
    Send an offer with a -1 value
    return
end if
Pick the best handoff candidate
if candidate camera == $camID$ **then** ▷ Attempt a swap
    Begin to temporarily observe the pedestrian associated with the request
    Send the requesting camera an offer with the candidate pedestrian as the condition
else ▷ Add a third camera to the negotiation
    Generate a new request structure for the child request
    Add the request to the list of active requests
    Send an auction initiation message to the candidate camera
    Enqueue an $EndAuctionEvent$ that will be triggered after $kAuctionDuration$
end if

Algorithm 5 Finding Handoff Candidates

Require: $candidates$ ▷ An empty dictionary
Require: $observeList$ ▷ A list of pedestrian IDs the camera is observing
Ensure: $candidates$ maps observed pedestrians to other cameras that know the pedestrian.
for hid **in** $observeList$ **do**
    $cams \leftarrow$ A list of camera IDs that know the pedestrian identified by hid
    if $cams$ is empty **then**
        continue
    end if
    $activity \leftarrow$ The activity associated with hid
    if not $activity$ **then**
        continue
    end if
    $startTime \leftarrow$ Start time of activity
    $position \leftarrow$ Position relative to the center of the cameras viewing region
    $candidates[hid] \leftarrow (startTime, position, cams)$
end for

pedestrian. This could potentially be expanded further to favour the requesting camera or the camera with the most available resources.

Algorithm 6 Picking The Best Handoff Candidate

Require: $candidates$ ▷ A list of handoff candidates

$scores \leftarrow$ Empty Dictionary
$sortedByStartTime \leftarrow$ The $candidates$ sorted by start time
$time \leftarrow$ The current time
$score \leftarrow 0$
for $entry$ **in** $sortedByStartTime$ **do**
    if $entry.startTime < time$ **then**
        $time = entry.startTime$
        $score = score + 2$
    end if
    $scores[entry.hid] = score$
end for
$sortedByPosition \leftarrow$ The $candidates$ sorted by position
$pos \leftarrow 0.0$
$score \leftarrow 0$
for $entry$ **in** $sortedByPosition$ **do**
    if $entry.position > pos$ **then**
        $pos \leftarrow entry.position$
        $score = score + 1$
    end if
    $scores[entry.hid] = scores[entry.hid] + score$
end for
$hid \leftarrow$ The hid of the candidate with the highest score
$cid \leftarrow$ The id of the first camera in the list of cameras that know the candidate
return (hid, cid)

Evaluate Failed Event

The EvaluateFailedEvent is triggered when a camera cannot find a pedestrian or is not able to meet all of the constraints. In such a case, an offer with a -1 value is sent back to the requesting camera and the request is removed from the active requests list.

Object Detected Event

The ObjectDetectedEvent is triggered when a camera detects a pedestrian that it is not observing. Upon receiving this event, a camera needs to decide if it wants to observe the pedestrian. In situations with a large number of pedestrians, automatically observing all pedestrians may not be the best use of resources. If a camera is configured to automatically observe all newly detected pedestrians, it must first check to see if it has the resources available before beginning to observe the pedestrian.

3.4.3 Controller Events

Controller Events are used by the controller to schedule a task at some point in the future. Such events are used to end auctions, send out status updates and clean up outdated requests. They can be one time events or events that are repeated periodically.

End Auction Event

The EndAuctionEvent is triggered when the allotted time for an auction is expired or when an offer was received from every camera that was notified of the auction. A camera picks a winner from the list of offers it has received using Algorithm 8. The offer with the highest value is chosen; however, priority is given to offers without a condition. Once a winner is chosen, the camera then notifies the winner of the result by sending a $MsgResponse$ message. It then ends the activity that was handed off and if there was a condition, it begins a new observe activity for the conditional pedestrian. If the auction was part of a Child Request, the parent request is revisited. Finally, the memory is updated and a status message is sent out to notify the neighbouring cameras of the result. The request is also removed from the list of active requests.

Algorithm 7 End Auction Event Handler

Require: rid ▷ The request identifier for an active request
1: $request \leftarrow$ The Request Structure for the request with id rid
2: $hid \leftarrow request.hid$
3: $winner \leftarrow$ The winner chosen from $request.offers$
4: **if** $winner$ **then**
5: Stop observing the pedestrian identified by hid
6: Send a response to the winning camera
7: **if** $winner.condition$ **then**
8: Begin Observing the conditional pedestrian
9: **end if**
10: **if** $request.parent$ **then**
11: Re-evaluate parent request
12: **end if**
13: Update Memory
14: Send out a status update to neighbouring cameras
15: **end if**
16: Remove the request with ID rid from the list of active requests

Algorithm 8 Picking The Winner

Require: $offers$ ▷ A list of offers
Ensure: cam Is the ID of the winning camera
Ensure: $condition$ Is a pedestrian identifier the auctioning camera agrees to observe
Ensure: $value$ Is the value of the winning bid
1: $value \leftarrow 0$
2: $cam \leftarrow$ Nil
3: $condition \leftarrow$ Nil
4: **for** $offer$ **in** $offers$ **do**
5: **if** $offer.value > value$ and not $offer.condition$ **then**
6: $value \leftarrow offer.value$
7: $cam \leftarrow offer.cam$
8: **end if**
9: **end for**
10: **if** not cam **then**
11: **for** $offer$ **in** $offers$ **do**
12: **if** $offer.value > value$ and CANSEE($offer.condition$) **then**
13: $value \leftarrow offer.value$
14: $cam \leftarrow offer.cam$
15: $condition \leftarrow offer.condition$
16: **end if**
17: **end for**
18: **end if**

Table 3.3: Constants we used for our testing

Name	Value	Description
$kMaxObserve$	4	Maximum number of objects a camera can observe at any time
$kStatusFrq$	2.0	The frequency of status messages sent by each camera
$kMemUpdateFrq$	1.0	How frequently a cameras memory is updated with information about its local tasks
$kMemExpiryTime$	5.0	Time at which memory entries expire and are removed
$kAuctionDuration$	1.0	Amount of time a camera waits for bids before choosing a winner
$kRequestExpiry$	1.0	Time at which information about a request is removed from a cameras memory.

Update Memory Event

The UpdateMemoryEvent is used to periodically update a camera controllers memory with information about the current activities being carried out by the camera. Outdated information is also removed.

Send Status Event

The SendStatusEvent is used to periodically send status updates to neighbouring cameras. The status update includes the latest information about the observe and evaluate tasks a camera is carrying out as well as the pedestrians that it knows about.

3.4.4 Concurrency

Cameras may have multiple negotiations active at the same time. Special care must be taken to ensure a camera does not assign the same resources to two different tasks. For example, if a camera has enough resources to observe 4 pedestrians and it is already observing 3, it cannot submit offers in two different negotiations at the same time. This could result in the camera having to observe 5 pedestrians if it won both auctions. To prevent this from happening, a camera begins to temporarily observe a pedestrian before it submits an offer. This temporary observation task is treated no differently than any other observation task except for the fact that

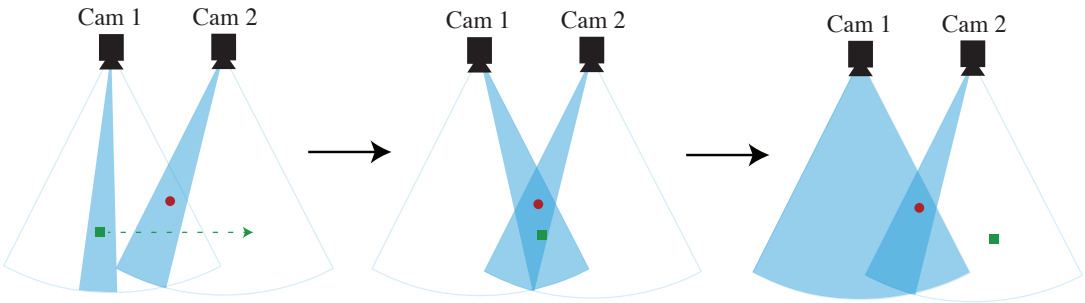
it expires after some period of time. If a camera has only one observation slot open, the first offer sent will put the camera at its observation limit. This will force other negotiations to send conditional offers or alternatively wait until a response is received.

Another issue that could arise with multiple negotiations occurring at the same time is a pedestrian being used as a condition in two different negotiations. This too could result in a camera exceeding its observation limit. This issue is addressed by not allowing a pedestrian to be a handoff candidate if it is a condition in an active negotiation.

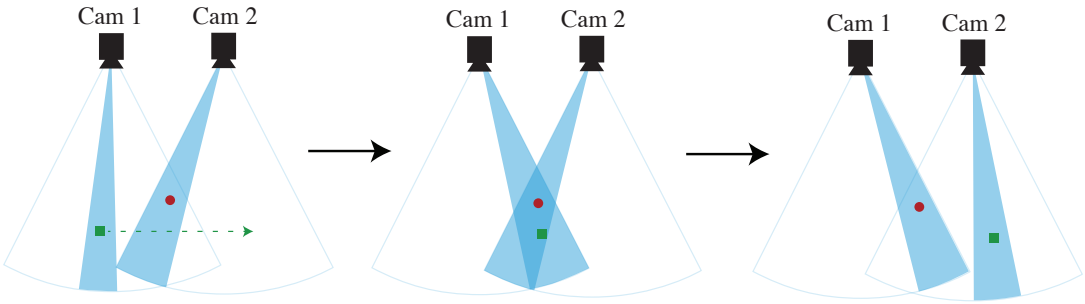
Chapter 4

Results

We simulated camera networks comprising 1, 2, 4, and 16 cameras observing up to 32 pedestrians in our 2D camera network simulator to evaluate the proposed approach. We also compared our method with the scheme proposed by Esterle et al. [12], which is a recent state-of-the-art technique for distributed camera handoff. Esterle et al. approach exhibits many of the characteristics of existing distributed handoff schemes: 1) there is no provision for counter-offers and 2) cameras ignore the state of neighbouring cameras. We implemented the messaging protocol in Esterle et al. [12] within our 2D simulator (see Appendix B) to compare the two techniques. The fact that we were able to simulate two different control strategies for camera networks within our simulator supports our observation that it is much easier and more cost effective to use simulated environments for studying camera networks, at least at the early stages. Our camera networks assume "perfect" pedestrian detection, tracking, and identification. This is indeed a simplifying assumption, however, the recent advances in pedestrian tracking suggests that our assumptions hold for low to medium density crowds. Furthermore, we believe that these assumptions do not diminish the camera handoff strategy proposed here.



(a) Without memory and conditional offers



(b) With memory and conditional offers

Figure 4.1: An overview of scenario 1 with and without the use of memory. Without memory, the pedestrian represented by the green square is no longer observed once it leaves camera 1's viewing region. With the use of memory, as swap can occur, resulting in the pedestrian represented by a green square being seamlessly handed over to camera 2.

4.1 Test Scenarios

We evaluate our approach on six different scenarios. Scenario 1 demonstrates the effects of resource limitations on performing camera handoffs. Resource limitations encode the fact that a camera may not be able to observe every individual that is present in its field-of-view. Scenario 2 showcases a camera network with blind regions. Blind regions are not covered by any camera in the camera network. Scenarios 3, 4 and 5 compare our method to the Esterle et al. scheme [12]. Scenario 6 simulates a camera network comprising 16 PTZ cameras observing a region with up to 32 pedestrians at any given time. This scenario is designed to study the scalability properties of our approach.

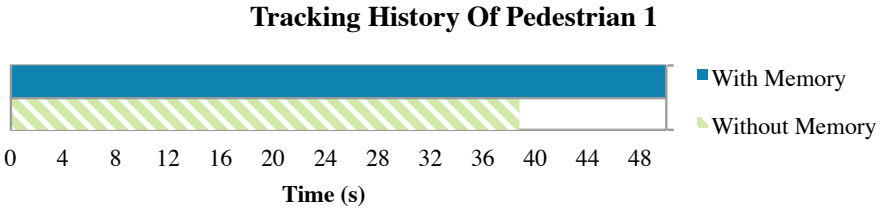


Figure 4.2: Scenario 1: The tracking history of pedestrian 1 with and without the use of a camera's memory for storing the internal state of the neighbouring cameras. When using memory, a task swap can occur and pedestrian 1 is seamlessly tracked throughout his stay in the region.

4.1.1 Scenario 1

In the first scenario, we demonstrate the effect our memory model and the use of conditional offers has on handoffs between cameras that are at their observation limits. Cameras at their observation limits are already observing the maximum number of targets that their resources allow. This means that the only way to complete a handoff is for two cameras to swap tasks. This scenario is made up of two cameras and two pedestrians as shown in Figure 4.1. The two cameras are aligned in a row looking in the same direction and each camera is restricted to only observe 1 pedestrian at a time. There are two pedestrians in the scene. One of the pedestrians—which is initially observed by camera 2—is stationary, while the other pedestrian—initially observed by camera 1—is moving across the region towards camera 2.

We performed two tests on this scenario. In the first test we do not use our proposed memory model and conditional offers, while in the second test we do. As Figure 4.1 (a) shows, in the absence of memory model and conditional offers, a handoff cannot occur and the pedestrian originally observed by camera 1 will no longer be observed once it leaves camera 1's viewing region. By using our memory model, camera 2 is able to respond to the auction initiated by camera 1 with a conditional offer asking camera 1 to swap tasks. Camera 1 accepts the conditional offer and the two cameras swap tasks. This is shown in Figure 4.1 (b). We plot the tracking history for pedestrian 1 from both tests in Figure 4.2. Here you can clearly see that without the use of our memory model, pedestrian 1 is no longer tracked after about 40 seconds.

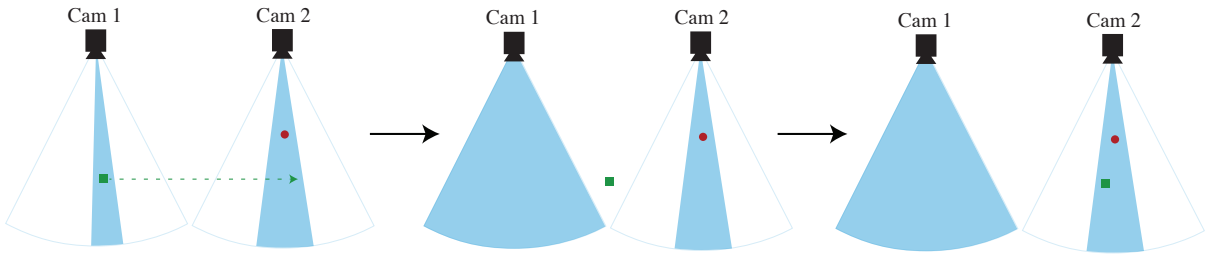


Figure 4.3: An overview of scenario 2. Here cameras do not have overlapping viewing regions. This results in pedestrians being not observed for a period of time; however, the cameras are able to recover once the pedestrian re-enters a cameras viewing region.

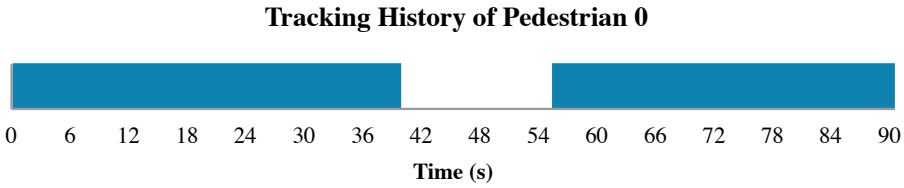


Figure 4.4: Scenario 2: The tracking history of pedestrian 0. Since the two cameras do not have overlapping viewing regions, the pedestrian does not get tracked for a period of time.

By using our memory model, the two cameras can initiate a swap, which results in pedestrian 1 being seamlessly transferred to camera 2 so that it can continue to be observed.

4.1.2 Scenario 2

In Scenario 2, we show how our approach works in cases where cameras do not have overlapping field-of-views. This scenario also consists of two cameras; however, this time the camera's do not have overlapping field-of-views and are allowed to observe more than 1 target. An overview of the scenario is shown in Figure 4.3. The scenario begins with both cameras observing a single pedestrian. Like in scenario 1, the pedestrian observed by camera 2 is stationary, while the pedestrian observed by camera 1 is moving across the room. After some time, the pedestrian that is moving leaves the first camera's field-of-view and enters the region not observed by any cameras. As it continues to move, the pedestrian enters camera 2's viewing region and is automatically detected by the camera.

Figure 4.4 plots the tracking history of pedestrian 1 as she moves between the observational

Table 4.1: Scenario 3 Parameters

Test Number	**1**	**2**	**3**	**4**	**5**	**6**
Number of pedestrians	1	2	4	6	8	10
Number of cameras	2					
Number of runs	30					
Simulation time	70 seconds					

ranges of camera 1 and camera 2. The first block represents the time the pedestrian is observed by camera 1, while the second block represents the time it is observed by camera 2. Since there is a region between cameras 1 and 2 that is not observed by any camera, this is the best that the cameras were able to do.

4.1.3 Scenario 3

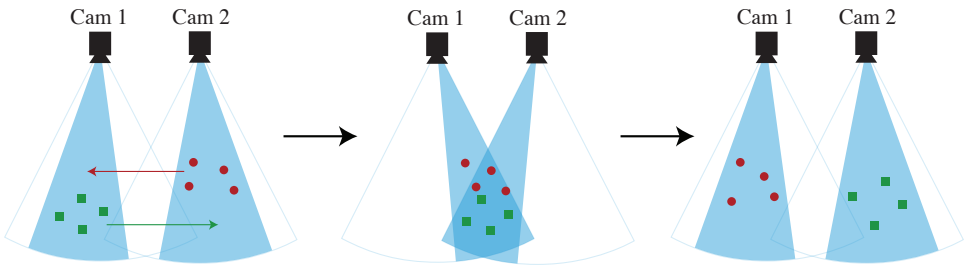


Figure 4.5: An overview of Scenario 3. The two cameras begin by observing a group of pedestrians walking towards the other cameras viewing region. After some time, both groups of pedestrians reach the edge of their respective cameras viewing region requiring multiple handoffs so that the pedestrians can be observed as they continue walking.

The third scenario is made up of two cameras with overlapping field-of-views just like scenario 1. For this scenario, we run 6 tests with 1, 2, 4, 6, 8 and 10 pedestrians. Other than the first test, all tests begin with an equal number of pedestrians present in a cameras field-of-view. The pedestrians walk toward each other resulting in a handoff situation when the two groups meet in the area where the viewing regions of the two cameras intersect. An overview of the test with 8 pedestrians is shown in Figure 4.5. The cameras in this scenario are restricted to observing at most 4 pedestrians at any given time. As a result, a swap of tasks is necessary to complete a successful handoff. For this scenario we run the same tests on both our approach

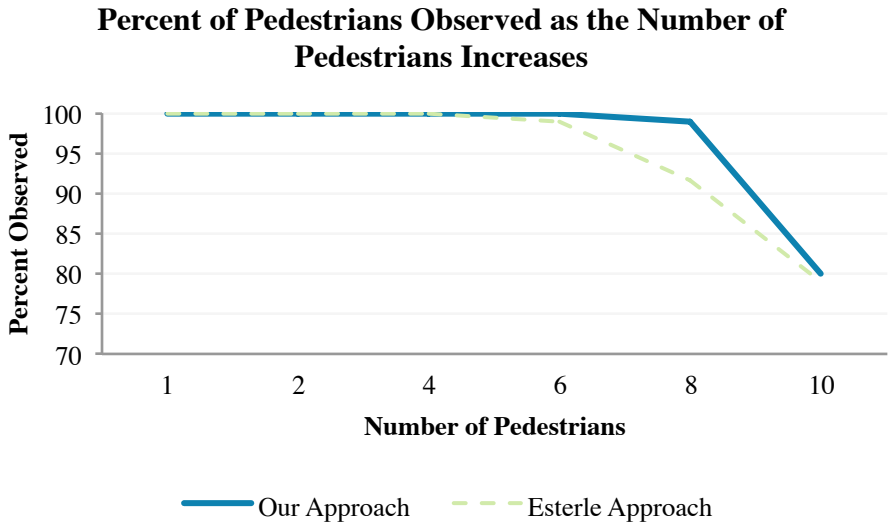


Figure 4.6: Scenario 3: The percent of pedestrians observed by our approach and that of Esterle et al. [12] as the number of pedestrians increases. The results are averaged over 30 runs.

and the approach by Esterle et al. [12]. We run each test 30 times for each approach and average the results.

The percent of observed pedestrians as the number of pedestrians increases in plotted in Figure 4.6. The percent observed for each test is computed using equation 4.1 with results averaged over the 30 runs. Here TS is the total number of time steps, P_{tot} is the total number of pedestrians present in the scene and P^i_{obs} is the number of pedestrians observed at time step i.

$$PercentObserved = \frac{100}{TS \times P_{tot}} \sum_{i=1}^{TS} P^i_{obs} \tag{4.1}$$

Figure 4.6 shows that our approach performed slightly better than the Esterle et al. approach once there are around 6 people in the scene. Figure 4.7 shows how smooth our approach is compared to the Esterle approach. Our approach is able to complete seamless handoffs, even when the cameras are tracking the max number of people. This is due to the memory module allowing cameras to swap tasks when cameras are at their tracking limits. The Esterle et al.'s

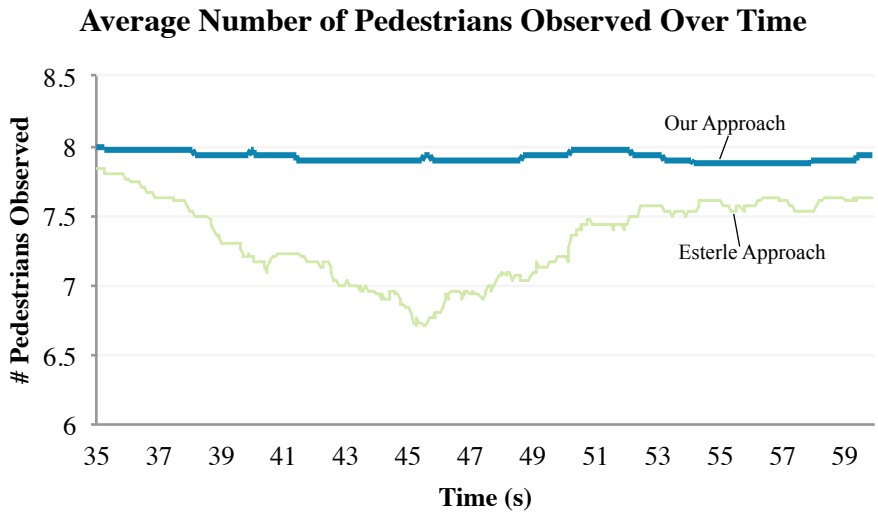


Figure 4.7: Scenario 3: Average number of pedestrians observed at each time step with 8 people present.

Table 4.2: Message counts for scenario 3

Message Type	**Our Approach**						**Esterle et al. Approach**					
	1	**2**	**4**	**6**	**8**	**10**	**1**	**2**	**4**	**6**	**8**	**10**
$MsgAuction$	1	2	4	5	7	5	1	2	4	43	117	159
$MsgOffer$	1	2	4	5	6	5	1	2	4	5	4	2
$MsgResponse$	1	2	4	5	4	3	1	2	4	5	4	2
$MsgStatus$	70	72	76	77	77	75	0	0	0	0	0	0
Total	73	78	88	92	94	88	3	6	12	53	125	163

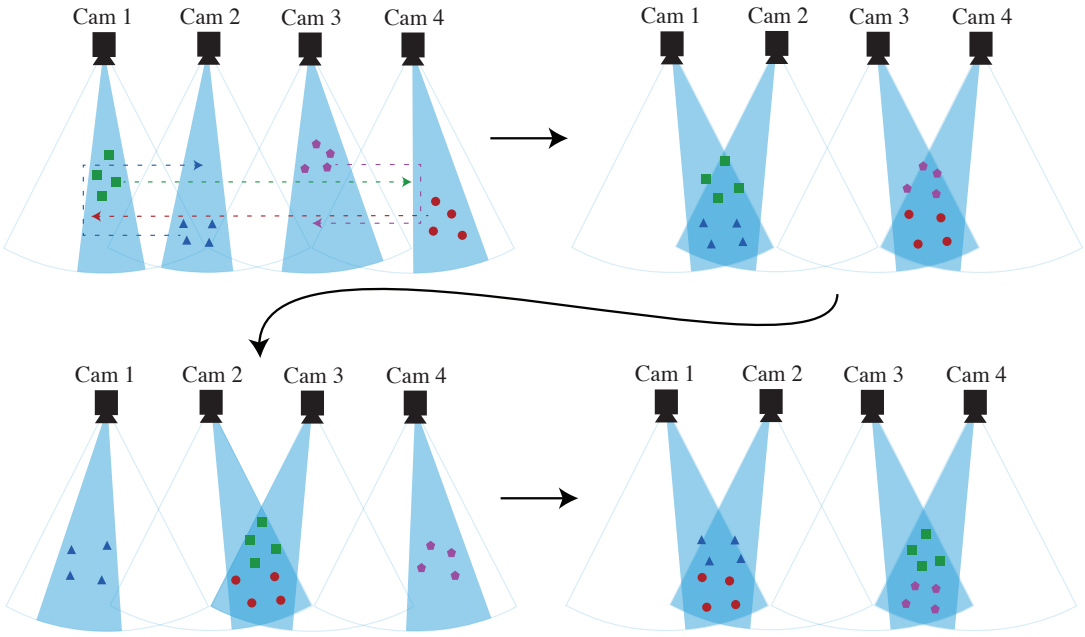


Figure 4.8: An overview of Scenario 4.

approach has to wait until a pedestrian leaves a cameras viewing region before being able to accept a new task. This results in a short period of time where a pedestrian will not be tracked. This is more evident in scenario 5.

The average number of messages sent for each test are shown in Table 4.2. In our approach, the majority of the message sent are status messages. The approach by Esterle et al. does not use status messages; however, multiple auction initiation messages are sent if a pedestrian is not successfully auctioned off. This results in a larger number of messages sent once cameras are at there observation limit. For our tests, status messages where sent every 2 seconds and every time a handoff was completed. Auction notifications were sent every 1 second since that was the duration of the auction.

4.1.4 Scenario 4

The fourth scenario is made up of four cameras lined up in a row with overlapping field-of-views as shown in Figure 4.8. For this scenario, we run tests with 2, 4, 8, 12 and 16 pedestrians. The pedestrians start in four groups, one group per camera. The group that starts in camera

Table 4.3: Scenario 4 Parameters

Test Number	**1**	**2**	**3**	**4**	**5**
Number of pedestrians	2	4	8	12	16
Number of cameras	4				
Number of runs	30				
Simulation time	120 seconds				

Table 4.4: Message counts for scenario 4

Message Type	**Our Approach**					**Esterle et al. Approach**				
	2	**4**	**8**	**12**	**16**	**2**	**4**	**8**	**12**	**16**
$MsgAuction$	7	13	21	34	43	6	13	64	116	248
$MsgOffer$	6	12	20	31	41	6	12	63	114	247
$MsgResponse$	6	12	20	22	25	6	12	23	31	36
$MsgStatus$	374	393	417	424	433	0	0	0	0	0
Total	393	430	478	511	542	18	37	150	260	531

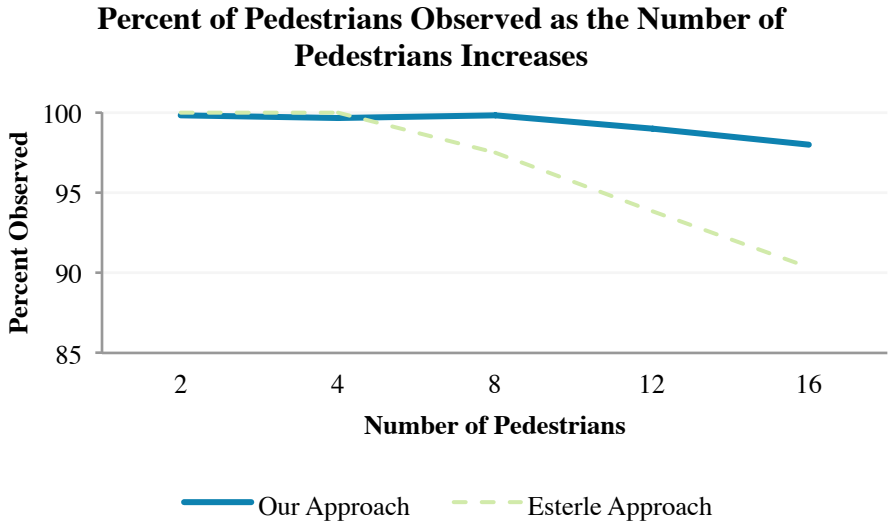


Figure 4.9: Scenario 4: The percent of pedestrians observed as the number of pedestrians increased. The results are averaged over 30 runs.

1's viewing region walks across to camera 4's viewing region, while the group that starts in camera 4's viewing region walks across towards camera 1. The pedestrians that start in camera 2's viewing region walk toward camera one before doing a u-turn back towards camera 2. Finally, the group that starts in camera 3's viewing region walks towards camera 4 and also does a u-turn back towards camera 3. This results in three different times when swap situations occur which is shown in Figure 4.8.

The cameras in this scenario are also restricted to observing at most 4 pedestrians at any given time. As a result, a swap in necessary to complete a successful handoff. We again ran the same set of tests on both our approach and the approach by Esterle et al for 120 seconds of simulation time.

The percent of observed pedestrians as the number of pedestrian increases is plotted in Figure 4.9 using the same equation as Scenario 3. Again, our approach does significantly better once the cameras are at their observation limits. Table 4.4 shows the message counts which exhibit the same pattern as those in Scenario 3.

4.1.5 Scenario 5

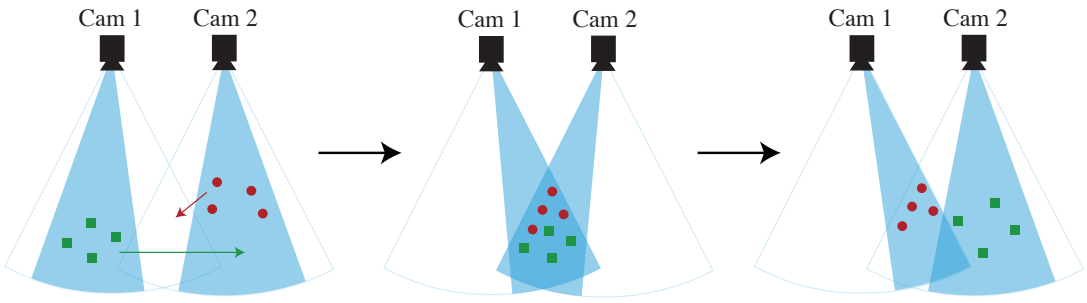


Figure 4.10: An overview of Scenario 5.

Scenario 5 is very similar to scenario 3; however, there is one key difference. The pedestrians viewed by camera 2 never leave the camera's viewing region. This combined with the fact that a camera is restricted to only being able to observe 4 pedestrians at any given time means a swap is necessary for a handoff to occur. Again we run tests with 1, 2, 4, 6, 8 and 10

Table 4.5: Scenario 5 Parameters

Test Number	**1**	**2**	**3**	**4**	**5**	**6**
Number of pedestrians	1	2	4	6	8	10
Number of cameras	2					
Number of runs	30					
Simulation time	70 seconds					

Table 4.6: Message counts for Scenario 5

Message Type	**Our Approach**						**Esterle et al. Approach**					
	1	**2**	**4**	**6**	**8**	**10**	**1**	**2**	**4**	**6**	**8**	**10**
$MsgAuction$	1	1	2	4	4	4	1	1	2	49	124	161
$MsgOffer$	1	1	2	3	4	4	1	1	2	2	2	2
$MsgResponse$	1	1	2	3	4	3	1	1	2	2	2	2
$MsgStatus$	70	70	72	74	76	74	0	0	0	0	0	0
Total	73	73	78	84	88	85	3	3	6	53	128	165

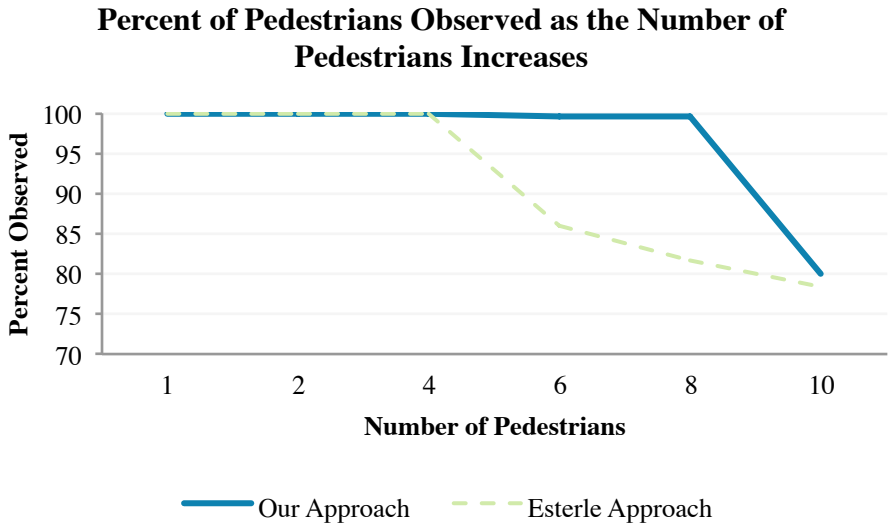


Figure 4.11: Scenario 5: The percent of pedestrians observed by our approach and that of [12] as the number of pedestrians are increased. The results are averaged over 30 runs.

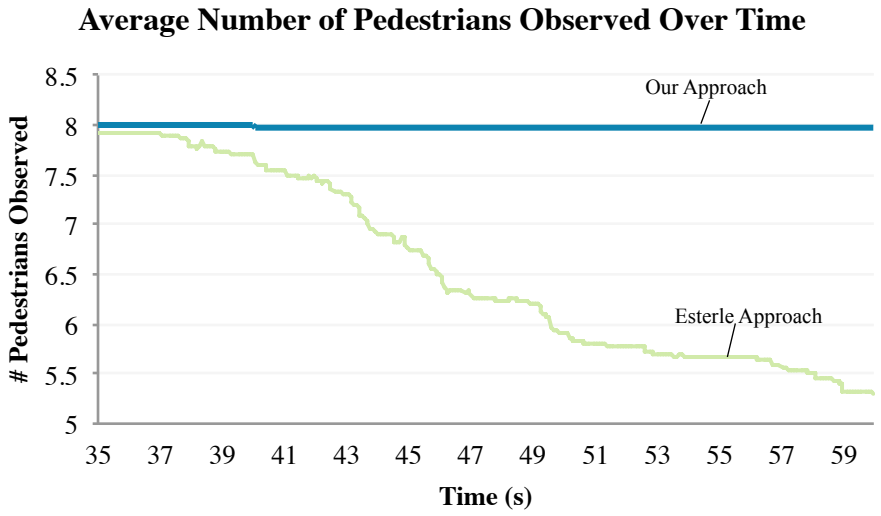


Figure 4.12: Scenario 5: Average number of pedestrians tracked with 8 people present.

pedestrians and 70 seconds of simulation time.

The percent of pedestrians observed as the number of pedestrians increases is shown in Figure 4.11. Our approach performs just as well as it did in Scenario 3; however, the approach by Esterle et al. performs significantly worse. This is due to the fact that the pedestrians observed by camera 2 no longer leave camera 2's viewing region. Scenario 3 allowed an indirect swap to occur—when using the approach by Esterle et al.—once both cameras dropped one or more of their pedestrians.

The average number of pedestrians tracked over time for test number 5 is shown in Figure 4.12. Notice how our approach consistently observes all 8 pedestrians compared to the approach by Esterle et al. The average message counts for this scenario are shown in Table 4.6. Again our approach averages around 80 messages for all six of the tests while the Approach by Esterle et al. requires almost double that when 10 pedestrians are in the scene.

4.1.6 Scenario 6

Scenario 6 consists of 32 pedestrians walking along a rectangular path. 16 cameras are tasked with observing these individuals (see Fig. 4.13). The pedestrians are tracked for 3000 simu-

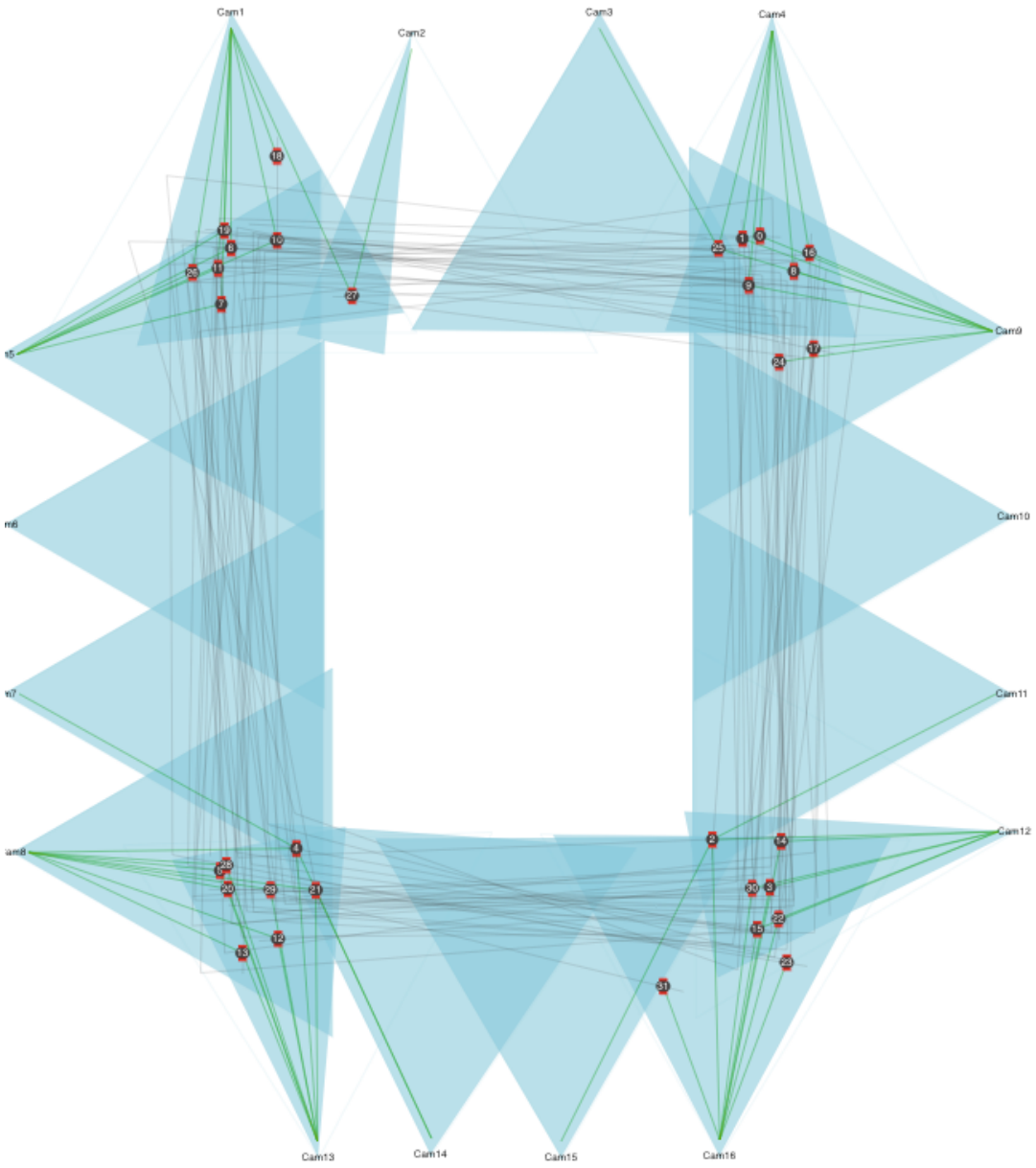


Figure 4.13: Scenario 6: 16 cameras are tasked to observe 32 pedestrians as these move along a rectangular path.

Table 4.7: Scenario 6

Test Number	**1**	**2**	**3**	**4**
Number of pedestrians	32			
Number of cameras	16			
Uses Memory	Yes	No	Yes	No
Max Observe	4	4	2	2
Number of runs	30			
Simulation time	100 seconds			

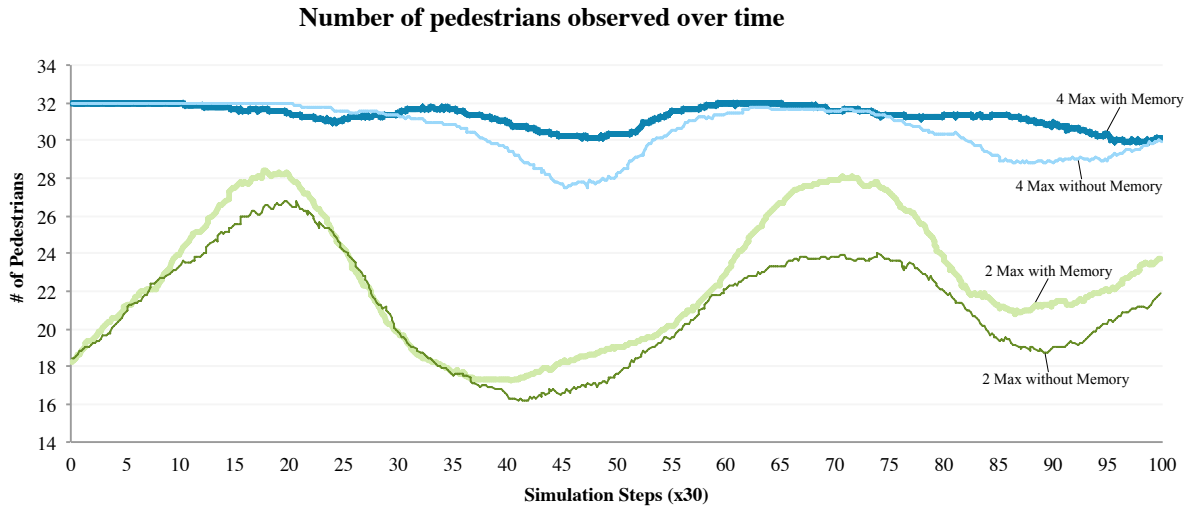


Figure 4.14: Scenario 6: The numbers of pedestrians observed over time. Data is averaged over 30 runs.

lation steps. Using this scenario, we have evaluated the proposed model under 4 settings: 1) each camera can track no more than 4 pedestrians, 2) each camera can track no more than 4 pedestrians and the cameras do not maintain the state of their neighbours, 3) each camera can track no more than 2 pedestrians and 4) each camera can track no more than 2 pedestrians and the cameras do not maintain the state of their neighbours. Fig. 4.14 plots the number of pedestrians tracked for each setting over time. On average 31.289 pedestrians are tracked when cameras can track up to 4 pedestrians. This value is reduced to 30.611 when cameras are not allowed to propose conditional offers (no memory). Similarly, on average the cameras track 22.557 pedestrians when each camera is allowed to observe only 2 pedestrians at any given time. This value is reduced to 21.129 when cameras are not allowed to propose conditional offers. While these results suggest that benefits of conditional offers are marginal, it is worth keeping in mind that conditional offers only effect situations where tasks swaps are possible. Perhaps, this scenario does not contain many such situations. In any case, scenario 1 clearly makes the case of conditional offers.

Chapter 5

Conclusion

In this thesis, we presented a new approach to the camera handoff problem. We present a new short-term memory model for maintaining a camera's own state and the states of neighbouring cameras. This memory model is used to introduce the concept of conditional offers during handoff negotiations. Conditional offers allow successful handoffs to occur in situations where existing approaches would fail. One such example is when a camera is at its tracking limit and cannot rack any more pedestrians. With the use of the memory model, cameras, are able to negotiate a swap of tasks if the pedestrians being tracked are visible in both cameras.

We have evaluated our approach on a variety of scenarios with 2, 4, and 16 cameras and up to 32 pedestrians. We compared our approach to the Esterle et al. scheme that appeared in [12] and the results seem to suggest that our approach outperforms the handoff approach developed in [12].

Although we focus on using the memory to send conditional offers during a handoff negotiation, there are other uses for the memory model. The memory model can be used to distribute tasks between multiple cameras as shown in Figure 5.1. Here, two cameras start out tracking the same group of pedestrians. Not only are the pedestrians tracked at a low resolution since the cameras have to be zoomed out to observe all of the people, but it also wastes resources since both cameras are doing the same work. These issues can be resolved by splitting the

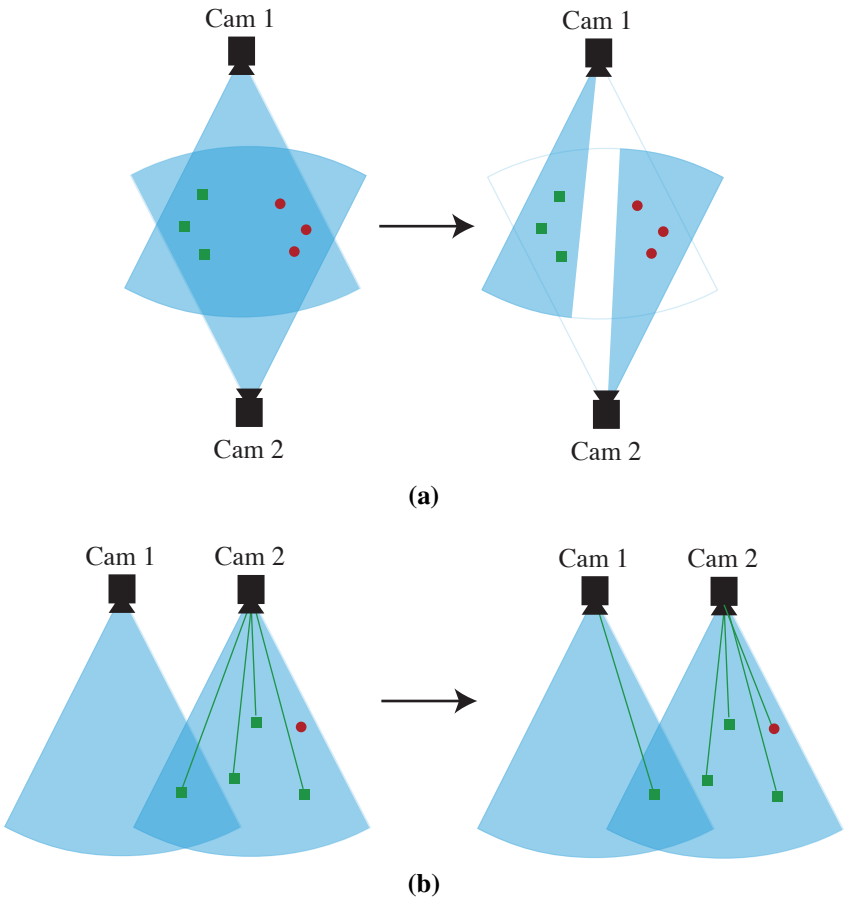


Figure 5.1: Other uses of memory. (a) distributing tasks between cameras to eliminate duplicate tasks. (b) Handing off one task to add another.

pedestrians between the two cameras.

Another potential case where memory can be used is shown in Figure 5.1. Here camera 2 is observing four pedestrians and is at its tracking limit. When a new pedestrian—represented as a red circle—comes into view, it is not able to observe it because it is at its tracking limit. Using its builtin memory model, a camera could handoff a pedestrian to camera 1 which would free up enough resources to observe the new pedestrians.

These are just a few examples of how a memory model can be used to improve the tracking performance of a group of cameras which we plan on exploring in the future. We also plan on evaluating the approach using real cameras.

Appendix A

Virtual Camera Model

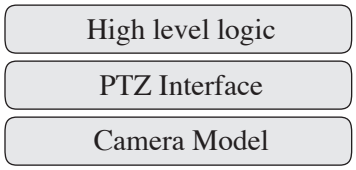


Figure A.1: The layered architecture of a virtual camera

Our Virtual Cameras are designed to mimic Smart Cameras. They use a layered architecture made up of the three layers shown in Figure A.1. At the base of our virtual camera architecture is the Camera Model layer which is a data structure for all of the information needed to define a camera. The next layer is the PTZ Camera layer which adds an interface that is typically found on a PTZ camera with options to pan, tilt and zoom. The final layer is where the camera control and coordination logic is implemented allowing the camera to complete high level tasks autonomously.

The layered architecture has many benefits. It is similar to a physical camera where the PTZ interface manipulates the lower level camera properties. It also makes it much easier to port our work to physical cameras. All we have to do is implement our PTZ Camera interface on top of a physical camera's PTZ interface and our smart camera node will be fully functional.

A.1 Camera Model

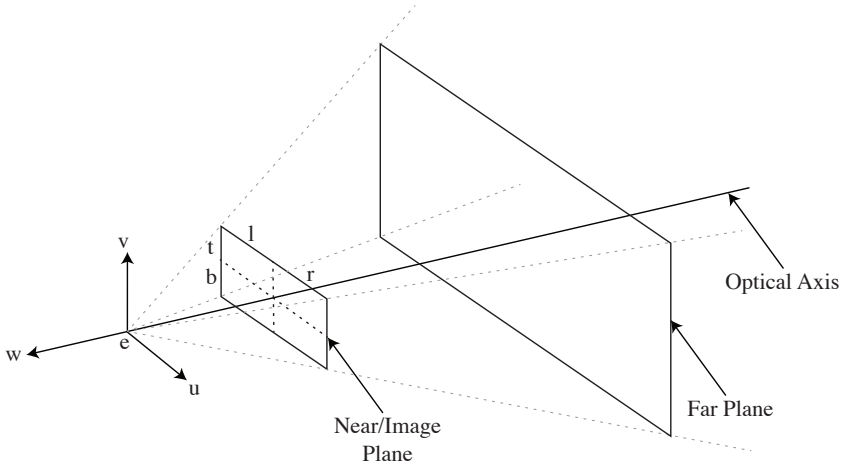


Figure A.2: Anatomy of a virtual camera.

The Camera Model is at the core of our virtual camera architecture. It is a data structure for storing all of the properties needed to define a camera and its lens. Some of the properties that make up the Camera Model are:

location The location of the camera in world coordinates represented by a 3D point (x, y, z)

default direction A 3D vector denoting the direction the camera is pointing at when in its default position. This is a reference direction with pan and tilt values set to zero.

current direction A 3D vector denoting the direction the camera is currently pointing at. It is computed by taking the default direction vector and applying the pan and tilt values to it.

pan angle The horizontal angle between the current direction and default direction vectors

tilt angle The vertical angle between the current direction and default direction vectors

field of view angle The angle from the bottom of the screen to the top of the screen.

up direction A vector perpendicular to the direction vector denoting the direction the top of the camera faces.

near plane The minimum distance an object must be away from the camera for it to be visible.

far plane The maximum distance an object can be away from the camera for it to be visible.

pan, tilt and zoom limits The minimum and maximum values for the pan, tilt and zoom angles.

All high level manipulations of the camera affect these variables which are then used to generate the viewing and projection matrices. The viewing and projection matrices allow a point in world coordinates to be converted to a cameras local coordinates and then be projected onto the cameras image plane. These matrices are also part of the Camera Model. The first of these matrices is the view matrix M_v.

$$M_v = \begin{bmatrix} x_u & y_u & z_u & -x_e \\ x_v & y_v & z_v & -y_e \\ x_w & y_w & z_w & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

The view matrix converts points from world coordinates to camera coordinates so that they can be projected onto the screen. Sometimes it is also necessary to convert camera coordinates back to world coordinates. This is done by taking the inverse of the view matrix denoted by M_v^{-1}.

$$M_v^{-1} = \begin{bmatrix} x_u & x_v & x_w & x_e \\ y_u & y_v & y_w & y_e \\ z_u & z_v & z_w & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

To get an image onto the screen, it has to be projected onto the image plane. In our system, we use a perspective projection matrix M_p.

$$M_p = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{A.3}$$

In this matrix, n is the near plane, f is the far plane, l, r, t, b are the left, right, top and bottom bounds that make up the screen relative to the optical axis (See Figure A.2). When a

point is projected onto the screen, its converted into a (u,v) point along with a z value used by the depth buffer. For a point to appear in the image, the u and v values must be in the range of -1 and 1 as shown in Figure A.3. Points not in this range fall outside the image and are not seen by the camera.

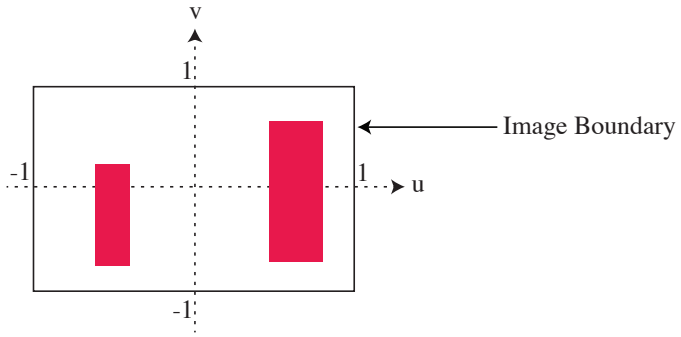


Figure A.3: An object is visible by a camera if its projected bounds are within the -1 to 1 range.

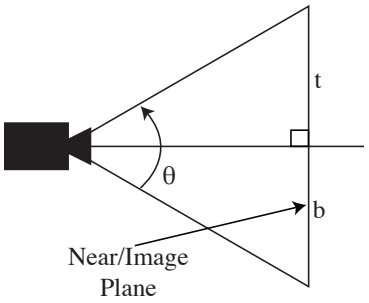


Figure A.4: A cameras field of view is defined as the angle between the cameras origin and the top and bottom bounds of the image plane. A smaller angle results in objects taking up more space when projected onto the image, while a larger value results in objects taking up less space in the image.

A.2 PTZ Camera

The PTZ Camera layer adds an interface to the Camera Model which can be used to control a camera. It includes all of the options that are found on a physical camera such as the ability to zoom in or zoom out, to pan left or pan right, to tilt up or tilt down. For all of these options the

angle as well as the amount of time it should take to transition from the current state to the new state can be specified. There is also an option to revert the camera back to its default position.

There also needs to be a way to mimic the motors on a PTZ camera. If a PTZ camera is set to pan 20 degrees to the right, it does not occur instantly. It takes some time for the panning motor to adjust the camera. To achieve the same effect, we use simple animations.

The PTZ camera has a list of animations which are made up of a start value, end value, duration, a pointer to the object that needs to be animated and the function that needs to be called. During each time step, the parameter being animated is slowly adjusted by a fraction of the difference between the start and end values until the animation is complete. We also have the ability to modify the animation while it is running if any sudden changes need to occur.

The commands available to both active PTZ and static wide-FOV cameras are listed in Table A.1.

Camera Type	Command	Description
PTZ & wide-FOV	setResolution	Set the resolution of the image
	getImage	Gets the latest image from the camera
PTZ	panLeft	Pan left by θ degrees
	panRight	Pan right by θ degrees
	tiltUp	Tilt up by θ degrees
	tiltDown	Tilt down by θ degrees
	zoomIn	Zoom in by θ degrees
	zoomOut	Zoom out by θ degrees
	default	Reverts the camera to its default settings

Table A.1: Simulated passive wide-FOV and active PTZ cameras support a set of commands similar to those available in a typical IP camera.

A.3 High Level Logic

The high level logic layer is where is where the camera control and coordination logic is implemented. For our implementation, we chose to break this up into three layers: the routine layer, the activity layer and the control layer (See Figure A.5).

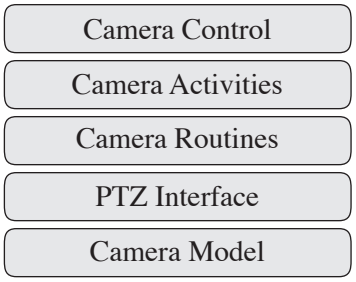


Figure A.5: The layered architecture of our Smart Camera Nodes.

The routine layer implements routines that communicate directly with the PTZ Camera layer below. This includes a tracking routine, a fixate routine, a zoom routine and a search routine.

A.3.1 Tracking Routine

The tracking routine replaces the computer vision routines that would typically be used to detect and track objects in an image generated by a physical camera. This routine iterates over the list of pedestrians and projects their bounding boxes onto a cameras image plane to see which ones appear in a cameras image (See Figure A.6).

At this time, we do not take into account occlusions; however, such a feature would be fairly straightforward to add. By making each person a different colour, it is straightforward to generate a simple image that contains a rectangle representing each pedestrian visible by the camera. A simple image operation could then be performed to find pixels of a specific colour to see if a particular pedestrian is visible.

A.3.2 Fixate and Zoom Routines

The camera tracks a pedestrian by using fixate and zoom routines [30]. Before tracking can occur, the camera must first compute the region of interest. The region of interest is the portion of the image that the camera wants to focus on. It is a rectangle that tightly encloses all of the pedestrians the camera wants to track.

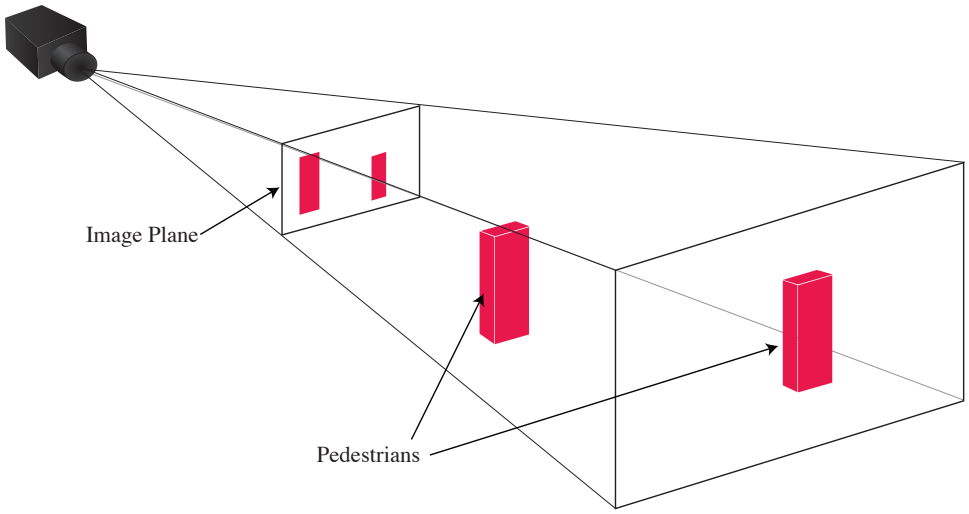


Figure A.6: A generated image uses a 2D coordinate system with its origin at the centre of the image. Projected points inside the range -1 and 1 appear in the image, while points outside this range do not.

The fixate routine brings the region of interest into the centre of the image by adjusting the tilt and pan angles of the PTZ camera. The zoom routine adjusts the field of view of the camera so that the region of interest occupies the desired percentage of the screen. This is called the coverage factor. Figure A.7 shows the fixate and zoom routines in action. The first row shows the camera panning to the right to fixate on the pedestrian. The second row shows the camera zooming in on the same pedestrian.

When adjusting the pan, tilt, and zoom values, a Proportional Integral Derivative controller (PID controller) [1] is used to compute how much the pan, tilt or zoom values should be changed by. A PID controller computes the error between the current value and the desired value by taking into account the past error and potential future error. PID controllers are widely used in control systems such as PTZ cameras.

The fixate and zoom routines operate independently of each other. This means a camera can fixate and zoom in at the same time. When the region of interest is close to the edge of the image, the camera automatically zooms out to keep the object in view.

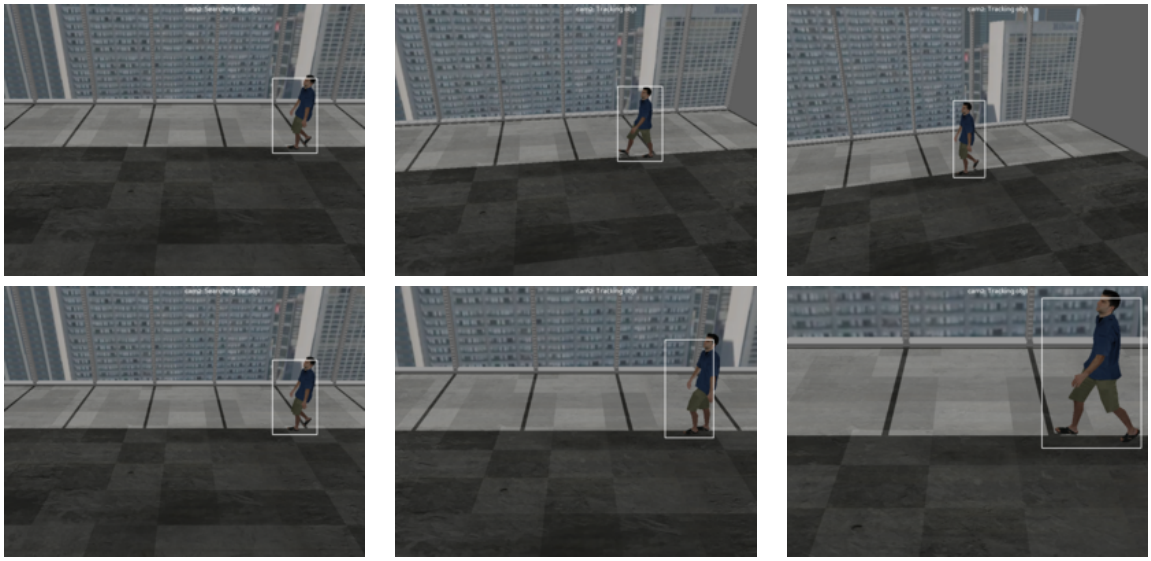


Figure A.7: Fixate and Zoom routines in action: the top row shows a camera fixating on the pedestrian, while the bottom row shows a camera zooming in on a pedestrian

A.3.3 Activity Layer

The Activity Layer is responsible for the level 2 behaviours shown in Figure 3.2. It also handles behaviour arbitration so that a single ROI is passed down to the Camera Routine Layer.

The set of activities that a camera can engage in are:

- $idle(c_i)$: The camera c_i is currently not performing any observation task.
- $observing(c_i, h_j)$: The camera c_i is currently observing pedestrian h_j.
- $evaluating(c_i, h_j)$: The camera c_i is currently evaluating its suitability to observe pedestrian h_j. A camera cannot take part in any negotiations involving h_j without first knowing its suitability to that task; suitability encodes the success probability of a camera with respect to an observation task. Typically, a camera evaluates its suitability to an observation task when it receives the task request from a neighbouring camera.

Behaviour Arbitration

Multiple instances of level 2 behaviours—Observe and Evaluate—may be active simultaneously in the activity layer. There is a one to one mapping between camera activities *observing*

and *evaluating* and these two behaviours. All of the activities in the Activity Layer rely upon the routines in the Camera Routine Layer. The routines in the Camera Routine Layer are singletons. This suggests that behaviour arbitration is needed to decide which routines will be called and what parameters will be passed to them. Behaviour arbitration aims to avoid negative consequences of behaviour interactions such as any one behaviour taking over camera resources and locking out all of the other behaviours. Behaviour arbitration also addresses the problem of *behaviour dithering*, where a camera will constantly switch between two (or more) competing tasks. We employ the behaviour arbitration model presented in [31].

Appendix B

Camera Network Simulator

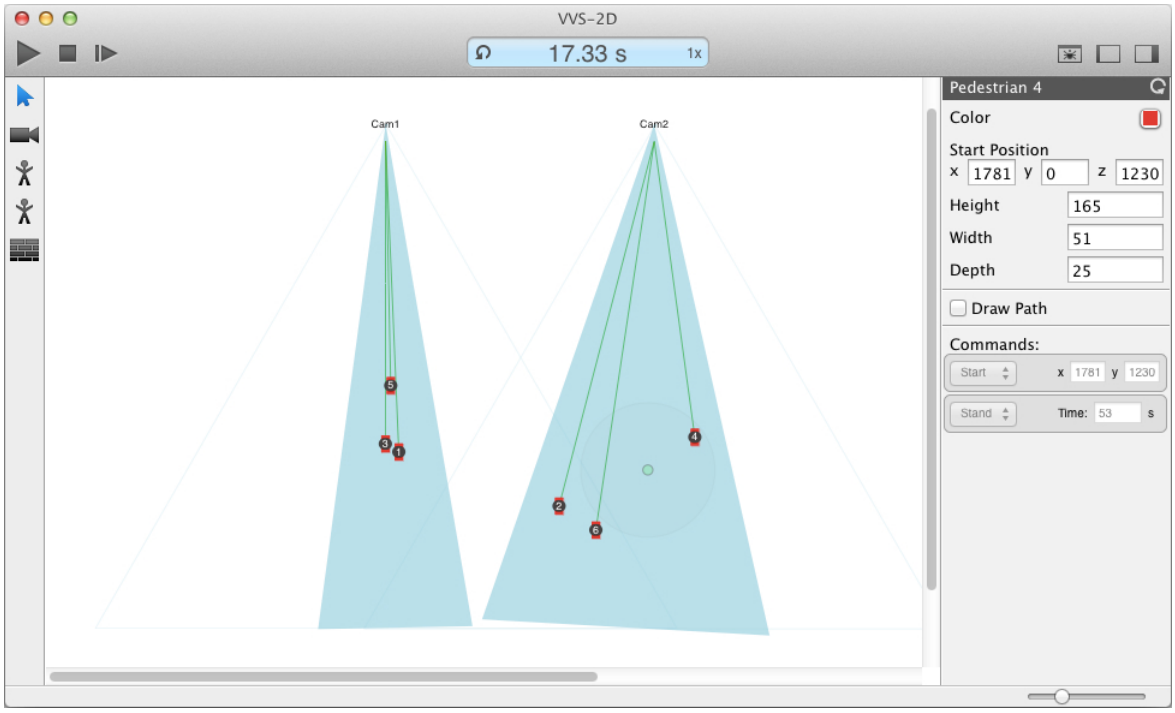


Figure B.1: A screenshot of our 2D camera network simulator which was used to test our approach.

To evaluate the proposed handoff technique, we developed a PTZ camera network simulator, which is shown in Figure. B.1. The simulator—which is implemented in Python—is able to simulate the movement of pedestrians, camera logic, camera motors (for PTZ cameras), inter-camera communication and camera sensing. It allows us to quickly set up different scenarios

```
<?xml version='1.0' encoding='UTF-8'?>
<vvs-2d version='1.1'>
<cameras>
  <camera pos='200.0 0.0 100.0' direction='0.0 0.0 1.0' id='1'>
    <up_vector>0 1 0</up_vector>
    <near_plane>1</near_plane>
    <far_plane>1500</far_plane>
    <default_fov>60</default_fov>
    <fov_limits>10 60</fov_limits>
    <pan_limits>0 0</pan_limits>
    <tilt_limits>-30 30</tilt_limits>
  </camera>
</cameras>
<pedestrians>
  <pedestrian height='165.0' width='51.0' depth='25.0'>
    <commands>
      <command type='start' x='500' z='300' radius='200'/>
      <command type='move' speed='walk' x='50' z='300' radius='200'/>
      <command type='stand' time='10'/>
    </commands>
  </pedestrian>
</pedestrians>
</vvs-2d>
```

Figure B.2: A sample scenario configuration file.

which can be saved and executed many times eliminating the need to set up real cameras and the need to get volunteers to act out different scenarios.

The simulator has graphical tools for creating, configuring and saving a scenario. By selecting one of the tools on the left and clicking on the canvas, a camera or pedestrian will be added to the scene. The property editor on the right can be used to tweak all of the properties of a camera or pedestrian making setting up a scenario quick and easy. A simulation can be started by pressing the Run button or the Spacebar can be pressed to move the simulation forward by one time step. The simulator uses an XML based scenario configuration file making it possible to write a configuration file yourself or tweak a file generated by the simulator. A sample configuration file is provided in Figure B.2

In addition to our 2D Camera Network Simulator, we also developed a 3D Virtual Vision

Simulator shown in Figure B.3. This simulator uses the same virtual camera architecture as our 2D simulator making it possible to test the same camera control and coordination approach in both simulators. It too is implemented in Python using the Panda3D game engine. Because we use a 3D game engine to render the view of each camera, it is possible to use real computer vision techniques for detecting and tracking objects in the rendered images allowing more realistic testing conditions. For a detailed description of the 3D Virtual Vision Simulator, we refer the reader to [39]. Do to its speed and flexibility, we chose to run all of our tests in our 2D simulator.

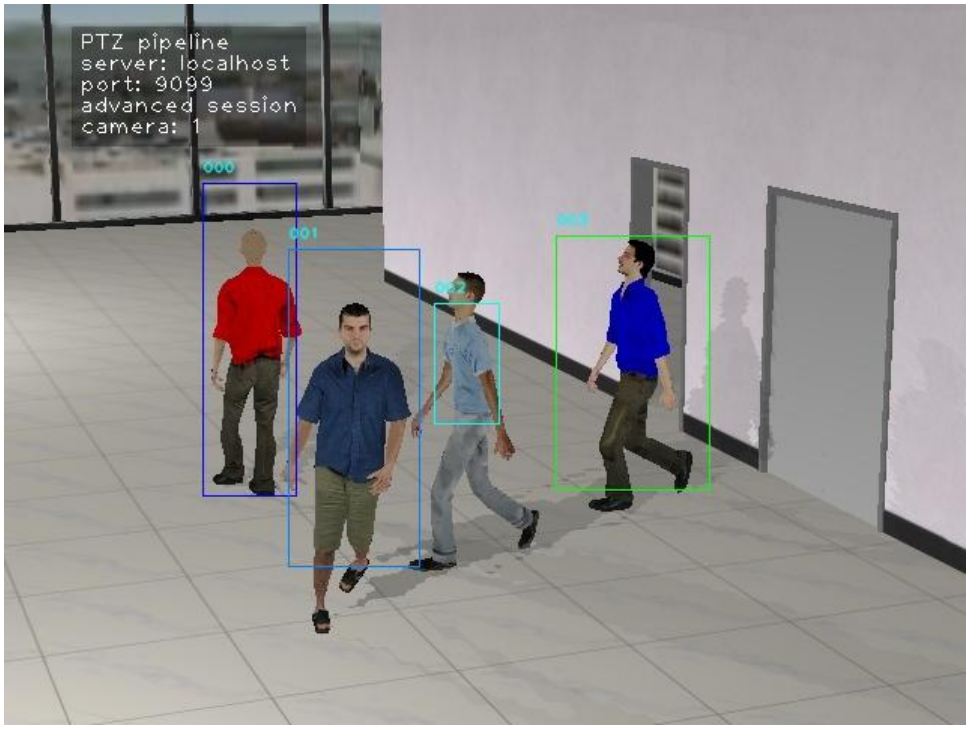


Figure B.3: A screenshot of our 3D virtual vision simulator

Appendix C

Virtual Pedestrians

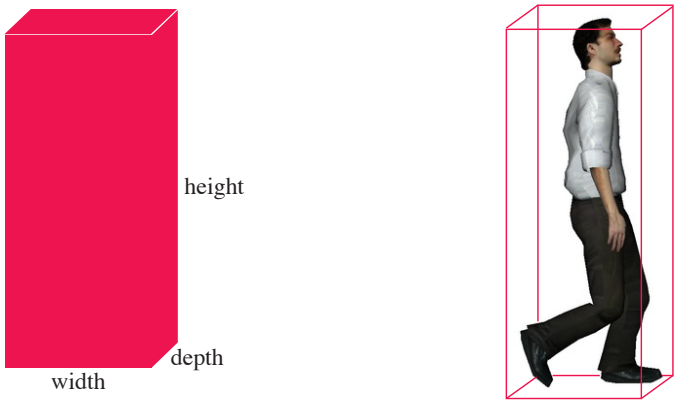
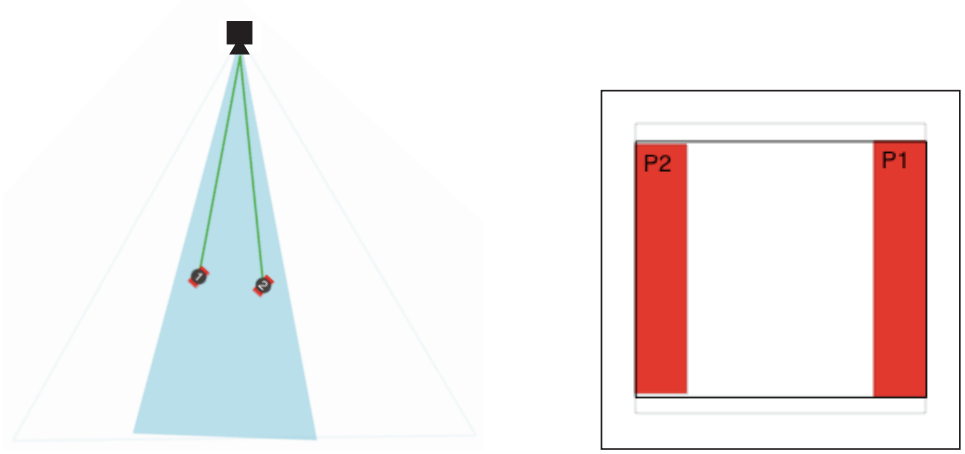


Figure C.1: Our virtual pedestrians are represented as rectangular prisms made up of a width, depth and a height. In our 3D simulator we use the bounding box of the 3D model used to represent the pedestrian.

For our simulator to be useful, it requires objects that can be observed by the virtual cameras. This is where our virtual pedestrian come into play. Our virtual pedestrians represent the objects that are typically tracked by surveillance cameras. In our case we assume these objects to by people; however they could easily be some other object such as an animal, a car or anything else. They are represented as rectangular prisms as shown in Figure C.1 (a). The rectangular prism is a bounding box defined by three values: $width$, $depth$ and $height$. In our 2D simulator, these values can be specified as properties of a pedestrian, while in the 3D



(a) (b)

Figure C.2: (a) A camera observing two pedestrians in our 2D Camera Network Simulator. (b) The image produced by projecting the bounding boxes of the two pedestrians onto the cameras image plane.

simulator, they are based on the 3D model used to represent the pedestrian (see Figure C.1 (b)).

This bounding box is what allows us to implement a simple tracker by projecting the 8 points that make up the bounding box onto a cameras image plane. We can see whether or not the object appears in the image, what part of the image it takes up and how big it is. This is shown in Figure C.2

To test handoff approaches we need the pedestrians to move around the scene. Pedestrian motions are scripted via a sequence of *start*, *move* and *stand* commands. The *start* command defines the region where a pedestrian will begin its motion. It is defined by a 2D position on the ground as well as a radius. The *move* command defines the region where a pedestrian should move to. It is also defined by a 2D position and a radius; however, it also includes a speed. We have defined three speeds a pedestrian can move in: walk, jog and run.

When a start or move command is initiated, a random position within the specified radius is chosen as the location where the pedestrian will start or move to. This allows us to re-run a scenario multiple times with a different position being chosen each time (See Figure C.3 (b)). For the tests presented in this thesis, the radius is set to 200 units (or 2 meters in simulation space). Movement is achieved by interpolating the position of the pedestrian from its starting

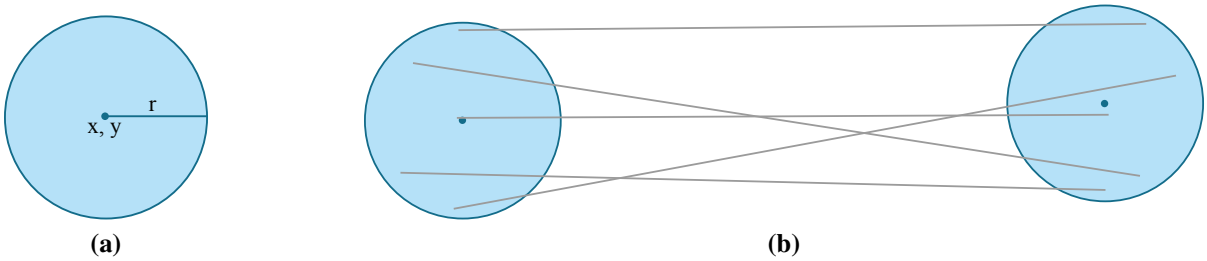


Figure C.3: (a). The position a pedestrian moves to is randomly chosen at runtime based on the position and radius of a move command. (b). The same scenario can be run multiple times producing a different path each time.

position to its end position. Currently we use straight line motion and ignore collisions; however, it is possible to use a steering approach such as OpenSteer [33] or RVO2 [36] instead of the simple interpolation. Using a steering approach would produce a better looking simulation, but it would not have any effect on the results and therefore we chose not to incorporate it at this time.

Bibliography

[1] Mituhiko Araki. Pid control. *Control systems, robotics and automation*, 2:1–23, 2002.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE journal of Robotics and Automation*, 2:14–23, 1986.

[3] Zezhi Chen and T. Ellis. Automatic lane detection from vehicle motion trajectories. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pages 466–471, Aug 2013.

[4] Zhaolin Cheng, Dhanya Devarajan, and Richard J. Radke. Determining vision graphs for distributed camera networks using feature digests. *EURASIP Journal on Advances in Signal Processing, Special Issue on Visual Sensor Networks*, pages 1–19, 2007.

[5] Edward G. Coffman and John L. Bruno. *Computer and job-shop scheduling theory*. John Wiley and Sons, 1976.

[6] R.T. Collins and Y. Tsin. Calibration of an outdoor active camera system. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages –534 Vol. 1, 1999.

[7] C.F. Crispim, V. Bathrinarayanan, B. Fosty, A Konig, R. Romdhane, M. Thonnat, and F. Bremond. Evaluation of a monitoring system for event recognition of older people. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pages 165–170, Aug 2013.

[8] Dhanya Devarajan, Richard J. Radke, and Haeyong Chung. Distributed metric calibration of ad hoc camera networks. *ACM Transactions on Sensor Networks*, 2(3):380–403, August 2006.

[9] Gianfranco Doretto, Thomas Sebastian, Peter Tu, and Jens Rittscher. Appearance-based person reidentification in camera networks: problem overview and current approaches. *Journal of Ambient Intelligence and Humanized Computing*, 2(2):127–151, 2011.

[10] TJ Ellis, Dimitrios Makris, and James Black. Learning a multi-camera topology. *Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pages 165—-171, 2003.

[11] Markus Enzweiler and Dariu M. Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2179–2195, Dec 2009.

[12] Lukas Esterle and Peter R. Lewis. A socio-economic approach to online vision graph generation and handover in distributed smart camera networks. *ACM Transactions on Sensor Networks*, 0(0):1–24, 2011.

[13] Lukas Esterle, Peter R. Lewis, Martin Bogdanski, Bernhard Rinner, and Xin Yao. A socio-economic approach to online vision graph generation and handover in distributed smart camera networks. *Fifth ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–6, August 2011.

[14] Stanislav Funiak, Carlos Guestrin, Mark Paskin, and Rahul Sukthankar. Distributed localization of networked cameras. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, pages 34–42, New York, 2006.

[15] N. Gheissari, T.B. Sebastian, and R. Hartley. Person reidentification using spatiotemporal appearance. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1528–1535, 2006.

[16] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[17] IndigoVision. Toulouse Casino Project. www.indigovision.com/documents/public/project-briefs/Toulouse Casino-Project brief-Letter.pdf, 2012.

[18] O Javed and S Khan. Camera handoff: tracking in multiple uncalibrated stationary cameras. *IEEE Computer Society Workshop on Human Motion*, 2000.

[19] Younggwan Jo and Joonhee Han. A new approach to camera hand-off without camera calibration for the general scene with non-planar ground. *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, 2006.

[20] Yiming Li and Bir Bhanu. Utility-based dynamic camera assignment and hand-off in a video network. *Second ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–9, 2008.

[21] Yiming Li and Bir Bhanu. A Comparison of Techniques for Camera Selection and Hand-Off in a Video Network. *Distributed Video Sensor Networks*, 2009.

[22] Dimitrios Makris, Tim Ellis, and James Black. Bridging the gaps between cameras. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 00, pages 0–5, 2004.

[23] C. Micheloni, B. Rinner, and G.L. Foresti. Video analysis in pan-tilt-zoom camera networks. *IEEE Signal Processing Magazine*, 27(5):78–90, Sep 2010.

[24] Kazuyuki Morioka, Szilveszter Kovacs, Joo-Ho Lee, Peter Korondi, and Hideki Hashimoto. Fuzzy-based camera selection for object tracking in a multi-camera system. *2008 Conference on Human System Interactions*, pages 767–772, May 2008.

[25] N. Nitta, T. Nakazaki, K. Nakamura, R. Akai, and N. Babaguchi. People counting across spatially disjoint cameras by flow estimation between foreground regions. In *Advanced*

Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on, pages 414–419, Aug 2013.

[26] C. Pane, M. Gasparini, A Prati, G. Gualdi, and R. Cucchiara. A people counting system for business analytics. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pages 135–140, Aug 2013.

[27] Johnny Park, PC Bhat, and AC Kak. A look-up table based approach for solving the camera selection problem in large camera networks. *Proceedings of the International Workshop on Distributed Smart Cameras*, 2006.

[28] Markus Quaritsch, Markus Kreuzthaler, Bernhard Rinner, Horst Bischof, and Bernhard Strobl. Autonomous Multicamera Tracking on Embedded Smart Cameras. *EURASIP Journal on Embedded Systems*, 2007:1–10, 2007.

[29] Faisal Z. Qureshi. Collaborative sensing via local negotiations in ad hoc networks of smart cameras. *Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras - ICDSC '10*, page 190, 2010.

[30] Faisal Z. Qureshi and Demetri Terzopoulos. Virtual vision: Visual sensor networks in virtual reality. In *Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology*, VRST '07, pages 247–248, New York, NY, USA, 2007. ACM.

[31] Faisal Z. Qureshi and Demetri Terzopoulos. Multi-camera Control through Constraint Satisfaction for Persistent Surveillance. *2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance*, pages 211–218, September 2008.

[32] R. Ratajczak, T. Grajek, K. Wegner, K. Klimaszewski, M. Kurc, and M. Domanski. Vehicle dimensions estimation scheme using aam on stereoscopic video. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pages 478–482, Aug 2013.

[33] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference*, pages 763–782, 1999.

[34] R. Romdhane, C.F. Crispim, F. Bremond, and M. Thonnat. Activity recognition and uncertain knowledge in video scenes. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pages 377–382, Aug 2013.

[35] Wei Shao and Demetri Terzopoulos. Autonomous pedestrians. *Graph. Models*, 69(5-6):246–274, September 2007.

[36] Jamie Snape, Stephen J Guy, Deepak Vembar, Adam Lake, and Ming C Lin. Reciprocal collision avoidance and navigation for video games. In *Game Developers Conference*, San Fransico, 2012.

[37] B. Song and A.K. Roy-Chowdhury. Robust Tracking in A Camera Network: A Multi-Objective Optimization Framework. *IEEE Journal of Selected Topics in Signal Processing*, 2(4):582–596, August 2008.

[38] Bi Song, Cristian Soto, Amit K. Roy-Chowdhury, and Jay A. Farrell. Decentralized camera network control using game theory. In *Second ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–8, 2008.

[39] W. Starzyk, A Domurad, and F.Z. Qureshi. A virtual vision simulator for camera networks research. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pages 306–313, May 2012.

[40] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4), December 2006.