UNIVERSITY OF CALIFORNIA,
IRVINE

Towards Adaptation in Sentient Spaces.

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Ronen Vaisenberg

Dissertation Committee:
Professor Sharad Mehrotra, Chair
Professor Deva Ramanan
Professor Nalini Venkatasubramanian
Professor Ramesh Jain

2012

DEDICATION

To Liat and Ariella

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am deeply grateful to my supervisor Prof. Sharad Mehrotra, who have guided me with his invaluable suggestions and criticisms. Whenever we had a great idea, Sharad's eyes would light up with excitement. Not once we would run out of the office calling out faculty and students to share our idea and get their feedback. Its been a real pleasure working with a person that is really passionate about what he does. I am deeply grateful to Prof. Deva Ramanan for his sharpness of thought. Not once I dropped in with a paper, filled with equations and formulations which I could not reason about and Deva made it all so simple in a matter of minutes. It was a great pleasure for me to have a chance of working with them.

Along the way, I have met a fantastic group of people who each contributed to my overall experience - Prof. Nalini Ventakasubramanian for her valuable suggestions and connections to other work. Dr. Opher Etzion who informed me of the applications of my research domain in an industrial lab setting and help communicate that to the people responsible of awarding Ph.D fellowships at IBM.

This research was partially supported by the National Science Foundation under Award Numbers 0331707 and 0331690, IBM PhD fellowship, Yahoo! best dissertation award and a Google PhD forum award. Any opinions, findings, conclusions or recommendations expressed in this thesis are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, IBM, Yahoo! or Google.

Finally, I would like to thank Liat for convincing me to do what I really want, Ariella for putting it all in perspective and my parents for always having the intuition for the right advice.

CURRICULUM VITAE

Ronen Vaisenberg

EDUCATION

Doctor of Philosophy in Computer Science — **2008-2012**
University of California, Irvine — *Irvine, California*

Master of Science in Computer Science — **2006-2008**
University of California, Irvine — *Irvine, California*

Master of Science in Computer Systems Engineering — **2003-2005**
Ben Gurion University of the Negev — *Beer - Sheva, Israel*

Bachelor of Science in Computer Engineering — **1997-2001**
The Open University of Israel — *Beer - Sheva, Israel*

RESEARCH EXPERIENCE

Graduate Research Assistant — **2006-2012**
University of California, Irvine — *Irvine, California*

Research Intern — **Summers of 2008,2009,2010**
IBM Haifa Research Lab — *Haifa, Israel*

Research Intern — **Summer of 2010**
Google Research — *Mountain View, California*

SELECTED HONORS AND AWARDS

IBM Ph.D Fellowship Award **2010**
IBM Inc.

Yahoo! Best Ph.D dissertation award
Yahoo! Inc.

Student Invitation and travel grant to attend a Dagstuhl Seminar **2010**
Dagstuhl Seminar on Event Processing

First prize at IEEE Percom'09 Ph.D Forum **2009**
IEEE Percom'09 Ph.D Forum

ICS-UCI Graduate Fellowship **2006-2009, 2011**
University of California, Irvine

Full Scholarship for Master's Degree **2004-2005**
Ben-Gurion University

Master's Thesis Selected for Patent Application **2005**
Ben-Gurion University - the Applied Sciences Arm of the University

Dean's and President's high honor list for 4 years **1997-2001**
The Open University of Israel

REFEREED JOURNAL PUBLICATIONS

SEMARTCam Scheduler - Semantics Driven Real-Time Data Collection from Indoor Camera Networks to Maximize Event Detection. (R. Vaisenberg, S. Mehrotra and D. Ramanan) **2010**
Journal of real time image processing

Database Encryption An Overview of Contemporary Challenges and Design Considerations. (E. Shmueli, R. Vaisenberg, Y. Elovici, C. Glezer) **2009**
SIGMOD Record

REFEREED CONFERENCE PUBLICATIONS

ZoombieCAM Scheduler - *Zoom B*ased *I*nformation *E*xtraction from Indoor *Cam*era Networks to Maximize the Application Utility. (R. Vaisenberg, S. Mehrotra and D. Ramanan). **2011**

Submitted to IEEE Pervasive Computing and Communication (PerCom)

Load Balancing Streams on a Cluster of Machines. (R. Vaisenberg and S. Mehrotra). **2011**

In a Technical report, in preparation for submission

Exploiting Semantics For Scheduling Data Collection From Sensors On Real-Time To Maximize Event Detection. (R. Vaisenberg, S. Mehrotra and D. Ramanan). **2009**

In Multimedia and Computer Networks (MMCN09, San Jose, CA)

Exploiting Semantics for Sensor Recalibration in Event Detection Systems. (R. Vaisenberg , S. Ji, B. Hore, S. Mehrotra and N. Venkatasubramanian). **2008**

In Multimedia and Computer Networks (MMCN08, San Jose, CA).

SMPL a Specification Based Framework for the Semantic Structure, Annotation and Control of SMIL Documents. (R. Vaisenberg, S. Mehrotra and R. Jain). **2009**

In IEEE International Workshop on Data Semantics for Multimedia Applications

Designing Secure Indexes for Encrypted Databases. (E. Shmueli, R. Vaisenberg, Y. Elovici and E. Gudes). **2005**

In Database Security (DBSec05, Storrs, CT).

A Structure Preserving Database Encryption Scheme. (Y. Elovici, R. Vaisenberg, E. Shmueli and E. Gudes). **2004**

In Secure Data Management (SDM04-In Conj. with VLDB04, Toronto, Canada)

INVITED PAPERS AND BOOK CHAPTERS

SATWARE: A Semantic Approach for Building Sentient Spaces. (D. Massaguer, S. Mehrotra, R. Vaisenberg, and N. Venkatasubramanian). **2011**

In distributed video sensor networks, Springer. Book Chapter.

Video Entity Resolution: Applying ER Techniques for Smart Video Surveillance. (L. Zhang, R. Vaisenberg, S. Mehrotra, D. V. Kalashnikov). **2011**

In Information Quality and Quality of Service for Pervasive Computing. Invited paper.

ABSTRACT OF THE DISSERTATION

Towards Adaptation in Sentient Spaces.

By

Ronen Vaisenberg

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 2012

Professor Sharad Mehrotra, Chair

Recent advances in sensing technologies enabled infusing information technology into physical processes. This offers unprecedented opportunities, the impact of which will rival (if not exceed) the opportunities created by the WWW. Such sensor enabled environments can realize sentient spaces that have the potential to revolutionize almost every aspect of our society. Sentient systems observe the state of the physical world, analyze and act based on it. Sentient systems enable a rich set of application domains including smart video surveillance, situational awareness for emergency response and social interactions in instrumented office environments to name a few.

In this thesis, we utilize additional information, namely the semantics of the monitored world, to improve the adaptation ability of sentient systems. We design real-time algorithms that utilized these models to address challenges such as data collection in the presence of resource constraints, sensor actuation and automatic re-calibration of analysis algorithms for correct monitoring. Our approach abstracts the application level from sensor level challenges, namely: data collection, actuation and re-calibration challenges. We propose addressing these challenges by a middleware layer that predicts the future state of the monitored environment and automatically schedules, actuates and re-calibrates sensors to maximize an application specified re-

ward function. We evaluate our approach in a monitored building setting in which surveillance cameras are using to detect events taking place in the building. Our results suggest that sensor level challenges can be abstracted and effectively addressed by a middleware layer. We believe that our approach will serve as a fundamental building block for building the next generation sentient systems.

Chapter 1

Introduction

1.1 Sentient Spaces

Advances in sensing, communications and computing technologies have made it possible to build large-scale physical spaces with diverse embedded sensors, ubiquitous connectivity and computing resources that, together, enable the creation of *sentient spaces*. Sentient spaces provide a view of the processes that are taking place in a physical space in a manner that enables human users and software components to adapt to the current state of the sentient space. Sentient space applications observe, analyze and act based on the state of the physical world.

It is now acknowledged [42] that infusing information technology into physical processes and elements offers unprecedented opportunities whose impact will rival (if not exceed) those opportunities created by the WWW. Coupled with appropriate feedback control, such sensor-enabled environments can realize sentient spaces that have the potential to revolutionize almost every aspect of our society. Application domains that stand to benefit range from domains such as infrastructure security, transporta-

tion, medicine, energy, entertainment, communication and social-networking applications. Interactive sentient systems enable a tight coupling of decision-making and computation with the physical elements of an infrastructure and are thus expected to dramatically increase the adaptability, efficiency, functionality, reliability, safety and usability of such systems.

The age of smart environments is just around the corner, as sensing technology coupled with cloud computing penetrates the main stream of consumer electronics. In 2015 alone, 645 million radio chips for wireless sensors will be sold, according to ABI research [47] (see figure 1.1).

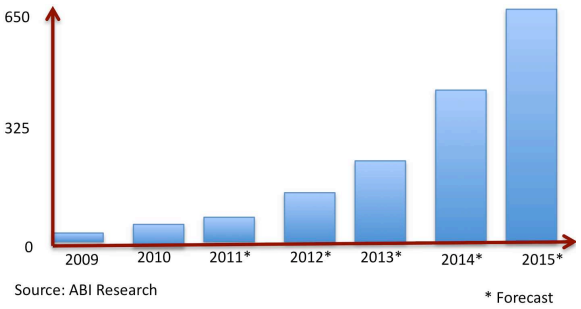


Figure 1.1: Projection of number of wireless sensors sold by 2015.

As a motivating application, consider "Meerkat"[1] - a system in which sensors are used to observe the environment and generate real-time notifications about activities that are taking place in the environment. For example, in a university setting, students are likely to subscribe to follow their advisor, which will result in real-time notifications being sent to students when their advisers arrive to the office. The users of the system can also get notifications when a large number of people gather in a particular location (e.g., talk of high interest). Finally, the system integrates multiple sensors, which monitor different systems in the building. Subscribers to coffee machine's update stream get notifications when fresh coffee is ready to be consumed. Subscribers to

[1]Meerkats watch the skies for birds of prey, such as hawks and eagles, that can snatch them from the ground. A sharp, shrill call is a signal for all to take cover. We named our motivating application after the mammal that symbolizes the most the sense-analyze-act cycle in the animal kingdom.

the printer stream can get notifications when the tuner is about to run out of ink. Historical video streams can be queried to investigate the rapid decline of the ink level. Such an application takes the social interactions in a building space to an entirely different level. In the context of this thesis we will refer to the university office space notification system as **MeerkatU**. Next, we describe the technologies related to sentient spaces, which enable the implementation of sentient space applications as the one described above.

1.2 Sentient Spaces and Related Technologies

Sentient space systems observe, analyze and act based on the state of the physical world. For example, in order to send a notification when the advisor of a student arrives to the office, MeerkatU needs to collect an observation from the hallways in proximity to the office of the faculty member. Next, it needs to process the observations and in those that contain an entity, determine the identity of the person in the field of view. If the identity is of a faculty member, the relevant notification is disseminated to its subscribers.

From the perspective of this thesis, we abstract the sentient space systems into three layers:

- **Application Layer** - responsible of specifying the application requirements.
- **Middleware Layer** - responsible of translating the application requirements to an execution plan.
- **Sensing Layer** - responsible of generating a digital measurement of the physical environment based on the execution plan.

In this thesis we focus on the middleware layer, which is a fundamental building block of sentient spaces. To illustrate the need for such a middleware, notice that in MeerkatU, the overhead of determining the identity of a person in the field of view of a camera can easily exhaust the system's computation resources. The number of observations generated by sensors is theoretically unbounded while the amount of images that a single computer (or even a cluster of computers), can process in order to determine the identity, is bounded. The middleware is responsible of making sense of the application needs and translating these needs to a real-time plan of action. The middleware is also responsible of translating a query, for example, determining the location of a faculty member, to a plan that best utilizes the resources available to answer the query. The plan that the middleware executes controls the data collection (get an observation near the office of a faculty member), sensor actuation (focus a camera to collect a high resolution facial image an entity for identity resolution) or sensor re-calibration (system suspects that the identity resolution algorithms require re-tuning).

There are numerous challenges in building the middleware for sentient spaces. This thesis focuses on challenges related to automatically adapting and controlling the sensors in order to improve the sentient space performance. In particular, we focus on automatic adaptations to address the following challenges: **Scalability Challenge**: Sentient spaces are characterized by large numbers of heterogeneous multimodal sensors that generate voluminous data streams. Since information utility decays with time, processing the relevant information quickly in the presence of computing and communication constraints is essential. In addition, the analysis of sensor readings is a computationally expensive operation. Although one would like to analyze all available sensor streams at all times, there are resource constraints, which range from networking limitations to sensor capture rate limitations, which prevent the system from monitoring all sensors at all times. MeerkatU would like to process each camera

for identity resolution, but this is not possible due to resource constraints. For example, the number of frames that we can process at a given time is limited. Moreover, In some cases, where acting on findings requires very high confidence, e.g., triggering alarm, there is a need for a "human-in-the-loop" to increase the analysis accuracy and reduce false alarms. The amount of information a person can comprehend is limited and thus, again, only a certain subset of readings can be verified and analyzed by a human operator. Thus, sentient spaces technology is faced with the need to choose which streams to analyze at any given time.

Adaptivity Challenge: Adaptation can occur at the system level or in the applications executing on the infrastructure and may be manifested as changes in how the space is observed (changes in sensing parameters) and how awareness is obtained from the sensed data. System adaptation involves the design of adaptation algorithms that optimize the usage of the infrastructure (via resetting device parameters and system policies) when changes occur. For example, changing the image resolution, or actuating the optical zoom of a camera are examples of such adaptations that are aimed at generating a more detailed observation. MeerkatU would need to actuate the cameras such that the identity resolution process has access to high resolution facial images of entities. This would reduce the uncertainty of the extraction process.

Robustness Challenge: Given that sensors are deployed in unsupervised and exposed environments, physical perturbations (wind, motion, tampering) might occur. This may change the validity of the information being captured. System robustness involves the resign of algorithms that detect and recover from situations where the observe and analyze in the observe-analyze-act cycle require re-tuning. For example, the appearance of a monitored phenomena[2] might change as a result of a physical perturbation of the observing sensor. The system needs to adopt to the change in

[2]Phenomena is defined in Section 3. For now, consider the phenomena being the activity of interest being monitored by the sentient system.

location and resume the monitoring automatically. MeerkatU would like sensors monitoring the coffee machine to be able to recover from perturbations to the camera, misplacements of the coffee machine and even from the replacement of the sensor used for monitoring.

Apart from adaptation, there are other challenges related to the middleware layer, which we introduce for completeness:

Abstraction Challenge: In order to support the development of a wide range of applications in sentient spaces, we must provide the right programming abstractions that address separation of concerns, i.e. express high level application goals, determine how events are detected and which sensors are used. A user of MeerkatU, can specify that they are interested in tracking their advisor. The system should be able to translate this high level request to a physical execution plan. The plan will contain a list of sensors to probe for frames, the processing operators required for the task of person recognition and a monitoring module, which will execute the plan within the resource constraints.

Privacy & Security Challenges: As sentient environments permeate everyday life, especially in residential and workspace settings, significant privacy and security challenges arise. From the privacy perspective, a key challenge is that fine-grained data collected through sensors could potentially be misused to reveal personally identifying information about individuals (e.g., their activities, work habits, preferences, or beliefs). In MeerkatU the subscribers of the notifications ask to get a notification about the whereabouts of another individual, a mechanism for people to opt out and protect their information has to be in place to enforce privacy preferences.

Uncertainty Challenge: The detection of events from sensor readings is a challenging, open research problem. For example, consider the problem of identifying a

person in a video stream from a camera sensor. The response to this query depends upon the correctness of the extractors and results in uncertainty associated with the observations generated. In MeerkatU, the process of identifying a person is inherently imperfect and will introduce a level of uncertainty to the extraction process.

Finally, since the observe-analyze-act cycle occurs while activities are taking place in real-time in the monitored environment, all actions need to be real-time since higher than tolerable latencies prevent the system from acting on its findings. A delayed action, for example, attempting to collect a video of a suspicious activity in a region of interest, is of no utility once the real-world activity has ended.

1.3 Focus of Thesis

At UCI we are in the process of building a middleware for sentient space which we refer to as Satware[3]. Satware is a multimodal sensor data stream querying, analysis, and transformation system. Satware provides applications with a semantically richer level of abstraction of the physical world as compared to raw sensor streams, providing a flexible and powerful application development environment. Decoupling (and abstracting) events from the underlying sensors offers numerous advantages. First and foremost, it provides a framework for the underlying middleware and runtime to optimize application execution based on a variety of application and resource constraints.

In this thesis we focus on creating the software support for sentient spaces to make such systems adaptive as described in detail in the following sections. Specifically, we address the scheduling, actuating and re-calibration of sensors for the task of maximizing application reward associated with observation of a phenomena. We

[3]Satware is described in more details in Sec. 2.2.

propose a unique approach to the task of adaptation:

We model and learn **phenomena semantics** - the underlaying pattern by which the phenomena evolves over time and space. Phenomena semantics, or **semantics** in the context of this thesis, represent the behavioral nature of the phenomena. The word "semantics" is often used to denote the problem of understanding the meaning behind signs, symbols, sounds or expressions. In the context of this thesis, we use semantics to refer to the understanding of the relationship between the current state of the phenomena and the possible future states it can evolve to. In particular, we model the phenomena semantics as state transition probabilities, which in turn allow the sentient system an "understanding" of the way the phenomena evolves over time and space. Such state transition probabilities, in turn, allows the system to decide which sensor readings are more important and thus need to be processed. Moreover, we use the phenomena semantics to compute the expected long term utility of each action and decide on the best course of sensor actuations. Finally, we use phenomena semantics to detect and determine if the history of extracted observations differs significantly from its "normal" behavior and adjust the process that analyzes the sensor readings accordingly.

To illustrate how challenging this problem is, consider the following questions:

- How should the phenomena semantics be represented?
- How should we take into account the application requirements, system constraints, and phenomena semantics to address the task of adaptivity?
- Due to resource constraints, our representation of the current state of the phenomena would, invariably be, incomplete. How should we balance the need to reduce the uncertainty with respect to the current state of the phenomena and the need to collect data of interest to the application?

And above all, the process that exploits the semantics must to be light weight in the sense that the adaptation (e.g., actuate and control sensors) is done in real-time. The phenomena is taking place right now, and does not wait for the monitoring system to make a call.

1.4 Overview of the Thesis

In Chapter 2, we summarize the related work and highlight the differences of the techniques proposed in this thesis from past work. In Chapter 3, we define the theoretical concepts and formally describe the problem that we address in this thesis. In Chapter 4, we describe how we learn semantics in sensor enabled spaces where the goal is to learn how the space is used. In Chapter 5, we discuss the data collection challenge. Systems that monitor a moderately large environment (e.g., building) require significant resources in terms of network, storage and processing and thus become a bottleneck. Furthermore, if human intervention is needed by visualization, for example, human attention is limited and thus becomes a bottleneck. In this chapter, we describe our approach for designing a schedule that exploits the phenomena semantics and maximizes the application reward within resource limitations.

In Chapter 6, we discuss the actuation challenge. Different applications have different sensing requirements and the data captured by a sensor can be varied based on parameters of the sensor. For example, camera sensors have a field of view and actuating the camera means setting the field of view as a function of the application requirements at any given time. The actuation challenge generalizes the data collection challenge - since parameter selection for sensors (e.g., the rate at which to collect data, or resolution of image, etc.) could be viewed as actuation of the sensors. And thus, in this chapter we describe our approach for designing a scheduler, which

exploits the phenomena semantics and controls the sensor state so that it is more likely to observe the phenomena of interest.

In Chapter 7, we discuss the reliability challenge that arises for two reasons: 1) Often in sensor environments where there is rapid deployment, there is no time to continuously calibrate sensors, so automated ways to calibrate are necessary. 2) Sensors are physical devices and typically calibrated to work under specific physical configurations. Physical devices may periodically lose their calibration due to (a) physical environment changes, (e.g., background changes, change in location of the monitored system) or (b) disturbance to the sensor (movement of a sensor). As a result, the analysis process may become invalid as time progresses. In this chapter, we describe our approach for detecting and re-calibrating sensor parameters based on the phenomena semantics. Chapter 8 concludes the thesis by highlighting its contributions and discussing future directions.

Chapter 2

Related Work

There has been extensive research in different areas of data management in sensor networks and pervasive environment that resulted in implementation of many systems. In the following section we provide a brief overview of some of these systems.

2.1 Event and Stream Processing Systems

2.1.1 Multimedia Stream Processing

Recently there have been systems for processing multimedia streams including IBM Smart Surveillance System (S3) [53], MedSMan [29] and IrisNet [18]. These systems provide features such as media capture, automatic feature extraction, declaration and query language, temporal stream operators, and querying algorithms. In this subsection we describe IBM S3 system and MedsMan in detail.

IBM S3 system. The IBM Smart Surveillance System is a middleware that provides video based behavior analysis. It has two main components, Smart Surveillance

Engine (SSE) and Middleware for Large Scale Surveillance (MILS). The main functionality of the SSE is to provide software based surveillance event detection. It uses video analysis techniques such as object detection, object tracking and object classification. The MILS middleware provides data management services for S3. It converts surveillance video into a relational database table. It also provides querying capabilities on the surveillance video. S3 also provides a privacy preserving surveillance feature that can hide personal information in different levels. Despite all the rich functionality that it provides, the S3 system is based on a centralized database that may result in scalability issues in systems with large number of video sources. Also it only considers one stream type (video).

MedSMan. MedSMan is a live multimedia management system. It consists of three main components: Stream Model, Environment Model, and Event Model. The stream model in MedsMan consists of two basic parts: media streams and feature streams. A media stream consists of tuples with sequence number, a time interval and media content corresponding to that interval. A feature stream consists of a sequence number, a timestamp, and a feature value that has occurred on that time. MedsMan defines stream operators that can convert media streams to feature streams and vice versa. In MedSMan an event has a start and end time. MedSMan allows a hierarchical event structure where an event can consist of a set of events. Based on its three main components, MedSMan defines a query language, which is an extension of CQL [4], to handle new concepts such as feature streams. Using this language, a user can define different queries on top of video streams and MedSMan evaluates them in real-time. In addition to video streams MedSMan supports audio streams; however, its architecture is not scalable and it should be extended to accommodate systems with large number of multimedia sensors.

As opposed to IBM S3 and MedsMan, Satware is a fully distributed highly scalable

system which was designed to process a very large number of streams utilizing parallel processing in a fully decoupled processing architecture. In this thesis we enhance the scalability and adaptivity of Satware by utilizing the semantics of the monitored phenomena.

2.1.2 Data Stream Processing

There has been a considerable work on data stream processing systems including TinyDB [32], Aurora [1], TelegraphCQ [11], and STREAM system [3]. These systems provide many features such as data model, continuous query semantic, query language, and scalable and adaptive continuous query processing. Much work has been done on different aspects of data acquisition and analysis in sensor networks which expands a broad spectrum from issues such as energy saving and query accuracy [31, 51]. Operator scheduling in data stream systems is studied in [5], where the goal of minimizing memory requirements for buffering tuples based on the selectivity of the detection operators.

However, none have proposed techniques have attempted to utilize phenomena semantics for the task of sensor adaptation in a principled manner as explored in this thesis. This thesis exploits the fact that the monitored phenomena has associated semantics which define the way the observations of the real world change over time. Exploiting these semantics opens the door to a large array of optimizations in the context of sensor networks.

2.2 Satware

Satware [34] is a distributed semantic based middleware for sentient spaces being designed at UCI to serve as an adaptive middleware for the sensor infrastructure that spans multiple buildings instrumented with diverse sensors including cameras, RFID, mote sensors, motion sensors, etc. Satware is a distributed semantic based middleware for sentient spaces. It is based on an architectural abstraction called virtual sensor that bridges the gap between the application-level concepts and the raw sensor data using operators that transform input sensor data streams (e.g. video sensor feed) to higher-level semantic streams that capture application-level concepts and entities (e.g. specic people in the room). The building blocks of the Satware middleware consist of four key modules:

- Query Processor - Applications pose continuous queries to the Query Processor module which in turn selects a set of virtual sensors to provide answers to the continuous queries and forwards this set of virtual sensors to the scheduler module.

- Execution Module - The execution module maps in turn, operators corresponding to these virtual sensors, for execution on physical nodes (machines) in the underlying pervasive computing infrastructure. A special type of operator, a source operator, is responsible of encapsulating the physical sensor and generating observations which are later being processed by up stream operators. For example, a source operator can encapsulate a video camera located at the entrance of the building. Observations by this source operator, for example, the entry of a person to the building are processed by upstream operators according to the execution plan. The source operator is controlled by the scheduler which in turn decides which source operators to probe for observations. We address

the data collection problem in Chapter 5 by creating a probe plan which dictates which source operators to probe based on resource constraints.

- Monitor - A monitoring module captures dynamic attributes of the underlying infrastructure (e.g. event occurrences, resource availabilities); the monitored information is used to enhance the performance, robustness and scalability of the system. In Chapter 7 we describe how such dynamic attributes are being used to perform re-calibration of extraction operators.

- Scheduler - The Scheduler Module combines the events captured by the Monitoring Module with system semantics to direct data collection activities (See Chapter 5). For example, an increased level of occupancy in a certain region (as captured by motion detectors) can be used to trigger a specic video camera that can capture the activities in that region. Furthermore, based on resource constraints, the scheduler determines the specics of the sensor data collection plan, e.g. the resolution and frame rate at which the video data must be captured (See Chapter 6).

This thesis focuses on the Scheduling module and address last three challenges outlined for sentient systems, namely: scalability, adaptivity and robustness. Performing optimization at the physical plan level requires and abstraction model for sentient space applications. Next we describe such abstractions.

2.3 Abstraction of Sensor Model

There have been attempts to represent sensors as abstract entities to enable building sensor based systems. These models try to abstract and standardize a sensor model thereby enabling different sensor applications to seamlessly exchange sensor data.

In Subsection 2.3.1 we describe such a sensor abstraction model - SensorML. The limitation of this standard is that it assumes that users and application writers are able to specify their needs by describing the sensors and operators required for the task at hand. It does not address middleware level challenges such as the need to represent high level concepts, such as entities and activities, which are much more natural for application designers to reason about. Furthermore, the specification does not address adaptivity challenges, hence assuming that there are no resource constraints.

To address this challenge Satware has developed a programming abstraction for sentient spaces which we describe in Subsection 2.3.2. Satware abstracts specific sensor formats, data types, and the representation of sensor readings so that developers can build applications at the logical level without specific details about acquisition process the specific format data is represented by the sensor. Satware suggests a conceptual model for sentient spaces in which applications are specified at the conceptual level abstracting the raw sensor streams from application writers. This enables optimizations at different levels, for example the middleware, can be utilized to address the adaptive data collection, sensor actuation and re-calibration challenges.

2.3.1 SensorML

Sensor Modelling Language (SensorML) [7] is an XML-based modelling language. SensorML provides the mechanism for describing the whole range from simple sensors to arbitrary complex sensor systems and its correspondent platforms. Each sensor is modelled as an operator that is an integral part of a system. Operators consist of input and output behaviour, i.e. describe the stimulus received by the sensor and its subsequent action, additional parameters and the input-output transformation

function. By this, SensorML meta-data provides answers to the questions, 1) what is measured (phenomenon); 2) how is it measured (calibration, quality); 3) where is it measured (geometry, spatial response, and sampling); 4) when is it measured (temporal sampling, impulse response); and 5) why is it measured (target application, future processing).

2.3.2 Sensor Abstraction in Satware

Functional requirements of applications are defined at a highly abstract level which does not include the raw data from sensor streams. Functional requirements deal with concepts such as people's attributes (e.g., *location*) and relationships among people and objects (e.g., *nearTo*). This section focuses on extending the entity-relationship diagram to model the state of the sentient space and the data needs of the pervasive applications, selecting a query language for continuously querying the state of sentient spaces, and describing the mechanisms to translate queries expressed at the level of abstraction of the application to successive transformations on sensor streams.

Observable E-R. From the point of view of the applications, a sentient space is a physical space in which activities and objects are embedded. In this space, there are 3 types of objects: (1) spatial objects such as rooms, floors, and buildings, (2) people (i.e., human objects) such as Mary, Peter, and Alice, and (3) inanimate objects such as coffee pots, recycle bins, and refrigerators. Each of these objects have attributes such as name, occupancy level, location, salary, level of coffee, and so on. These attributes are either static or dynamic (i.e., they change as a function of time). For instance, name and salary are static whereas location is static for spatial objects but dynamic for people. We call *observable attributes* the subset of attributes that can be sensed by the sentient space.

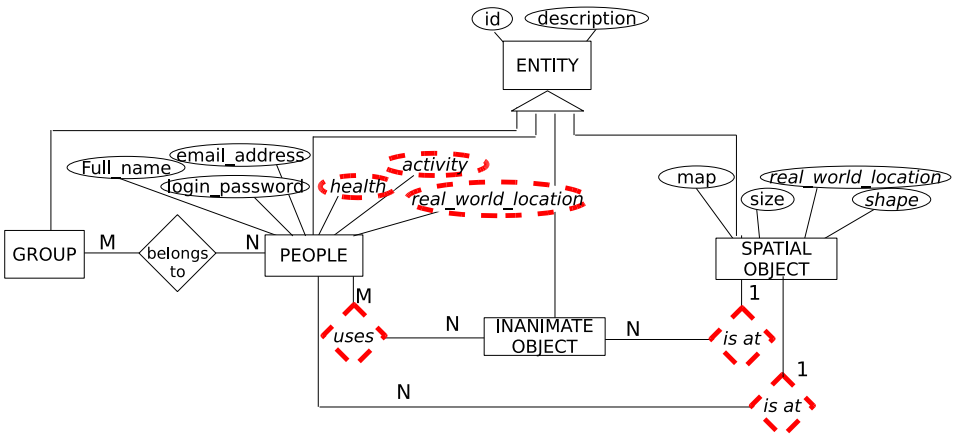


Figure 2.1: Sensor Abstraction in Satware

To model a sentient space, Satware extends the (Enhanced) Entity Relationship Diagram (a *de-facto* standard for designing databases) with new data and relationship types: observable attributes and observable relationships. These extensions are pictorially denoted using dashed circles and dashed rhombuses and refer to the modified diagram as the Observable Entity Relationship Diagram (OERD). Figure 2.1 depicts a generic OERD for a sentient space. In Satware this diagram is referred to as the *Generic OERD*. If necessary, the Generic OERD can be extended for each application to add new roles that people take (e.g., students and professors), inanimate objects (e.g., backpacks), and spatial objects (e.g., meeting rooms and kitchens). This extension is achieved by adding new relationships as well as new entities that are a specialization of either the entity people, the entity inanimate object, or the entity spatial object.

The OERD is translated to the relational model (i.e. tables) applying the same standard procedures that are used to translate an Entity Relationship Diagram to its relational model. Entities become tables with one column per attribute, N:M relationships become tables with one column for each entity pair taking part of the relation, and so on. This modeling allows Satware to support queries posed in an SQL-like language with streams extensions. Applications query Satware as they would

query a database. Continuous queries are posed on the relational model (derived from an OERD) following an SQL-like syntax with sliding windows from CQL ([4]). For example, "*Select Peter's location*" is expressed as:

```
SELECT NAME, LOCATION FROM PEOPLE WHERE NAME='PETER';
```

2.4 Focus of this work

The algorithms developed in this thesis, implement Satware's source operator. The source operator is a software component which encapsulates the physical sensor for Satware. Each source operator in Satware handles a single physical sensor and generates observations which are used by other up-stream operators. In this thesis, we study the following questions: How to dynamically choose which source operators to use at any given time given resource constraints? How to actuate sensors in real time and control their sensing parameters? How to detect recover from erroneous observations generated by a source operator requiring re-calibration?

The guiding theme for this thesis is the usage of semantics for different tasks of event detection. Accordingly, in this section we cover the most relevant work for each of the tasks: We cover related work on data collection from sensor networks in Section 2.4.1. We cover the related work on camera actuation in Section 2.4.2. Finally, we cover the related work on re-calibration in Section 2.4.3.

2.4.1 Data Collection

The general problem we addressed relates to the problem of **deadline assignment to different jobs** [25] where a global task is to be completed before a deadline and

multiple processing components are available for processing. We can think of the task of event capture as a global task which is to be detected until a deadline, the end of the event. Only a specific node (the one where the event is in the FOV) can help the system meet the "deadline" or in our terminology, maximize event capture we exploit semantics of motion to guide the selection of nodes to probe, by definition nodes are not equivalent processing units.

In the context of multimedia applications, [41] suggest a scheduler for multimedia applications which supports applications with time constraints. It adjusts the allocation of resources dynamically and automatically shed real-time tasks and regulate their execution rates when the system is overloaded. This approach is similar in its motivation to our work, however, we exploiting semantics for the scheduling of data collection tasks.

In the context of video surveillance, [2] propose a new approach known as experimental sampling which carries out analysis on the environment and selects data of interest while discarding the irrelevant data. In the case of multiple surveillance cameras, the paper assumes that all video frames are can be collected for processing at any given time. The motivation is similar to our work since out of the set of available images, the most important ones need to be selected. However, our problem is fundamentally different as we address the problem given a resource limitation which prevents the system from accessing all video frames.

In the context of sensor networks, [68] and [21] describe a prediction based approach used to **utilize battery consumption** in sensor networks. Only when the observed value deviates from the expected value based on the prediction model, by a predefined error bound, the sensor transmits the observed value to the server. The error bound is defined by the application according to different quality requirements across sensors. [28] proposed an online algorithm for creating approximation of a real

valued time series generated by wireless sensors, that guarantees that the compressed representation satisfies a bound on the L_∞ distance. The compressed representation is used to reduce the communication between the sensor and the server in a sensor network.

In the context of sensor networks, Yu et al [68] and Han et al [21] described a prediction based approach utilizing battery consumption in sensor networks. Only when the observed value deviates from the expected value based on the prediction model, by a pre-defined error bound, did the sensor transmit the observed value to the server. The error bound was defined by the application according to different quality requirements across sensors. Lazaridis et al [28] proposed an online algorithm for creating approximation of a real valued time series generated by wireless sensors, that guarantees that the compressed representation satisfies a bound on the L_∞ distance. The compressed representation was used to reduce the communication between the sensor and the server in a sensor network.

In the context of networking, Fu et al [16] exploited time series analysis to model network traffic. They proposed a two-phase information collection process: The first phase was to set a range so that the deviation between the predicted and observed values remained in the range of a given level confidence. Based on the size of the range and the level of confidence, a bound on the sampling rate is determine. In the second phase, the information collection process dynamically adjusted the sampling rate based on the briskness of the incoming traffic. The model was used by Sang et al [48] for network traffic prediction to show how multiplexing can be used to improve traffic prediction.

In the domain of **people and object tracking** [49] and [13] address the problem of tracking when the estimates of location are not accurate, i.e., some of the generated estimates are erroneous due to the inaccuracy of the capturing sensor. The task

in both papers was to reconstruct the actual trajectory of the object being tracked with statistical models, such as Kalman Filters by [8] are utilized to find the most probable trajectory. The kinds of problems that these statistical models are used for try to estimate the location of moving objects, for example tracking planes given a wandering blip that appears every 10 seconds. Work by Song et.al used semantic information to address the problem of inaccuracy feature based tracking algorithms that try to associate subjects seen at different cameras by [55]. Here the challenge is again individual tracking and specifically, being able to associate two images from two different cameras to the same person.

There is a large body of work on **schedule optimization** within the sensor modeling literature [20,23,27,66]. A common approach is to build a probabilistic model of object state and probe sensors that reduce the entropy, or uncertainly in state, at the next time frame. Our application is different in that, rather than reducing the uncertainty in building state, we want to collect images of events (even if, and indeed especially if, we are certain of their occurrence) for further high-level processing. Directly modeling object state is difficult in our scenario because a large number of people move, interact, and enter/exit, making data association and tracking extremely challenging. We take a different approach and directly model the *environment* itself using a simple, but effective, sensor-correlation model.

Modeling people activity was studied by [23]. It was found that a Poison process provided a robust and accurate model for human behavior, such as people and car counting, and showed that normal behavior can be separated from unusual event behavior. In the context of intrusion detection [50] proposed a Markov modulated nonhomogeneous Poisson process model for telephone fraud detection. Scott used only event time data, but detectors based on Scott's model may be combined with other intrusion detection algorithms to accumulate evidence of intrusion across con-

tinuous time. Our work differs as we tried to find event of interest on real time under constraints.

One common task in sensor networks is **topology reconstruction** - [33] explicitly do this in the context of non-overlapping cameras. Our correlation model implicitly encodes such a spatial topology, but also captures temporal dependencies between sensors (Fig.4.1(c)). We know of no previous work on spatio-temporal structure estimation, though we found it vital in our building environment model.

2.4.2 Sensor Actuation

To the best of our knowledge, we are the first to address the problem of actuating a large scale camera network based on unscripted human activities. We accomplish this by using a semantic model of building activities and approximating the expected utility of actions. Our correlation-based model is simple enough to be implemented in real-time, yet powerful enough to capture meaningful semantics of typical behavior. Our action selection process is as fast as a table lookup in real-time.

However, there is numerous relevant and related work which we will now discuss.

In the domain of **people and object tracking** there exists a large body of work, we point the interested reader to the following surveys [15, 67]. The most relevant work to our is by [65] who suggests a POMDP approximation approach to address a single target tracking using a sensor network, where the goal is to conserve sensor battery life by querying only sensors that are likely to improve the location estimate of the target. Although the problem domain and formulation are similar, our approach differs from theirs in several key aspects: first, the scale of the problem in our case is different. [65] assumes that there is a single target in the system which makes the

state space linear with the number of regions. In our case we track multiple targets not just one and address the exponential nature of the state of system as well as the state transition function. Second, [65] assumes that people walk according to a linear gaussian model, while our formulation learns it directly. Third, in our case the policy is computed offline and actions are selected by an efficient lookup in a state-action map. In their case, the utility of action needs to be computed at real time which makes the scheduler less agile and less likely to meet real-time deadlines, even for a single target.

In the context of sensor networks other ways for modeling the problem were proposed, for example for energy efficient data collection from sensor networks, recent work by [14] applies a Q-learning [63] technique to allow each sensor to self schedule its tasks an allocate its resources by learning their utility in any given state. The main advantage of Q-learning is that it does not require a model of the environment. In our case, we would like to utilize the motion characteristics of our monitored space. Furthermore, due to the size of the state space we are prevented from learning the utility of different states individually.

The most relevant theory to our work is on the tractability of POMDP solutions. A number of exact value iteration algorithms have been proposed [10, 44]. None of these alternatives can be used in our context since they are limited to a very small state space. For our needs we are not interested in finding the exact solution. Our approach avoids the exponential graph in state space by approximating the scheduler belief space using a single probability value for each region of the space, instead of a probability value for each state of the world. The difference is linear (in our case) vs exponential. Second, policy search methods have also been used to optimize POMDP solutions [6, 40], Their strength lies in the fact that they restrict their optimization to reachable beliefs [44]. Much like the value iteration techniques, policy search

methods require the belief state to be fully represented and for any reasonably-sized camera network the belief state space is too large. Third, approximate value iteration algorithms were presented. These techniques consist mainly of grid-based methods [30]. They can solve larger problems (90 states [44]) by updating only values at discrete grid points. Our approach is different in the way that it takes into account the specific characteristics of the monitored environment and the reward function that makes our approach practical for this problem setting.

A significant body of relevant work has recently emerged under the term "Network Distributed POMDP" [39, 61]. ND-POMDP literature addresses the challenges that arise when two agents need to work in coordination. As an example, consider an application which is only interested in two frontal images of the same person from two different views at the same time (e.g., to reconstruct a 3D image). ND-POMDP can be used to find an optimal solution for this case. The state of the system as well as the system transition function is represented explicitly which renders DP-POMDP not tractable for the size of the problems that our approach addresses. As part of future work we are interested in addressing cameras with overlapping regions and joint camera reward functions. Applying techniques from ND-POMDP together with our approximation techniques seems like a promising direction.

2.4.3 Sensor Recalibration

Although we were not able to find work that utilized semantics for re-calibration of prediction models, much work has been done on closely related applications.

In the context of tracking patterns of human activity the proposed models were mainly utilized for abnormal event detection, assuming that the underlying sensing infrastructures work correctly. Our results can be used as an underlying model for

better detection on which the abnormal event detection can be detected more accurately. Sweeney et al [58] created a "historical database" for each camera in a set of publicity placed cameras that approximated the number of people present in the camera's view at regular time intervals. The application of the database proposed by the authors was to detect abnormal number of people present (or absent) in a scene. Stauffer et al [56] proposed a probabilistic method for reliable background subtraction. This was used to build a visual monitoring system that observes moving objects on a site and "learns" patters of activity from these observations. These patterns are later clustered and used for the detection of an abnormal pattern of movement of the monitored scene. In the context of systems following MBAC (measurement-based admission control), Grossglauser et al [19] studies the impact of the sliding window size of sampling measurements on the probability that actual values exceed the estimated values.

Modeling people activity was studied by Ihler et al [22]. It was found that the MMPP provide a robust and accurate model for human behavior such as people and car counting and showed that normal behavior can be separated from unusual event behavior.

In the context of intrusion detection Scott(2000) [50] proposed a Markov modulated nonhomogeneous Poisson process model for telephone fraud detection. Scott used only event time data, but detectors based on Scott's model can be combined with other intrusion detection algorithms to gather evidence of intrusion across continuous time.

Chapter 3

Problem Formulation

The goal of this chapter is to develop a formulation such that a variety of challenges that arise in the context of sentient systems can be specified using the formulation. We begin by designing such a model, and then we specify the specific problems addressed by this thesis using the model. Let us first define the key conceptual building blocks of sentient spaces. We have followed many of the concepts from the sensor model language - SensorML [7] when it comes to the sensor model. However, we extend SensorML to capture concepts relevant to sensor based systems, such as scheduling.

In Fig. 3.1 we outline the high-level concepts and the interaction between them. The application specifies a reward that is associated with an observation. The goal of the scheduler is to control and actuate the sensors such that the sensors observe phenomena in a way that maximizes the application reward. To achieve that task the scheduler uses semantics that are associated with a phenomena.

- **Phenomena**: any kind of feature property whose value is amenable to observation or estimation, including physical properties, existence and occurrence assessments, etc. Phenomenon has a type, for example, a person walked in cer-

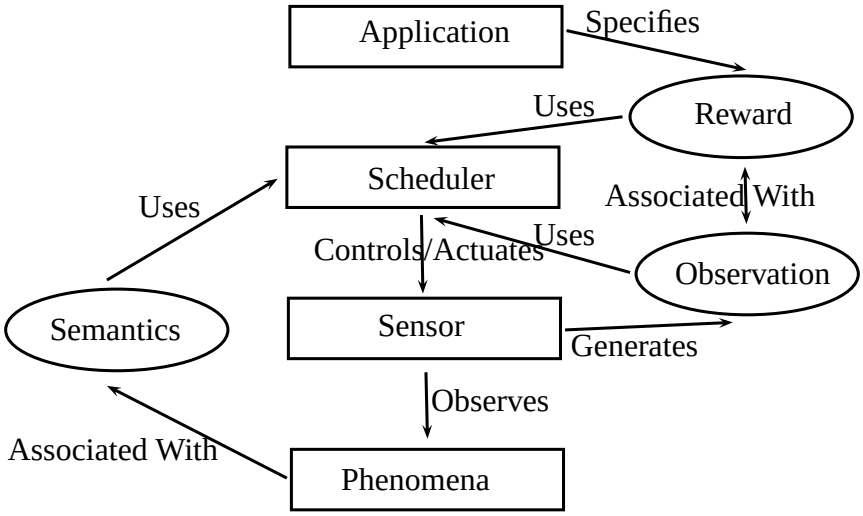


Figure 3.1: Sentient Spaces High-Level Concepts.

tain space, meeting has started in a certain room, number of people in a region. The phenomena happens in the instrumented space and changes over time. The phenomena has a measurable value instance. Almost any phenomena can be captured and represented by discretizing the value range. For example, the level of coffee in a coffee pot can be categorized to one of the following values: "full", "half-full" and "empty". We focus on categorical value instances, specifically we represented the current state of a phenomena as one of a finite number of states - $\Psi = \{\psi^1, ..., \psi^f\}$. The phenomena transitions between states as a function of time - $\Psi(t) = \psi^{i_t}$. In the context of MeerkatU - consider the task of detecting the coffee level at the coffee machine. This is a feature property whose value is amenable to estimation (e.g., coffee level full, empty) and thus fits the definition of phenomenon we are using in this thesis. The coffee level changes due to brewing of fresh coffee and consumption of coffee and thus the level transitions between states as a function of time.

- **Phenomena semantics**: represent the way that the phenomena evolves as a function of time and space. The phenomena semantics that we focus on in this

thesis capture the state transition nature of the phenomena as a function of time and space. Consider the coffee level detection task described above, a full coffee level can transition to half-full, empty and overflowing states. The likelihood of each of the possible future states based on the current and past states gives us a meaningful context and understanding of the monitored system, in this case the coffee machine.

Specifically, in the context of this thesis, we model the phenomena semantics $\gamma = Pr(\{\Psi_{t+1}, .., \Psi_{t+d}\}|\{\Psi_1, .., \Psi_t\})$ is a function from the history of phenomena states to a probability distribution over $\{\Psi_{t+1}, .., \Psi_{t+d}\}$ predicting the states of the phenomena in the next d seconds in the future. We describe the semantics that we have experimented with in details in Chapter 4. The semantics of a phenomena give the application a "window" to the future states of the phenomena in the expected sense. This, in turn, is used to guide the middleware in tasks such as data collection - if it knows that an entity is more likely to appear in a certain camera we can allocate resources accordingly.

- **Environment Model**: creates an abstract representation of the monitored environment. The abstract environment representation allows the modeling of the phenomena state independent of the sensing infrastructure. The environment model creates unique space identifiers for different regions in the monitored space - $r_1, .., r_k$. For example, in a building setting the environment model is used to refer to different regions such as "r_1 = south west hallway, second floor" and "r_2 = second floor kitchen".

- **Sensor**: is the physical device as well as the processing mechanism[1] that is used to observe a phenomena. The system has access to sensors $s_1, .., s_n$ that are controlled by a scheduler.

[1]We treat the sensor and the processing of the raw observation as a single entity, which generates a quantifiable observation of the phenomena.

The sensor is associated with metadata information that is used by the different processes that extract observations:

a) Sensor type. Sensor types include video, motion, temperature, etc. Different sensor types are capable of observing different types of phenomena, for instance, measuring the temperature can be done using a thermometer and not a video camera.

b) Environment Cover - the regions of the space that are covered by this sensor. For example, in the case of a camera sensor the regions covered include the parts of the environment in the field of view of the camera.

c) Sensor parameters - the state of the sensor (e.g., in the case of camera sensor the state can be zoomed in or zoomed out, titled at 65%, etc.). A sensor has a finite number of possible states that control the way the sensor observes the monitored phenomena.

- **Observability of a phenomena**: given a phenomena at a given location and time, a sensor would be able to observe it and hence will be in the observability set of the phenomena if the following four conditions hold: 1) The sensor type can observe the phenomena of interest. 2) The coverage of the sensor includes the location of the phenomena as specified by the environment model. 3) The state of the sensor is such that the phenomena type is observable in that state by that sensor type. 4) The observation parameters are calibrated to extract the observation correctly. For example, collecting a high resolution frontal face image of an entity in MeerkatU, can only be done if: 1) the sensor is of video camera type. 2) the face of the entity appears in a region that is part of the field of view of the camera, as specified by our environment model. 3) the camera is zoomed into the region in which the face is present. 4) The face detection software is calibrated to detect that there is face in the field of view

of the camera. The observability of a phenomena is related to observability in control systems [64] in which the internals of a system need to be observed using external measurements. In our case the internal system is the phenomena and the external measurements are generated by sensors. In our case there is also an environment model that dictates which sensors are able to observe certain regions in the monitored environment.

- **Observation Extraction Process**: The observations are generated following an **extraction process -** D that is controlled by extraction parameters $\omega \in \Omega$. The extraction process is responsible of generating a digital representation of the high level event that was captured by the sensor. Generating the observation might include multiple operators and processes, however, for our problem we abstract that process and consider that the observation process contains an extraction process as well. For example, detecting if there is a face in the field of view requires processing by a face detection operator. We assume that the observation extraction process also executes all the relevant operators to generate the observation of interest to the application. However, we do not assume that the observation parameters are always calibrated and in fact we suggest an approach to detect when the parameters need to be re-calibrated as well as a methodology to automatically perform the re-calibration process.

- **Schedule:** is the way that the sentient system adapts to the way the phenomena evolves over time and allocates system resources for monitoring. The scheduling happens in real-time, as the observations are available such that the following observations are more likely to satisfy the application requirements. Associated with a schedule is the schedule plan cost, which is defined by the cost function. The scheduler controls the capture and processing parameters at time t by choosing an action $A_t \in A$.

- **Cost and Constraints of a schedule** - the cost function is a measure of the cost the plan imposes on the system resources. The cost is usually coupled with a constraint that specifies the maximal system capacity that the scheduler must not exceed. The cost of a plan depends on the cost of data capture by the sensors in their given states. For example, the number of frames that can be processed by the system at real-time is limited. The scheduler uses the cost function and system constraints to decided on the most effective allocation of system resources. The scheduler actions are associated with a cost function $C(A) \rightarrow \mathbb{R}$.

- **Observation of a phenomena**: a phenomena is observed if the schedule contains a sensor that is actuated to a sensor state that is capable of observing the phenomena and the phenomena is observable by that sensor at the time of the schedule. In MeerkatU, an entity has to be present in the field of view of a camera at the time that the scheduler calls for an observation for that camera to be able to extract and observation of the entity. At any time t, given the phenomena state and scheduler action, the system generates an observation $O(\psi_t, A_t)$. Application reward is associated with observations $R(O(\psi_t, A_t)) \rightarrow \mathbb{R}$.

- **Applications** - applications are user defined to monitor phenomena of different types. The sentient system is responsible of generating a schedule plan such that the applications are satisfied. For example, to implement MeerkatU we would define an application that is interested in collecting observations w.r.t faculty in order to enable the real-time notifications being sent to the students, which have subscribed to receive notifications when their advisors arrive to office. Applications are specified in a high level language using continuous queries following an SQL-like syntax, e.g.,"*Select Peter's location*". These queries are

translated to a reward function that is associated with different observations.

- **Reward of an application** - an application interested in a phenomena at a given time, defines the reward associated with the observed phenomena. The reward defines a measure of importance that is associated with different observations to different applications. For example, detecting that there is a large number of people gathering can be done using a low resolution image in which faces can be detected and counted by a processing operator. For this application, the reward of an image containing multiple entities is higher than an image that contains no entities.

Given the above definition we can specify the adaptation problem as follows:

The goal of the scheduling module of a sentient system is to continuously select actions A_t, given history the of observations $\{o_1, .., o_{t-1}\}$ and the phenomena semantics γ to maximize the total reward $\sum_t R(O(\psi_t, A_t))$ such that the cost $C(A_t)$ is within resource constraints.

We assume a 1-to-1 mapping from regions in the environment model to cameras. Thus, each camera covers a different region of the space. Thus referring to a region in our environment is synonymous to referring to a camera that observes this region. We discuss the case in which cameras have overlapping regions in Chapter 8. Using this formulation, we describe the different challenges that we have addressed in this thesis:

3.1 Data collection

- Consider the case in which the scheduler can access only k out of the n available sensor for an image at any given time. The goal is to decide on which sensors to probe for an image and any given time to maximize the total reward associated with the observations collected. **Formally**, we define a plan for N cameras to be a binary vector of length N that specifies which cameras will be probed in the next time instant. $Plan = \{C_i | 1 \leq i \leq N\}$, where $C_i \in \{0, 1\}$.

Our task is not only to know when and where motion was detected but to probe the cameras for frames for a surveillance application or a human operator. Motion is an underlying primitive, that is used to select data that might contain interesting events.

To illustrate how the choice of the scheduler affects a real-time tracking system, consider the following example. Suppose we have $N = 6$ cameras and budget of $K = 1$ probes per time instant. At time t, a frame processed from camera c_j reveals motion. Semantic correlation data collected off-line suggests that motion is expected to continue at c_j at probability $p_{stay}(c_j)$. Suppose further that at time $t - 3$, motion was detected at camera c_k. Semantic correlation data further suggests that given the motion was seen at c_k at $t - 3$, motion is expected to be seen at camera $c_{correlated\_to\_k}$ with probability $p\_correlation(c_k, c_{correlated\_to\_k}, 3)$. This probability is a function of the time elapsed and the two cameras in order: from, to. At the same time, the semantic information suggests that motion can start spontaneously at c_{spont} with probability $p_{spont}(c_{spont})$.

These probabilities provide estimates as to where motion is expected (we will see how they can be computed shortly). The actual sub-task selection should be performed based on the specific benefit/cost functions defined. Note that decisions are made based on partial knowledge of where motion is since we only know what we have

"seen".

3.2 Sensor actuation

Consider the case in which the scheduler can control all cameras and actuate them to exist in a zoomed in our zoomed out configuration. The goal is to decide on which sensors to actuate to collect as many a high resolution facial images as possible.

Our monitored space is divided into N disjoint regions. At (discrete) time t the state of the world is assumed to be in some state X_t. X_t is represented as a binary vector of length N where $X_t^i \in \{0, 1\}$ indicating the presence of a face in region region i at time t or lack of thereof[2].

The environment transitions (stochastically) between states as characterized by the state transition function $P(X_{t+1}|X_t)$ (transitions are attributed to activities that take place in the space, e.g., people walking, talking, meeting, etc.).

The physical space is monitored by a network of cameras $Cam = \{1, ..., K\}$, each of which can observe certain regions that are specified by the cover function -

$$C(r, j) = \begin{cases} 1 & \text{region } r \text{ is covered by camera j.} \\ 0 & \text{Otherwise.} \end{cases}$$

We write $A_{t-1} \in A$ for the action plan that specifies the PZT actuation state for each camera at the next time step t. We write $O^r(X_t, A_{t-1})$ for the observation at time t at region r given the world is in state X_t and the cameras are actuated according to

[2]We'll use superscripts to denote a region in space and subscripts to denote time.

A_{t-1}:

$$O_t^r = \begin{cases} Sc(X_t^r, high) & \exists j \in Cam | C(r,j) = 1 \\ & \text{and } A_{t-1} \text{ calls for } j \text{ to zoom in } r. \\ Sc(X_t^r, low) & \exists j \in Cam | C(r,j) = 1 \\ & \text{and } A_{t-1} \text{ calls for } j \text{ to zoom out.} \\ \text{No Observation} & Otherwise \end{cases}$$

Where Sc, denoting the scheduler observation is defined as:

$$Sc(s,r) = \begin{cases} \text{High Resolution Facial Image} & \text{s=1 and r=high} \\ \text{Low Resolution Facial Image} & \text{s=1 and r=low} \\ \text{No Event} & \text{Otherwise} \end{cases}$$

Thus, there are four possible values available for O_t^r: high resolution facial image, low resolution facial image, no event and no observation. Note the difference between the last two values - no event stands for the scheduler knowing that there is no person in that region while no observation represents the fact that the scheduler has no information about the state of that region. Scheduler actions cannot affect the environment state, but rather its observation of it - $O(X_t, A_{t-1})$. We assume that there is no observation error, thus an event that is taking place will be detected if it is being watched. However, zooming in on a region is at the expense of observing other regions covered exclusively by this camera.

We define the reward for time t as the sum of rewards over all monitored regions.

$$R(O_t) = \sum_{r=1}^{N} R(O_t^r) \tag{3.1}$$

The reward for each region is defined as follows:

$$R(O_t^r) = \begin{cases} \alpha & O_t^r = \text{High Resolution Facial Image} \\ \beta & O_t^r = \text{Low Resolution Facial Image} \\ 0 & Otherwise \end{cases}$$

This representation expresses a large family of reward functions. It is inspired by the task of recognizing entities using pan-zoom-tilt enabled surveillance cameras. High resolution facial images enable more features to be extracted and thus increase the likelihood of a correct identification. Entities can still be recognized from a low resolution image, but with less certainty [69]. The choice of α and β defines the application utility associated with each observation. For example, if we are 10 times more likely to identify a person using a high resolution facial image as opposed to a low resolution we should set $\alpha = 10 * \beta$

3.3 Sensor re-calibration

Given a set of extracted observations, the goal is twofold: first detect if the extraction process is operating with observation parameters that are out of tune. Second, automatically re-assign the parameters to re-calibrate the observation extraction process.

In this section we consider a monitored phenomena of the following nature: The phenomena state can be one of $\Psi = \{\psi^1, ..., \psi^f\}$ states at any given time t. The state

of the phenomena is determined by an extraction process - $D_\omega(SR_t) \rightarrow o_t$, which extracts observations $o_t \in O$ from sensor readings (SR_t) at time t based on $\omega \in \Omega$, where ω is a parameter of the extraction process.

Our goal is given a extraction parameter $\omega \in \Omega$, observations $o_1, ...,_n$ and phenomena semantics $\gamma = Pr(\{\Psi_{t+1}, .., \Psi_{t+d}\}|\{\Psi_1, .., \Psi_t\})$:

1. Determine the likelihood that the generated observations are not representing the original phenomena.

2. Re-calibrate the detection process by finding a new extraction parameter $\omega_{calibrated} \in \Omega$, for which the generated states are likely to be generated by the monitored system.

Chapter 4

Learning Phenomena Semantics in Sensor Enabled Spaces

In this section we describe how we modeled the phenomena semantics for data collection, camera actuation and re-calibration. For sake of concreteness we focus on exploiting the semantics of human activities in an office environment monitored by a network of camera sensors. The key intuition behind modeling the semantics of a phenomena is that by knowing how it "generally" behaves this can be used in situations in which we need to decide on which parts of the space to allocate our resources, focus our collection or trigger re-calibration.

We note that the exploitation of semantics for data analysis, data interpretation, or, for that matter, system adaptation is by no means a new idea. For instance, semantics are used extensively in the "semantic web" [35] context" to improve search by enabling web page authors to tag web pages with appropriate ontologies or by using machine learning techniques to associate appropriate descriptors. Semantics have often been used extensively for data interpretation (e.g., extracting meaningful

content from images/video [54], for data cleaning and improving data quality [24]. As an example of exploiting semantics at system level, database system often use lower level statistical information as well as correlations to improve on query processing and optimizations [26]. The research described in this thesis can be viewed as a continuation of a similar direction of exploiting semantics for system adaptation. In particular, the semantics covers the low level correlations learnt directly from data and the system we consider consists of a sensor-enriched pervasive sentient space.

In the context of this thesis we refer to the semantics of a phenomena as the state transition probabilities which in turn allow the sentient system an "understanding" of the way the phenomena evolves over time and space. Since sentient spaces monitor people or systems controlled by people, there is a predictable pattern, generated by the people using the space. This predictable pattern lets us reason about the activities that we anticipate will take place in the future. When comparing different scheduling alternatives for data collection, we can reason about the "expected utility" of the different alternatives. Our goal is to learn how the phenomena evolves as a function time and space. For concreteness, we focus state transition probabilities of motion in a building setting.

To give an intuitive example on how semantics can help with these tasks, consider a building setting in which people walk in an instrumented environment. If we know that people likely to appear nearby a location they were spotted a short while ago, that can be used to guide data collection to the places which are more likely to yield an informative observation. The proposed techniques take advantage of the fact that the monitored environment and entities are of a well defined and of highly predictable nature. For example, people tend to walk at a certain pace, walk patterns tend to "flow" from a given point to adjacent points, as opposed to appearing and disappearing at random locations. A system that is being monitored, has certain

patterns of appearance if observed over a long period of time. Since the semantics of the monitored space and systems are intrinsic and are part of their very nature, these tend to be stable over time and can serve as an island of stability and predictability in a world of overflowing information and high uncertainty.

4.1 Motion Semantics

The following sections describe the models that we constructed in order to capture semantics of motion. All of the semantics extracted were learned from training data (see experimental setup section for information about how the data was collected). In the following sections, subscript will denote time while super script will denote camera.

4.1.1 Apriori

Arguably the simplest level of motion semantics exploits the fact that certain cameras are more likely to see motion, and so they should be probed more often. We compute the probability of seeing motion at a camera from training data as follows: we divide the number of time instants where motion was observed by the total time period.

4.1.2 Self-Correlation

Assume the state of a camera at time t is $s_t \in \{0, 1\}$, signifying a motion event or the lack thereof. Let us call the observed motion reading from a camera at time t as $o_t \in \{0, 1\}$. We assume a *Hidden Markov Model* (HMM) [46] of the state and

observations

$$P(o,s) = P(o|s)P(s) \tag{4.1}$$

$$= P(s_1)\prod_{t>1} P(o_t|s_t)P(s_t|s_{t-1}) \tag{4.2}$$

The parameters of the HMM are (SC, B, π) where

$$SC = P(s_t|s_{t-1}) = \begin{bmatrix} a & 1-a \\ 1-b & b \end{bmatrix} \tag{4.3}$$

$$B = P(o_t|s_t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.4}$$

$$\pi = P(s_o) = \begin{bmatrix} \pi_1 & \pi_2 \end{bmatrix}^T \tag{4.5}$$

For simplicity, we use a zero-noise observation model (Eq.4.4), though we could learn these values as well. Here, $a = P(s_t = 0|s_{t-1} = 0)$ and $b = P(s_t = 1|s_{t-1} = 1)$.

Inference (computing $P(s_t)$): Exact inference in an HMM can be performed with the forward-backward algorithm [46]. Since we are concerned only with the current state for online scheduling, we need only to implement the forward stage. This can be implemented with a *prediction* step that predicts $P(s_t)$ from $P(s_{t-1})$ using the dynamic model A, and a *correction* step that refines $P(s_t)$ using the observations o_t (and B) if available. **Learning (estimating SC, B, π)**: One can learn model parameters from noisy data using the Expectation Maximization algorithm [46]. Since our observation model is noise-free, we can learn model parameters directly by frequency estimation. For example, b from Eq.4.3 is set to the fraction of times one observed motion in a camera given there was motion in the previous second (from training data).

4.1.3 Cross-Sensor Correlation

We would like to capture the intuition that, given that motion is observed in a particular camera, the scheduler should probe other cameras that lie on typical walking paths emanating from that camera. A important consideration is that it can take many time units, to move between cameras in a typical network. Hence a first-order markov model no longer applies.

For ease of exposition, we first describe a first-order multi-sensor model, but will extend it subsequently to a M^{th} order model that looks behind (or alternatively, looks ahead) M time units.

First Order Cross-Sensor Model

We would like to capture interactions between N cameras. For now, only consider interactions one second later. Let us write the state of a camera i at time t is s_t^i. The overall state is $s_t = \{s_t^1, \ldots, s_t^N\}$, while the observations are $o_t = \{o_t^1, \ldots, o_t^N\}$. The observation model is naturally written as

$$P(o_t^{1:N}|s_t^{1:N}) = \prod_{i=1}^{N} P(o_t^i|s_t^i) \tag{4.6}$$

where we use the shorthand $o^{1:N}$ for $\{o^1, \ldots, o^N\}$. We use superscripts to denote cameras, and subscripts to denote time. Let us write the transition model at $P(s_{t+1}^{1:N}|s_t^{1:N})$. The first assumption we will make is that motion events occur independently at each camera

$$P(s_{t+1}^{1:N}|s_t^{1:N}) = \prod_{i=1}^{N} P(s_{t+1}^i|s_t^{1:N}) \tag{4.7}$$

This model is an instance of a Dynamic Bayesian Network (DBN). It consists of N separate HMMs, where the state at time t depends on all N previous states (a *coupled HMM*). The conditional probability table on the right-hand-side has 2^N entries, so a naive representation will require both exponential storage and time $O(2^N)$ when making predictions. Clearly this does not scale.

Let α_{ij} be the probability that a person moves to camera j in the next timestep given they are currently at camera i. If we assume that people move independently throughout our camera network, we want to intuitively "add" such probabilities across i to compute $P(s_t^j)$. We use a *noisy-OR* model to do this [43]. Formally, let us compute the binary state with a logical OR: $s_t^i = c_1 \wedge \ldots \wedge c_N$ where c_j are equivalent to s_{t-1}^j but are randomly flipped to 0 with probability $(1 - \alpha_{ij})$. The transition model now simplifies to

$$P(s_t^i | s_{t-1}^{1:N}) = 1 - \prod_{j=1}^{N} P(c_j = 0) \tag{4.8}$$

$$P(s_t^i | s_{t-1}^{1:N}) = 1 - \prod_{j=1}^{N} (1 - \alpha_{ij} s_{t-1}^j) \tag{4.9}$$

The above model has $O(N^2)$ parameters, and so scales much better than a naive representation.

Learning: Assuming noise-free training data makes the hidden states observable, and so we can learn α_{ij} parameters by frequency counting as before.

Inference: Exact inference in a coupled HMM is intractable because the probabilistic model contains cycles of dependance [43]. Nearby cameras will tend to develop correlated states estimates over time, meaning that the joint $P(s_t^{1:N})$ does not factor into $\prod_i P(s_t^i)$. Due to real-time considerations it is difficult to employ typical solutions that computationally expensive, such as variational approximations or sampling-based al-

gorithms [20, 23, 27, 66]. One natural approximation is to assume that the states at the previous timestep are independent, and "naively" apply prediction/correction updates (Eq.6.4) to compute $P(s_t^i)$. In the DBN literature, this is known as the *factored frontier* algorithm [38], or one-pass loopy belief propagation [43].

M^{th}-Order Cross-Sensor Model

Assume we have a M-order Markov model, where the prediction of state s_t depends on the past M states. Again, a native implementation of $P(s_{t+1}^i | s_{(t-M):t}^{1:N})$ would require a table of size 2^{NM}. If we assume that people move independently, this suggests a noisy-OR model of temporal correlations.

$$P(s_t^i | s_{(t-M):(t-1)}^{1:N}) = 1 - \prod_{o=1}^{M} \prod_{j=1}^{N} (1 - \alpha_{ij}^o s_{t-o}^j) \tag{4.10}$$

where α_{ij}^o is the probability that a person moves to camera j o-seconds later, given that they are currently at camera i.

The above model states that if there is no motion activity in any camera in the past M-seconds, there can be no motion in any camera currently. To overcome this limitation, we allow for a probability α_i that a person will spontaneously appear at camera i - this is equivalent to a *leaky* noisy OR. We add in $(1 - \alpha_i)$ as an extra term in the product in (6.5).

Structure learning: We observe that most of the α_{ij}^o dependencies are weak; far-away cameras do not effect each other. To improve speed and enforce sparse connectivity, we zero out the α_{ij}^o values below some threshold. For a given camera i, these prunes away many of the dependencies between $s_{(t-M):(t-1)}^{1:N}$ and s_t^i. This means we are learning the temporal *and* spatial structure of our sensor network, formalized as

a DBN.

M^{th}-Order Model with Extended Predictor Motion Semantics

All of the previous models try to predict motion based only on motion events. However, it can be possible to extract more information from the current camera probed, for example, if the scheduler probed for motion at time t_i as well as no motion at time t_{i+1}, it knows that motion event had just ended. This is a significant piece of information that can be used to predict the beginning of motion at a correlated camera more accurately than based only on the information that there was motion probed at time t_i.

Assume that we have a M^{th} order Markov model where the prediction of the current state s_t depends on the past M states. We first develop the model for the single-camera case. The full transition model is now encoded with a set of 2^M variables

$$\alpha_{i_{1:M}} = P(s_t = 1 | s_{(t-M):(t-1)} = i_{1:M}) \tag{4.11}$$

If $M = 2$, then $a_{i_1 i_2}$ encodes the probability that $s_t = 1$ given that the previous two states encoded no event, a sustained event, an event start, or an event end.

Note that the motion only model is nested within the current model, specifically, when $M = 1$ we get the motion model.

For shorthand, we will write (4.11) above as

$$\alpha_{s_{(t-M):(t-1)}} = P(s_t = 1 | s_{(t-M):(t-1)}) \tag{4.12}$$

For large M, it is impractical to learn the exponential number of α parameters above. One approximation common for binary probabilistic models is a noisy-OR model

$$P(s_t = 1 | s_{(t-M):(t-1)}) = 1 - \prod_{o=1}^{M} (1 - \alpha^o_{s_{(t-o)}}) \tag{4.13}$$

If we fix $\alpha^o_0 = 0$, we can interpret α^o_1 as the probability that there will be motion o time steps from now given that there is currently motion. We show these self-correlation probabilities in Fig 1. This model ignores motion semantics of ongoing/starting/stopping events, which should be useful for prediction. One option for a tractable model exploiting these motion semantics is a noisy-OR defined over pairs of states

$$P(s_t = 1 | s_{(t-M):(t-1)}) = 1 - \prod_{o=1}^{M} (1 - \alpha^o_{s_{(t-o-1)} s_{(t-o)}}) \tag{4.14}$$

For example, we can interpret α^o_{10} as the probability of motion occurring o steps from now given that a motion event is currently ending. We would expect this to be smaller than α^o_{11}, which captures the probability of future motion given that a motion event is currently on-going.

M^{th}-Order Cross-correlation Model with Extended Predictor Motion Semantics

It is straightforward to extend the exponentially-large M^{th}-order model from (4.12) to a multi-sensor model using a noisy-OR model. One needs a different set of α tables for each pair of sensors ij in the model. Formally speaking, the transition probability model is written as:

$$P(s_t^i = 1 | s_{(t-M):(t-1)}^{1:N}) = 1 - \prod_{j=1}^{N} (1 - \alpha^{ij}_{s_{(t-M):(t-1)}}) \tag{4.15}$$

Here, $\alpha^{ij}_{s_{(t-M):(t-1)}}$ can be interpreted as the probability of motion occuring in camera i given that camera j undergoes the state sequence $s_{(t-M):(t-1)}$. For large M, it is impractical to learn the exponential number of $\alpha_{i_{1:M}}$'s. We can use the same approximation exploited in (4.14) by considering pairs of states:

$$P(s_t^i = 1 | s_{(t-M):(t-1)}^{1:N}) = 1 - \prod_{o=1}^{M} \prod_{j=1}^{N} (1 - \alpha^{oij}_{s_{(t-o-1)} s_{(t-o)}}) \tag{4.16}$$

For example, α_{10}^{oij} can be interpreted as the probability of motion occurring in camera i in o time steps given that a motion event is currently ending in camera j. Because a person must leave one camera before transitioning to another, we would expect α_{10}^{oij} to be larger than α_{11}^{oij}, which is the opposite behavior from the single-sensor model.

Strictly speaking, the pairwise model from (4.16) is more general than (6.5), and so subsumes it. However, a binary state model limits the scheduler's probing behavior,

for example, it cannot behave like RR and probe a different camera each second. We found better results by a combined model:

$$P(s_t^i = 1 | s^{1:N}_{(t-M):(t-1)}) =$$

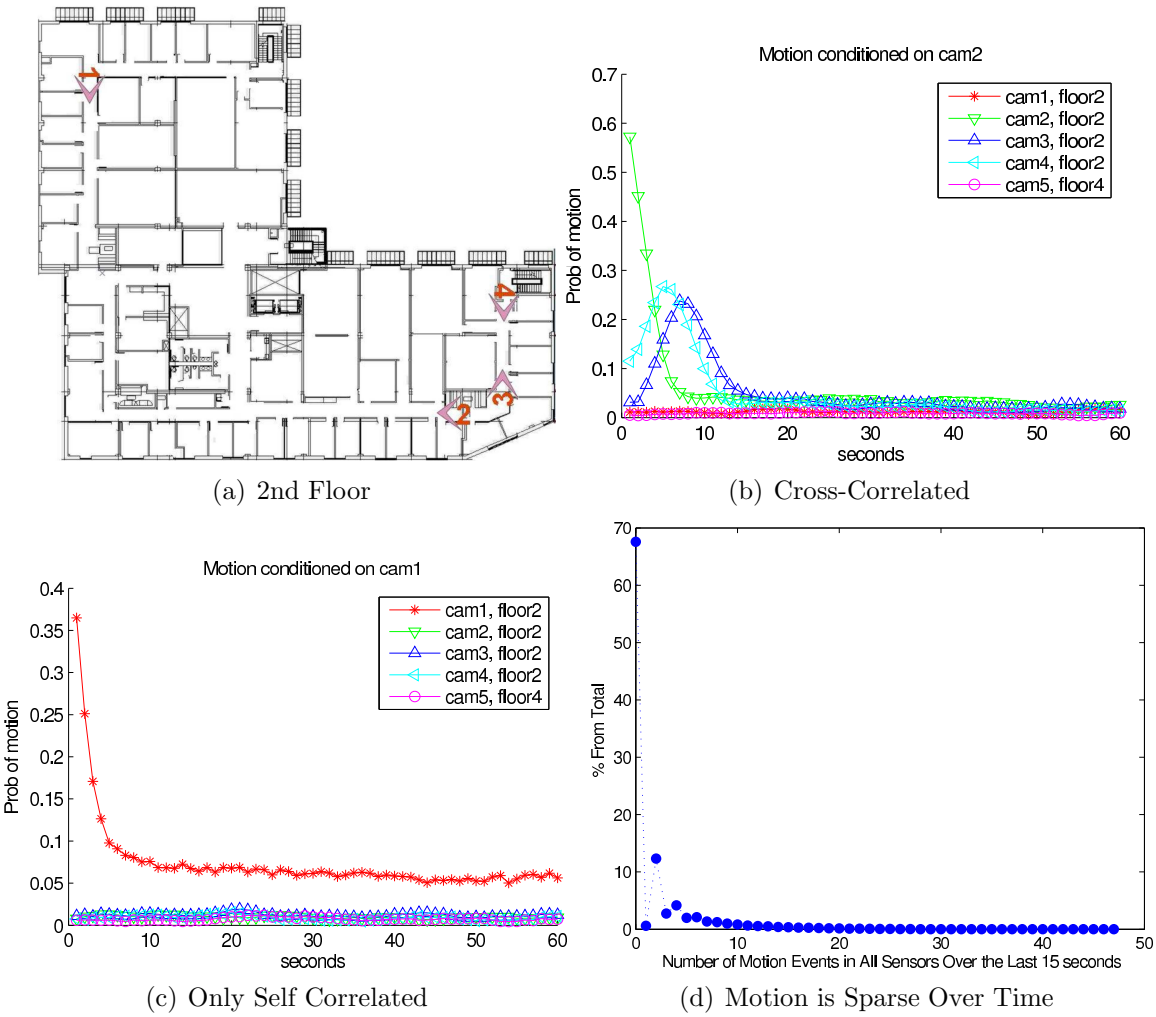
$$= 1 - \prod_{o=1}^{M} \prod_{j=1}^{N} (1 - \alpha^{oij}_{s_{(t-o-1)} s_{(t-o)}})(1 - \alpha^{o}_{ij} s^{j}_{t-o}) \tag{4.17}$$

The above model has the freedom to probe different cameras each time instance and use the binary motion semantics if they are available. The binary motion semantics are available if the scheduler's plan was to stay at the same camera for a time interval greater than 1 second.

As in the single motion model, the above model states that if there is no motion activity in any camera in the past M-seconds, there can be no motion in any camera currently. To overcome this limitation, we allow for a probability α_i that a person will spontaneously appear at camera i - this is equivalent to a *leaky* noisy OR. We add in $(1 - \alpha_i)$ as an extra term in the product in (4.17).

4.1.4 Visualization of the Motion Semantics Collected

The camera layout in the second floor is illustrated in Fig. 4.1(a). Camera 2 is deployed in a short hallway that leads in cameras 3 and 4. This leads to an increased probability of motion at cameras 3 and 4 given motion in camera 2. We plot the conditional probability of motion in all other cameras given that motion was observed at camera 2 in Fig 4.1(b). Note the increased probability of motion in cameras 3 and 4, in the time flowing motion in camera 2. Another possible case, is when only self correlation is present as it is the case for camera 1, the conditional probability of



(a) 2nd Floor

(b) Cross-Correlated

(c) Only Self Correlated

(d) Motion is Sparse Over Time

Figure 4.1: Physical Instrumentation and Visualization of Motion Semantics.

motion in other cameras given motion in camera 1 as a function of time is plotted in Fig 4.1(c). This semantic information is exploited by the scheduler in the following way: Given that motion appears in camera 1, it will estimate motion continuing in camera 1 with high probability. 6 seconds after motion was detected in camera 2, it will estimate motion in the correlated cameras (3, 4), with high probability. A case of such motion pattern is illustrated in the image in Fig 4.1.4.

Fig 4.1(d) plots the average number of average motion events over a 15 second sliding time window. This illustrates several semantics of motion: 70% of the time there is no motion during a 15 second time window. About 30% of the time, there are

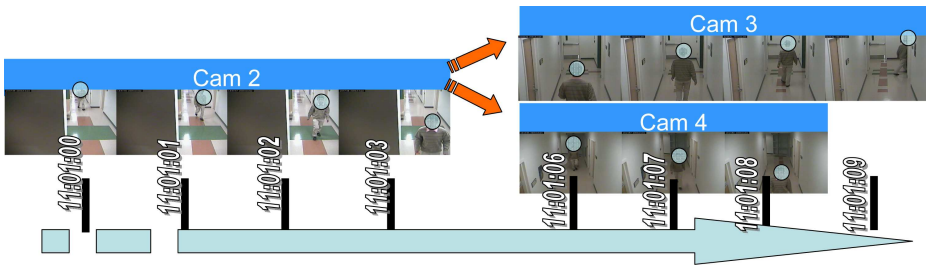


Figure 4.2: A Case Illustrating the Conditional Correlation of Motion.

between 2 to 8 motion events. This teaches us that when motion is detected, there really is a benefit for seeking correlated cameras as motion is rare and when it occurs it doesn't come alone.

4.2 state transition semantics

Many systems of interest and observed environments can be modeled as Finite State Machines. For example, the level of pressure ("Critical","High", "Medium", "Low") on the foundations of a bridge are of crucial importance to maintenance authorities. Real time GPS navigation systems might consult the traffic sensors for average speed of vehicles ("30mph","40mph","50mph","60mph") to determine alternative routes to various destinations.

Four states of interest were specified for the coffee machine in our office kitchen: "Empty", "Half-Full", "Full", "Coffee-Pot Off". Fig. 4.3 shows the four different states of our system. Our initial goal was to detect the state of the coffee machine at all times and notify the subscribers when there was fresh coffee in the pot. Our final goal is to design a robust system by making it resilient to physical perturbations, for example, slight changes in the field of view of the camera, physical displacement of the coffee machine or replacement of camera used for monitoring using semantics. The reminder of this section defines a system as a finite state machine (FSM) and

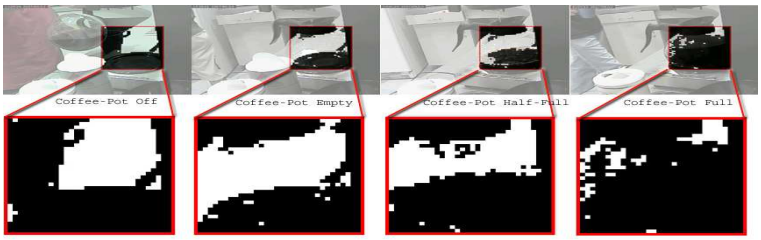


Figure 4.3: The 4 different states in our example. The black/white cells represent dark/bright average pixel color.

the semantic model as the transition times of that FSM.

4.2.1 System Model

The system is modeled as a finite state machine (FSM) of n possible states. Observing a system as a finite state machine implies several assumptions that must be taken into consideration. First, many systems are of a continuous nature and a discreteized representation will lose information. Also, setting strict limits between states, might cause the detection to fluctuate between two states when the actual system state stabilizes around the boundary point of the two states. This will result in an incorrect representation of the system's behavior. Second, choosing the incorrect number of states will reduce the reliability of the detection process. For example, discarding Coffee-Pot off, will potentially result in reduced accuracy of detection since there is a possibility that when the system reaches this state it will be classified incorrectly as one of the other three. Third, in uncontrolled environments, even if the number of states selected expresses the complete set of possible system states, outlier states are possible. Outlier states[1] will cause more "noise" to be added to the detection process.

[1]We refer to outlier states as states that are not part of the normal operation of the system. These states are reached as a result of an unexpected event in the detection processes or system behavior. For example, temporary obscurity in video camera will most probably result in an outlier set of features being captured from the camera.

4.2.2 The Semantic Model

A semantic model for the system is derived by observing its state transition characteristics over time. For example, the average transition time for: Half-Full→Full is 8 minutes and 6 seconds according to the statistics we collected.

Different Time Classes of the Model

Time is broken down into classes. The classes represent varying behavior patterns of the system over several time intervals. For example, the number of people in a conference room, is highly dependant upon the day of week (work day or weekend) and the time of day. (working hours vs non-working hours).

Average and Standard Deviation of Transition Times

For each class C and for each state Si, we calculate the average and standard deviation time it takes to make the transition $Si \rightarrow S_{i+1}$ where $Si \neq S_{i+1}$ and the transition time is in C. The average transition times are used to approximate the probability of a given transition detected to be consistent with the system regular transition times. We expect most transitions to fall in the $\bar{x} - 2\sigma$ and $\bar{x} + 2\sigma$ range. For example, the deviation from the average transition time for the transition: Half-Full→Full is relatively small since it depends on the time it takes the machine to make the coffee which is relatively predictable. We will refer to this statistical model as the *semantic model* of the system from this point on.

Chapter 5

Using Semantics to Guide Data Collection

5.1 Introduction

Consider a camera-based surveillance system deployed throughout a building environment. The task of the system is to process a typically immense quantity of data in real-time, possibly forwarding "suspicious" activities to a human for further inspection. We envision that multiple components are involved in this processing: low-level motion detection algorithms will identify specific regions and cameras of interest, camera sensors must record and transmit images to a central server, and finally, high-level object and activity recognition algorithms should identify suspicious movement patterns that are finally presented to human user. A vital aspect of this processing is that it must be done in **real time**.

Real-time restrictions are particularly limiting because of the enormous computational demands and the resource limitations imposed by a large-scale network. CPU

resources limit the number of frames we can process, and image file sizes limit the amount of images that can be transmitted throughout a network. In this work, we consider a fundamental resource limitation due to network bandwidth: we can only probe K out of N cameras at each time instant, where $K \leq N$. Under such restrictions, we examine: what is the system's *optimal* plan of sensor probes?

Our intuition is that much of the behavior and activity in a building, even for the "suspicious" cases, follows certain low-level semantics - people enter and exit through doors, walk along hallways, wait in front of elevators, etc. We demonstrate that one can build a semantic model of such behavior, and use the model to guide efficient resource allocation for the K available sensor probes.

We look at the planning problem as a sub-task selection problem. The sub-tasks were selected to optimize an application-dependant **benefit function (BF)**. For example, a particular application may want all image frames for which there is motion (all motion events are equally important), while another application may define that two images of two different individuals are more important than two of the same person.

Another consideration is the **cost** of a plan, in terms of network resources, referred to as **cost function (CF)**. Different plans may not cost the same in terms of network resources since it may be less expensive to probe the same sensor at the next time instant. In a fully general model, one might also place the number of sensor probes K into the cost function.

A major motivating application for such an algorithm is a real-time display of image data collected from a network cameras. As the number of cameras is much larger than the number of available screens for display, a subset of the camera feeds need to be selected at any time point based on a user defined, benefit function, e.g., visualize as many motion events in the camera network on a display of k screens. Currently, most

such surveillance systems follow a Round-Robin approach to schedule the camera feeds to display on the available screens.

Formally, we define a plan for N cameras to be a binary vector of length N that specifies which cameras will be probed in the next time instant. $Plan = \{C_i | 1 \leq i \leq N\}$, where $C_i \in \{0, 1\}$.

Our task is not only to know when and where motion was detected but to probe the cameras for frames for a surveillance application or a human operator. Motion is an underlying primitive, that is used to select data that might contain interesting events.

To illustrate how the choice of sub-tasks affects a real-time tracking system, consider the following example. Suppose we have $N = 6$ cameras and budget of $K = 1$ probes per time instant. At time t, a frame processed from camera c_j reveals motion. Semantic correlation data collected off-line suggests that motion is expected to continue at c_j at probability $p_{stay}(c_j)$. Suppose further that at time $t-3$, motion was detected at camera c_k. Semantic correlation data further suggests that given the motion was seen at c_k at $t-3$, motion is expected to be seen at camera $c_{correlated\_to\_k}$ with probability $p\_correlation(c_k, c_{correlated\_to\_k}, 3)$. This probability is a function of the time elapsed and the two cameras in order: from, to. At the same time, the semantic information suggests that motion can start spontaneously at c_{spont} with probability $p_{spont}(c_{spont})$.

These probabilities provide estimates as to where motion is expected (we will see how they can be computed shortly). The actual sub-task selection should be performed based on the specific benefit/cost functions defined. Note that decisions are made based on partial knowledge of where motion is since we only know what we have "seen".

Related work is discussed in details in Section 2.4.1, for the completeness of presentation we briefly describe related work and its relation to ours:

Scheduling under resource constraints for data collection has been previously studied in a variety of contexts [23, 66]; e.g., Markov decision processes, in sensor networks to create an accurate system state representation or for optimizing query performance [**?**, 21]. Such work has also exploited semantics of the environment and/or phenomena being sensed to optimize resources (e.g., powering down sensors and/or reducing the rate at which they transmit). While such previous work is very related to our approach, none of the related strategies applies directly to our real-time camera network setting, as there are several aspects of our work that make it unique. First, we study resource constrained scheduling in the setting where sensors are video cameras which, to the best of our knowledge, has not been studied previously. The nature of cameras and activity these cameras capture provide new opportunities for optimization – e.g., exploiting cross correlation of motion amongst cameras to predict future motion. Second, our goal is to design a principled extensible framework that enables diverse semantics to be incorporated in making scheduling decisions – we illustrate the framework using three semantic concepts (all learnt from data): apriori probability, self-correlation, and cross-correlation. Finally, we illustrate the utility of the technique on real data captured using video cameras deployed at different locations of the building.

We believe that the sub-task selection problem is a central, and frequent, one in multimedia applications [41].

Generally, there is a limit on how many processing tasks can be applied in real-time and an optimal subset of the computation tasks to be selected. In most of these cases, semantic information seems to be a promising way to efficiently direct the available resources.

In order to study the sub-task selection problem, we made several assumptions:

- **Semantics Available** are only of those of **motion**. Although more information can potentially be extracted as semantic information, we focused only on motion.

- **Semantic** system information is learned **off-line** with no budget constraints and is not updated while in operation.

- **Prediction Window** is $t = 1$. Although, it may be important to plan for the next t seconds[1] extending our approach to support this is part of our planned work for the future, and for this work we assume that $t = 1$.

- **Homogeneity** assumption. In order to simplify the algorithms, we assumed that the cameras are homogenous in terms of probe costs. As such, the cost function was always constant. We are leaving the study of the effects of incorporating cost functions to future work.

- **Scheduling Decisions** are made by a (logically[2]) central node which knows the current state of the system and its history.

Push vs. Pull: Note that our approach is based on a pull-based model wherein data is probed from cameras based on a schedule generated by exploiting semantics. An alternate approach could be for cameras to push data whenever an event of interest is observed (push-based model). While push-based models are interesting, they display a few shortcomings in our setting. First, push-based models depend upon cameras to have some computational resources (which is not the case in our infrastructure). Second, such models require basic event detection to be done at the camera which is achievable if events are well defined but more complex in settings such as surveillance where the interest may be in ill-defined events such as "abnormal" event. Of course,

[1]For example, due to network delays, deployment has to be done in advance.
[2]More on this in sec. 5.5

push-models could be based simply on motion detection but then scalability becomes an issue since, depending the usage of the monitored space, multiple cameras may detect motion together leading to network resource saturation. Even in our experimental set up, push based model (which we used for collecting training data) led to significant delays (about 30 seconds in some cases). This is specially the case during periods of high activity (e.g., in-between classes) when all sensors are likely to have motion. The delay caused due to network resources adversely affects the real-time nature of the application. Finally, push-based approach at the motion level is not as robust as the pull-based model we use since false positives such as lighting changes from doors swinging, etc. would result in data transmission. In this chapter, we concentrate on exploiting semantics in a pull-based approach - a hybrid strategy based on combining both push and pull offers an interesting direction for future research.

As we proceed, we will address the following questions:

1. What are the options in selecting sub-tasks?
2. What kind of semantic information needs to be collected off-line, for the case of motion?
3. Which approach proves to work better on real-data?
4. What are the underlying reasons for the observed differences of sub-task selection strategies?

The rest of the chapter is organized as follows: In Section 5.2 we describe our system architecture and the semantics exploited in our framework. Section 5.3 discusses different strategies for sub-task selection. Section 5.4 describes our implementation of the framework and presents accompanying results. We conclude and discuss future work in Section 5.5.

5.2 Our Approach

To study the sub-task selection problem, we postulate a simple model where nodes are cameras and task is to pull frames from nodes (cameras) such that an application defined criteria, such as detect as much motion is maximized. A central server process (following the pull based approach) decides which subset of frames to be analyzed for each time unit[3].

To address the scheduling problem, the approach we adopt is to exploit semantics. While the framework we develop (which is based on probabilistic reasoning) is general and can accommodate any type of semantics, we focus on three different semantic concepts with which to illustrate and experiment. We then show how the system can be extended to accommodate other types of semantics as well.

The semantics of interest included:

1. **Apriori** knowledge of where motion is likely to be. Such semantics can be learned from the data by simply detecting which camera is more likely to have an event occurring. In the case of the building, it is likely that the camera at the front door will see more motion than other cameras. Also, cameras in front of classrooms (or meeting facilities) are likely to observe events more often than others.

2. **Self correlation** of camera stream over time. Given that a camera observes an event and given the camera's field of view (FOV), one could predict the probability that the event will continue. Of course, this is a camera specific property since it depends upon FOV as well as how individuals in the FOV use the space. For instance, a camera focussing on a long corridor will have a person in view for a longer period of time compared to a camera that is focused on an exit door.

[3]We chose second, but other scales were possible to discrete time.

3. **Cross Correlations** between cameras. Clearly a person who exits a FOV of one camera will be captured by another depending upon the trajectory of the individual and the placement of the cameras. Given that a person exits the FOV of camera A, there is an implied probability that he/she will enter into the FOV of camera B.

Given the above, our approach to scheduling is as follows:

1. We learn the above semantics from data.

2. We develop a framework, where given the state of the system currently, we project the state in the near future by determining probabilities of event (motion in FOV). These probabilities are based on all the three semantic concepts above. The three main modules of our framework are:

- **Task Processing Module-** Takes as an input a set of cameras to probe (query plan) performs the querying of the frames from the cameras and returns a list of cameras where motion was detected.
- **System Semantics Module-** Uses a pre-generated log, containing list of cameras and time at which motion was detected at each camera and outputs semantic information in a form of correlation matrix which is used by the scheduler.
- **Scheduling Module-** Uses the current state of the system, i.e., the cameras at which motion was detected, semantic information, BF - Benefit Function, CF - Cost Function and k - Constraint on the number of sensors to probe and returns a list of k cameras to query (a query plan) which maximizes BF and minimizes CF.

The **semantic module** computes the probabilities of the following three events predicted to occur in the future:

(a) $E(apriori(i))$ motion starts at sensor i.
(b)$E(Self-Correlation(i,p))$ motion continues at sensor i, given that there was motion with probability p, at the previous time unit. (c) $E(Cross-Correlation(i,M))$ motion at sensor i, given M - a matrix containing the probability of motion for all sensors for the last t time units.

Based on the probabilities of the above events, given by the **semantic module** and the current state of the system, detected by the **task processing module** the **scheduling module** generates a schedule for the next time unit under the constraints, that minimizes CF and maximizes BF.

We observe that this probabilistic framework is very general. If we had some other semantics we could further incorporate it as long as we can map the semantics to a conditional probability between events over time.

Later on, we will offer other examples of semantics which may be useful. The key technical challenge and contributions of this chapter are as follows:

1. Techniques to learn probabilities from past (as well as current) data.

2. Techniques to schedule based on probabilities. The key issue here is efficiency of computing the schedule without a complex probabilistic analysis.

The next section describes our learning algorithms and the scheduling algorithm followed by an experimental setup, testing and validation.

5.3 Sub-task Selection Strategies

In this section, we propose six strategies for sub-task selection, give their formal definition and algorithm sketch. We consider a system with N sensors $C_1, .., C_N$ and subjects appearing in view of these cameras. A budget of K frames can be processed each second. The statistical attributes were available and were computed off-line, while the real-time attributes are gathered while frames are processed.

First, we presented three ad-hoc, **deterministic** approaches termed as such because the logic they implement is based on ad-hoc intuition of how resources should be allocated.

1. **RR;** A based line algorithm, where regardless of the state of the system, the scheduler probes in a Round Robin fashion.

2. **RR + Stay;** We observed that while motion tended to be continuous, our next algorithm followed this approach and exploited the continuous nature of the signal. It planned to stay as long as motion was detected and when it was not detected at the previous probe, it moved to the next sensor in a round-robin fashion.

3. **RR + Stay + Go to Nearest Neighbor;** We observed that motion is correlated, i.e, motion in one location usually triggered motion in a another location. Thus, when motion was observed, the algorithm would choose to keep probing the sensor. Once the motion ended, we waited for 60 time units for motion in the nearest neighbor[4]. Note that it would be very hard to come up with a systematic algorithm that will capture the correct *semantics* of the system. The ad-hoc values we choose were selected before we know the actual semantics

[4]Each camera has one nearest neighbor

of the motion in the monitored building. As will be shown in the experimental section, this approach proves to perform even worse than the previous alternative. The probabilistic algorithms that we present next follow a systematic approach based on the actual system semantics and prove to achieve a great improvement in detecting motion events under budget constraints.

The following are **probabilistic** algorithms that exploit the semantics of motion at three different levels: probability of spontaneous motion, motion continuing at the same sensor and motion appearing at a sensor after it appeared in a previous sensor. These probabilities, which we refer to as motion semantics, are computed before the actual execution of the algorithm, based on training data.

1. **Semantic RR;** Given the probability distribution of motion at each sensor, we selected a subset of k sensors sampled randomly from the distribution. For example, if cameras i and j were assigned probabilities of 0.1 and 0.2 respectively, camera j is twice as likely to be probed by the algorithm.

2. **Semantic RR+Stay;** Our next Algorithm implements the single-sensor Markov model from Sec.4.1.2. At each time instant, the system maintains a probability of motion at each sensor. This probability can be used, with the dynamic model, to predict the probability of motion in the next time instant. If the scheduler choose to probe this camera, the state is set to 0 or 1. If the probe yields a '1', the predicted probability of motion for the next time instant will be high, and so the scheduler will tend to probe the same sensor until another sensor becomes more likely or, alternatively, no motion is found. Hence the algorithm is naturally interpreted as a probabilistic variant of **RR + Stay**.

3. **Semantic RR+Stay+Correlations;** Our final algorithm with we refer to as Algorithm 1, implements the multi-sensor Markov model from Sec 4.1.3. This

allows nearby cameras to affect the motion prediction for a particular camera.

To implement the prediction step, we need to maintain a MN matrix of probabilities for motion of all N sensors for the last M seconds. To compute the predicted probability for camera i, we multiply the matrix with the correlation values a^0_{ij}. Doing this for all N cameras requires MN^2 operations. We observe that most of the values in the MN matrix of state history probabilities are either zero (if probed) or very close to it[5] and thus their contribution is insignificant. By rounding the state probabilities to {0,1} we optimized this step significantly, as most of the multiplication (and the lookup) operations were avoided when the probability is zero.

Note that the complexity of generating a plan, in this algorithm is $\Theta(N*N*M)$ compared to $\Theta(N*2^{NM})$ required by a naïve implementation that would try to generate the complete state table (see sec 4.1.3). Incorporating the extended predictor semantics is done the same manner discussed above, but instead of only incorporating information about motion or no motion, we also include the event start, ongoing and event end semantics if the scheduler probed the same camera in two subsequent occasions.

While experimenting with the algorithm we identified two pitfalls: The first is in line 15: we choose to probe based on **sampling**, as opposed to probing based on the order of probabilities (high to low). The ordering alternative will prefer probing cameras where motion is probable and might cause starvation in other sensors and thus will perform sub-optimally, see Fig. 5.2.

The second is in **line 6** where we add to the history table (**H**) only motion that was probed. And thus only motion probed is used for prediction. An alternative would be to use predicted probabilities computed in **line 14** for the process of prediction as well. This alternative would cause the scheduler to

[5]As most of the time there is no motion

Algorithm 1: Semantic RR+Stay+Correlations

Data: **P** set of cameras probed at the last time unit. **K** Budget to probe. **H** History of motion probabilities for all cameras in the last M seconds. α- For each pair of cameras and $1 \leq t \leq M$ contains the dynamic model as in eq. 6.5. **current_time** System timestamp (Subtracting two timestamp values gives the difference in seconds.).

Result: **Plan** cameras to probe at time t_i.

```
1  begin
2      /* Correction Step: Update the probability of motion based on what we
       know */
3      for i ∈ P do
4          if p(i).last_state_probed = motion_probed then
5              /* Take into account previous motion seen */
6              H ⟵ H ∪ {i.cam, 1, current_time}
7      /* Prediction Step: Go over the current probabilities and update them (see
       eq. 6.5)*/
8      /* Sensors is the list of Sensors, MP is initialized with the probability of
       motion for each sensor. */
9      for j ∈ Sensors do
10         for i ∈ H do
11             /* Compute the cross-correlation probability */
12             MP(j) ← ∏[1 − j.prob ∗ α_{i.cam,j.cam}^{current_time−j.time}]
13         /* Add the apriori probability */
14         MP(j) ⟵ 1 − (MP(j)) ∗ (1 − α_j)
15     Plan ⟵ Choose_Based_on_Distribution(K, MP)
16 end
```

continue probing sensors for a very long period, even when motion is no longer present, as the following example illustrates:

Assume that two sensors: A and B are correlated, $k = 1$ and motion was present and detected by the scheduler at A for 1 second, and then ended. That motion instance in A will cause the probability of motion in B to increase, as B is correlated to A. At this point, B's probability for motion becomes relatively high. When using predicted probabilities for motion prediction, B's increased probability would cause the probability of motion at A to rise (as it is correlated to B). Eventually this will cause the scheduler to over-estimate the probabilities

of motion at A and B and be "trapped", probing A and B. The reason for this is using probabilities to estimate motion.To further illustrate the problem of over estimating these probabilities see Fig. 5.1, where the actual probabilities computed by a partial run of the algorithm, with $k = 1$ are displayed. Notice that when using predicted probabilities to make decisions, the probability of motion at camera 10 remains high even when motion has ended there. This is since it is accumulation of both self correlation and also based on cross correlation with camera 9. In contrast, in fig. b the prob is reducing since now it does not take into account speculated motion, basing the computation of the probabilities only on motion that was actually probed.

The more accurate probabilities allow the scheduler to visit C_5 instead of wasting more probing resources on C_9 and C_10 and perhaps encounter a new motion event.

Both of those pitfalls apply to the extended predictor semantics as well.

Probe Plan	C_{10}	C_{10}	C_9	C_{10}	C_{10}	C_9	C_9	C_9
Cam9 Projected Prob.	0.11	0.26	0.33	0.51	0.66	0.65	0.65	0.67
Cam10 Projected Prob.	0.45	0.66	0.69	0.79	0.875	0.88	0.9	0.916
time	1	2	3	4	5	6	7	8
Probe Result	+	+	-	+	+	-	-	-

(a) When predicting based on probability for motion.

Probe Plan	C_{10}	C_{10}	C_9	C_{10}	C_{10}	C_9	C_9	C_5
Cam9 Projected Prob.	0.11	0.21	0.25	0.34	0.42	0.43	0.41	0.38
Cam10 Projected Prob.	0.45	0.65	0.56	0.71	0.8	0.67	0.63	0.6
time	1	2	3	4	5	6	7	8
Probe Result	+	+	-	+	+	-	-	-

(b) When basing decision only on probed motion.

Figure 5.1: Actual Probabilities Computed the Algorithm While Executing.

Note the inflated probabilities at the last time unit when predicting based on speculative motion compared to when basing decisions only on motion probed. The affect of this pitfall on the over all recall rate, is illustrated in Fig. 5.2.

We named this phenomena "the gossip motion trap". It can be avoided by not predicting motion based on "gossip" (speculated motion), and only on motion that was actually probed.

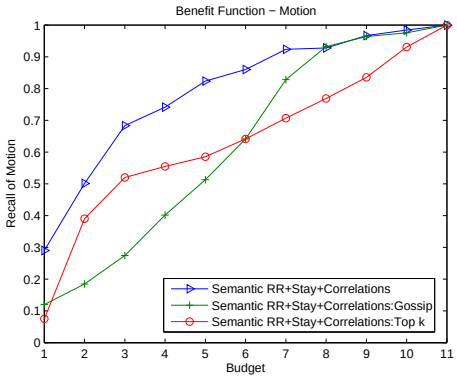
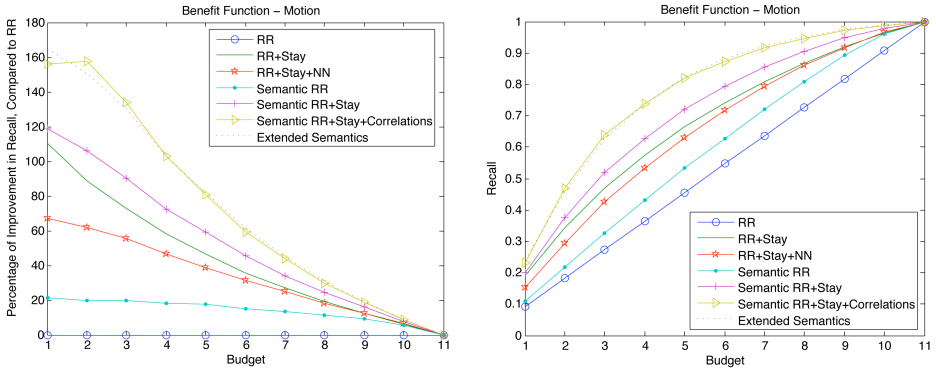


Figure 5.2: The Effect of the two Pitfalls on the Recall Rate.

5.4 Experimental Setup



(a) Improvement Over RR (b) Recall

Figure 5.3: Comparison of the Different Algorithms - Motion Hits.

We evaluated our algorithms on a network of 11 cameras distributed on two floors in our Computer Science building at UCI. The Data we collected was over a period of two weeks (the first and second weeks of the Spring quarter), all cameras were located in hallways. We then used the first week to train for optimal collection of the

other week. The physical location of the cameras is shown in fig 4.1. The cameras were Dlink DCS-6620 with built-in motion detection. The cameras were configured to transmit a time-stamped picture to an FTP server, whenever motion was detected. The time interval was set to 1 second between two subsequent images.

Each floor has a Cisco Catalyst 6500 wired switch-router and each camera is plugged in at 100Mbps (wired connection), switched Ethernet each router is connected to each other at 1Gbps. The FTP server collecting the images has runs proftpd over SunOS with 500MB of memory.

Note that we actually collect accurate ground truth (without resource limitations) data during training and test periods; However, the data was collected with serious delays - this does not prove harmful for testing or training because images are tagged with local camera timestamps. Our end goal is to design a **Real-Time** scheduler which avoids such delays using a semantic model.

Furthermore, each cameras was able to process the images locally in hardware, sending only images containing motion. In a general setting where the sensors don't have these **local** processing capabilities, scheduling becomes a vital issue since a central server must also perform motion detection for each time instant for each camera.

We ran the evaluation on a Windows XP PC running Oracle with Intel Pentium 4, 3.0 Mhz processor and 1Gb of RAM.

5.4.1 Evaluation Methodology

The training data was used to generate three tables: first the probability of unconditioned motion for each camera was computed and stored in a table. The cardinality of that table is 11, one record per camera. Second, for each camera we computed

the self-correlation matrix, as discussed in sec 4.1.2. The cardinality of this table was 11*4, four values per sensor. Finally, we computed the correlation between sensors (as discussed in sec 4.1.3), and stored the correlations in a table of size 11*11*15[6].

To reduce the size of the correlation matrix, we did the following: We computed a threshold probability, $p = 0.05$, and removed all entries of less probability from the table. This value to be visually verified by noting that in Fig. 4.1(c) and 4.1(b), everything below 0.05 is noise. Similarly, we found that it is sufficient to look back only 15 seconds, when considering correlations between different cameras.

The correlation matrix was not to be used for self-correlation, which justified the removal of all entries where a camera was self-correlated. Eventually, the cardinality of the table containing these associations was 165 as opposed to 1815 we started with[7]. The different algorithms were implemented in PL/SQL. We simulated the different algorithms, with varying budget: $1 \leq k \leq 11$ by running it over the test data. Each algorithm generated a plan to probe k sensors based on the motion detected by the previous probes. We counted the number of "hits" of each plan and later computed the precision, recall, and relative improvement in recall over round robin. Precision was computed as the fraction of probes containing motion out of the total number of probes. The recall was the fraction of motion detected out of the whole motion in the system.

5.4.2 Evaluation Results

With a budget of 1, our last two semantic algorithms achieved over 150%(!) improvement over RR (see Fig. 5.3(a)).

[6]which is n*n*M where n is the number of sensors and M is the time window to look back

[7]The table size is 1262 after thresholding in the extended semantics case, as opposed to 11*11*15*4 = 7260. The last 4 represents the four states: event start, event ongoing, event end and motion.

"RR+Stay+NN" performed *much worse* than "RR+Stay". This proves that without exploiting semantics properly, an ad-hoc approach will prove to perform worse than a simple alternative("RR+Stay"), which in this case better exploits the semantics of the data.

The probabilistic approach "Semantic RR" proved to work better then RR but worse than all other alternatives that took into account the continuation of motion. "Semantic RR+Stay" is the "second-best" candidate exploiting more of the available semantics than all other alternatives and proves to achieve about 100% improvement over the best out of the deterministic approaches.

"Semantic RR+Stay+Correlations" and the model exploiting extended semantics dominate the chart and offer a tremendous improvement over all other alternatives, when the resources are constrained.

To give an intuition for the reasons behind this vast improvement, imagine that a sensor falsely detects that there is no motion. The deterministic approaches would choose to go to a different sensor, and will not be able to "recover" from that error. The probabilistic approaches elegantly recover from that situation as the training data teaches the algorithm that sometimes it is a good idea to look back and check again for motion, as the probability for motion will not drop to zero immediately after no motion was detected (see Fig 4.1(c)).

The Model with Extended Predictor Semantics, performed almost the same as Semantic RR+Stay+correlation in this case, when the benefit function is motion. Thus, the extra information didn't improve the performance of the scheduler when all motion events are equally important. We will see a difference in Sec. 5.4.3 when we score the performance of the scheduler based on different benefit functions discussed next.

5.4.3 Experimenting with Other Scoring Types

One might claim that the application might not benefit from multiple motion events of the same person walking down a long hallway, but rather from capturing as many frames from different such individual trajectories.

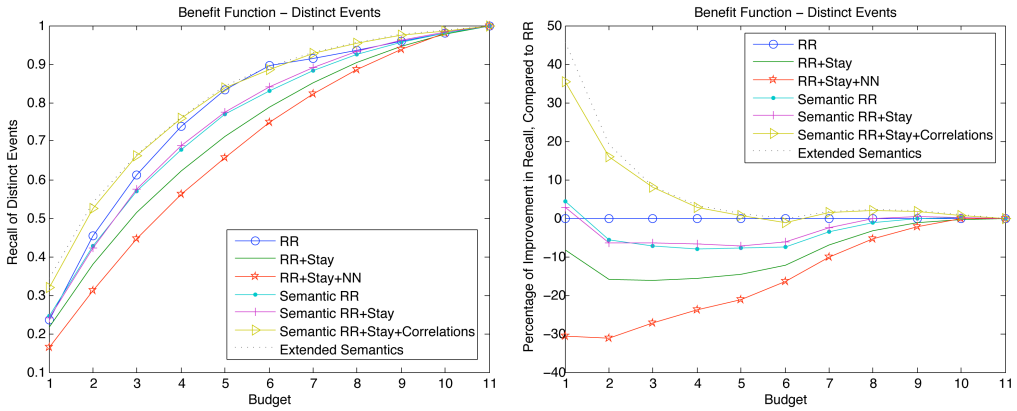
In order to evaluate the performance of our approach in this scenario we created a notion of a continuous event as follows: We ran a connected component analysis over the motion stream to generate continuous motion events. To illustrate, the following motion pattern at a single sensor: ..-++-+++-.. would generate two motion events, of length 2 and 3. Given these motion events we are able to define different scoring functions (benefit functions):

- **Distinct Events** as the number of distinct events that were probed.
- **Motion start time** as the first second of the motion burst.
- **Delay** as the number of seconds elapsed from the start time of the event until the first probe.

Next we report the recall rates based on these different scoring function.

Distinct Events

When the number of different events found is counted (fig 5.4(b)) we can see the benefit of using the model with extended semantics, as the recall rate improves even compared to Semantic RR+Stay+Correlation. The number of distinct events that the scheduler was able to probe was 45% higher compared to RR with $k = 1$. Semantic RR+Stay+Correlation achieved a 34% improvement over RR.



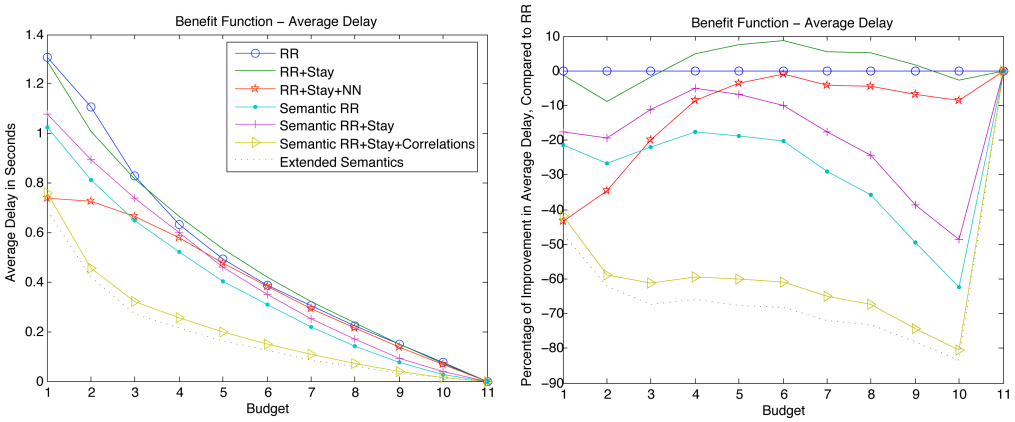
(a) Recall of Different Events (b) Recall Compared to the Recall of RR

Figure 5.4: Comparison of the Different Algorithms - Different Events.

The best Semantic algorithms dominate all other algorithms but the improvement in this case was smaller than what was found in the case of motion - Semantic RR+Stay+Correlation was only able to get 35% improvement over RR with $k = 1$ and 15% with $K = 2$. The reason that RR proves to be a reasonable strategy is that long lasting events, have a very high probability of being probed once by RR. Even with a budget of 1, in our case, RR will hit on all events of length 11 or more. With budget of $k = \frac{n}{2}$ RR will visit every camera every two seconds and achieve very high recall rates (all events of length 2 or more will be captured). This leaves a very small slack for improvement. An improvement of any algorithm over RR, under this scoring function is only when $k = \frac{n}{p}$ and $p = \frac{n}{k}$ is greater than the average length of motion events. In our system events on average last 2.65 seconds, we would expect to see almost no improvement over RR when $k \geq 4.15$. This is supported by our results: as we see no improvement when $k \geq 5$.

Notice that it is possible to do much worse than RR, actually, RR+Stay+NN performs considerably worse than RR. The reason is that this algorithm will choose to continuously probe a sensor (Stay) which serves no benefit to capture distinct events. The "hard-coded" rule of staying in a neighboring sensor until motion begins serves as another factor which decreases the recall of different events.

Average Delay



(a) Average Delay in Seconds

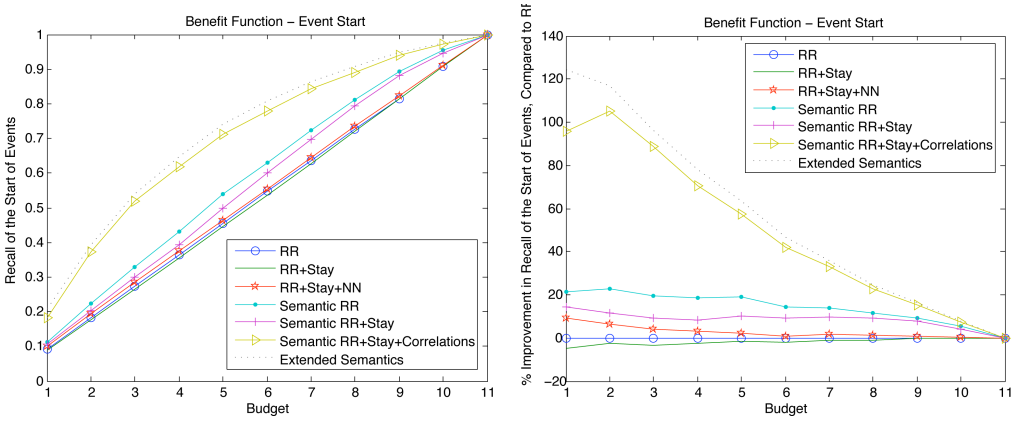
(b) Average Delay Compared to the Recall of RR

Figure 5.5: Comparison of the Different Algorithms - Average Delay.

An Alternative to different events would be to measure the "responsiveness" of the system in terms of the average number of seconds that pass from the beginning of the event until the algorithm probes it. Again, the richer model was able to outperform all other methods, the delay shortened on average on 8% compared to Semantic RR+Stay+Correlation. Semantic RR+Stay+Correlation achieves 40% to 80% decrease in the average delay time compared to RR. The reason is that RR simply **"guesses"** where to go next, while the semantic algorithms incorporate information about where motion is expected to start and probe sensors in an **informed** systematic fashion leading to better responsiveness. RR+Stay+NN is comparable only when $k = 1$, which is explained by it probing nearest neighbors which other algorithms, except the semantic algorithms that incorporate cross correlations, don't. When k increases, RR+Stay+NN quickly loses its dominance and the reason is that its decision making is not accurate, e.g., wait 60 seconds in the nearest neighbor. The best semantic elegantly incorporates all the available semantic information and efficiently exploits it. Moreover, notice that RR+Stay+NN has a much smaller recall rete of distinct events compared to the best semantic algorithm. Although it achieves a good average

response time for $k = 1$, the recall rate of events is smaller. The semantic algorithm with extended predictor semantics dominates the chart as it uses information about the end of events to accurately predict the start of events as opposed to treating all events as motion events.

Event Start



(a) Recall of Events Start (b) Recall Compared to the Recall of RR

Figure 5.6: Comparison of the Different Algorithms - Event Start

The semantic algorithm with extended predictor semantics was able to get a considerable improvement over Semantic RR+Stay+Correlation, over 20% improvement for $k = 1$ and about 8% on average for $k > 1$. Semantic RR+Stay+Correlation achieved about 100% improvement over RR with $K = 1, K = 2$ and a considerable improvement over all other algorithms. This goes hand in hand with the results for number of distinct events and average delay. This result is extremely important for cases where the sensors are located close to entry/exit points as the first frame is expected to contain the highest quality of picture of an individual that just walk in the field of view of the camera.

Experimenting with Other Scoring Types - Summary

Overall, the best semantic algorithm achieves the best results, compared to all other algorithms. Even in the case where RR performs relatively well (different events) the best semantic is still the best choice.

So far we evaluated the algorithm based on different benefit functions, and the decision was always to probe based on where motion is most likely to happen (best semantic). It is part of our planned future work to investigate an alternative approach which will probe the sensors based a function of the predicted probabilities. For example, for different events, it seems that once we saw motion for a particular sensor, probing it again at the next time unit would serve no purpose, as it is expected that the next time unit will contain a motion event - part of the one already detected.

As an example of such function we probabilities are computed as before but are used for probing as follows: for each sensor we estimated the probability that motion in the next time unit is a continuation of a previously observed motion event, $P_{motion\_seen\_already}$. The last can be obtained computing the probability of the HMM associated with that sensor staying at state motion for the time interval since it was last probed to be in motion state.

Next subtract from the computed probability of motion $P_{motion\_seen\_already}$ and use the computed probabilities to schedule the next set of probes. This alternative was able to meet the performance of our best semantic algorithm but not improve on it.

The intuition behind the difficulty of that problem is that probing a sensor again, although not serving the benefit function immediately, is important for accurate state estimation. Since motion events are used to predict future motion events it is important to keep an accurate state estimation - which serves the benefit function indirectly.

The challenge is to design an algorithm that would find the optimal balance between probing for state estimation and probing for the benefit of the application. We will address this challenge in Chapter 6.

5.5 Conclusions and Future Work

In this chapter, we address the challenge of scheduling data collection from sensors to maximize event detection. We designed and implemented a fully functional system using available off-the-shelf cameras to learn motion semantics for our building and evaluated different scheduling algorithms based on non scripted motion over a period of a week. Our semantic based algorithm that takes into account all available semantic information proves that even under significant resource constraints, we can detect a very large number of events.

We followed a pull based approach, however, for future work plan to investigate a hybrid push/pull approach that will benefit from the advantages of computation at the sensors while meeting the deadlines of a real-time system and supporting the dynamic nature of a surveillance task.

We started this work stating five assumptions. In our future work, we plan to relax these assumptions: **Semantics available:** we will extract features from the images, such as number of people in the FOV learn semantic information based on such information and use semantics at different levels (motion, number of people) for scheduling. In this work we already incorporated semantics at two different levels: motion and event type (event start, end or ongoing). **Updating the Semantic Model:** while the system is in operation, up-to-date information is collected about the state of the system. This information can be used to tune the semantic model, for example, due

to changes in the semantic behavior (A new coffee-machine was installed and more people tended to stop for coffee). This problem relates to a problem addressed by the database communitee: estimating query selectivity based on the actual query results [12]. We plan to study the affect of different **Prediction Windows** on the recall rates of our algorithms. The scheduler will be given the state of the system as was "seen" in the last t seconds, and will generate a plan for the next t seconds. At that stage, **Distributed Decision Making** might be taken into account as each node might choose to "stay" when motion is "seen" and further "tip" a correlated node to expect as opposed to being loyal to an obsolete plan. Note that our current algorithms can scale with large number of sensor as semantics are local and several decision making nodes can operate in parallel, e.g., we could have installed a different scheduler for each floor in out building case. Different cost functions that depend on the actual network topology, delays and connection overheads is in our plan to relax the **Homogenous** assumption.

Chapter 6

Using Semantics for Actuation

6.1 Introduction

A distributed camera network allows for many compelling applications such as real-time event detection and monitoring. As an example of an event detection application, consider an office-based social application that automatically detects entities and sends notifications to team members when their colleagues arrive at work, walk into a conference room, have a spontaneous hallway meeting or go out for lunch. The system also monitors the space, identifies unauthorized personnel and alerts security when access policies are violated.

A fundamental building block for such applications is face recognition technology [59]. The challenging aspect of face recognition with regards to a surveillance setting is that the quality of faces captured by surveillance cameras is low. This can significantly increase the difficulty in recognition [69]. Aside from event detection, high resolution facial images are important for forensics purposes.

In this chapter we consider a zoom-enabled camera network collaboratively working to collect high resolution facial images of entities in a monitored space. We design and evaluate a lightweight, real-time scheduler that actuates a distributed camera network to satisfy an application need. The extracted semantics are utilized to predict the future state of the environment and optimize our resource allocation to meet application needs.

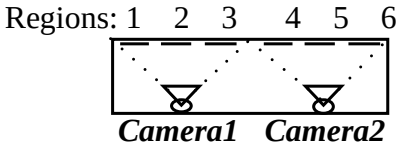


Figure 6.1: A PZT Camera Monitored Hallway.

Due to physical limitations, a camera can be actuated to a single Pan Zoom Tilt (PZT) state. For ease of analysis, we discretize the set of possible PZT states into a collection of two types; (1) a zoom-out state for which a camera can view a large number of regions (but only at small resolutions) and (2) a set of zoom-in states for which a camera views a particular region at high resolution. For example, camera 1 can zoom into a single region at a time - either region 1, 2 or 3 to get a high resolution image or get a low resolution image of all three regions.

Raw camera images are processed and high level events are extracted - "face is detected in region r at time t". This event information is transferred to a scheduler, which in turn controls different camera parameters.

The scheduler processes all past low-resolution and high-resolution observations, and decides an actuation plan for all cameras in the next time step. It may decide to zoom-out all cameras, in which case all regions are observed in low-resolution. Or, alternatively, it can zoom-in to a region of interest and produce a high-resolution image of that region, which might include an entity, if one happens to be present at

the time the region was observed.

Thus, the scheduler is constantly facing a tradeoff between actions that have the potential of generating a high resolution image and actions that reduce the uncertainty of the activities that are taking place. A greedy approach where the scheduler collects only high resolution images is, sometimes, not optimal. This is because such high resolution images provide only a partial observation of the world, due to a limited field of view. Low resolution images provide a coarse but "complete" view of the movement objects in the monitored space, which may increase the ability of the scheduler to collect high resolution images in the future.

Furthermore, the cost-benefit tradeoff between low-resolution and high-resolution images is application-dependent; for example, if the application requires only identifying a male/female label for a person, then a lower resolution image may suffice.

Because we wish the scheduler to plan future camera actuations given an application-specific utility function and a history of partial observations of the world, we use the framework of Partially Observable Markov Decision Processes (POMDPs). POMDPs are a popular approach for modeling sequential decision making in systems operating under uncertainty [37] [10]. POMDP solvers suggested by prior art can find exact solutions for significantly smaller state space. In our case, the size of the problem is vastly larger than what existing techniques are able to solve. Furthermore, even if we were able to obtain a decision policy for our problem setting, using it would require computing a belief state [37] that in our case would be too large to represent[1], let alone compute at real-time.

The key technical challenges and contributions of this work are as follows: first we propose POMDPs as a framework to model actuation in a sensor network that must

[1]The belief state is exponential w.r.t the zoom regions, see Section 6.2.

balance application need and situational awareness (Sec. 6.2). Second, we introduce tractable approximations that are required to scale POMDP solvers for real-time actuation of large-scale sensor networks (Sec. 6.3). Third, we demonstrate the effectiveness of our system, compared to commonly-used baseline scheduling algorithms, on a live camera network in a campus building (Sec. 6.4).

6.2 Problem Formulation

We begin our discussion with a brief overview of Markov Decision Processes (MDPs) [45]. A MDP is a four tuple (S, A, P, R) where S is a **finite** set of states, A is a finite set of actions, $P_a(s, s_{next})$ is the probability that action a at state s would lead to s_{next}. $R(s, a)$ is the reward associated with being in state s while taking action a.

State transitions occur (stochastically) between states based on the actions taken. The goal is to find a policy π which specifies an action for each state $\pi(s)$ which has the maximal total expected reward over some horizon: $\sum_t R(s_t, a_t)$. For instance, in our setting, S would be a binary vector indicating if there is a frontal face of a person in a given camera region. A would be a discrete PZT state (region to zoom into or zoom out) of each camera. P is the state transition probability, which represents the "motion semantics" of how people move in the monitored space. For example, people tend to appear in spatially adjacent regions of cameras as they move. $R(s, a)$ would be the reward associated with being at state s while taking action a. For example, a natural reward would be the number of cameras which are zoomed into a region which contains a person. MDPs can be solved and the optimal policy can be found using dynamic programming [45]. However, in our case, there are several fundamental challenges that prevent us from modeling the space as an MDP: First, the probability transition matrix is too large to even store, as the number of possible world states is

$O(2^N)$; each region can contain a face or not and we have N regions.

A more fundamental challenge is that the true state S of the world is not fully observed. For example, a zoomed in camera has uncertainty about the rest of the regions outside of the feild of view. This means we **cannot** use the previously presented MDP forumation. Rather, we use the framework of Partially Observable MDPs (POMDPs). Crucially, this requires a scheduler to maintain an internal representation that captures the uncertainty about the outside world. Interestingly, one can formulate a POMDP as a MDP whose states S are continuous "belief states" about the world. Belief states assign every possible world state a probability value. This allows one represent uncertainty about the state of regions outside the feild-of-view[2]. In our POMDP, the number of distinct worlds is 2^N, and so each belief state $s \in S$ is a probability vector of length 2^N whose entries are nonnegative and sum to 1. Hence even representing a single belief state is not tractable. For more details on POMDPs, the interested reader is referred to the surveys in [10, 37]. The rest of this section, we formally specify a POMDP for scheduling sensor actuations in a sensor network. In Sec. 6.3 we present extensive approximations which are necessary to implement a realizable, real-time POMDP scheduler for a large-scale network.

Mapping a sensor network to POMDP formulation

Our monitored space is divided into N disjoint regions. At (discrete) time t the state of the world is assumed to be in some state X_t. X_t is represented as a binary vector of length N where $X_t^i \in \{0, 1\}$ indicating the presence of a face in region region i at time t or lack of thereof[3].

[2]For example, in a camera network of 6 cameras where each camera has 4 regions, there will constantly be $3 * 6 = 18$ regions out of the 24 for which the scheduler has high uncertainty.

[3]We'll use superscripts to denote a region in space and subscripts to denote time.

The environment transitions (stochastically) between states as characterized by the state transition function $P(X_{t+1}|X_t)$ (transitions are attributed to activities that take place in the space, e.g., people walking, talking, meeting, etc.).

The physical space is monitored by a network of cameras $Cam = \{1, ..., K\}$, each of which can observe certain regions that are specified by the cover function -

$$C(r, j) = \begin{cases} 1 & \text{region } r \text{ is covered by camera j.} \\ 0 & \text{Otherwise.} \end{cases}$$

We write $A_{t-1} \in A$ for the action plan which specifies the PZT actuation state for each camera at the next time step t. We write $O^r(X_t, A_{t-1})$ for the observation at time t at region r given the world is in state X_t and the cameras are actuated according to A_{t-1}:

$$O_t^r = \begin{cases} Sc(X_t^r, high) & \exists j \in Cam | C(r, j) = 1 \\ & \text{and } A_{t-1} \text{ calls for } j \text{ to zoom in } r. \\ Sc(X_t^r, low) & \exists j \in Cam | C(r, j) = 1 \\ & \text{and } A_{t-1} \text{ calls for } j \text{ to zoom out.} \\ \text{No Observation} & Otherwise \end{cases}$$

Where Sc, denoting the scheduler observation is defined as:

$$Sc(s, r) = \begin{cases} \text{High Resolution Facial Image} & \text{s=1 and r=high} \\ \text{Low Resolution Facial Image} & \text{s=1 and r=low} \\ \text{No Event} & \text{Otherwise} \end{cases}$$

Thus, there are four possible values available for O_t^r: high resolution facial image, low resolution facial image, no event and no observation. Note the difference between the last two values - no event stands for the scheduler knowing that there is no person in that region while no observation represents the fact that the scheduler has no information about the state of that region. Scheduler actions cannot affect the environment state, but rather its observation of it - $O(X_t, A_{t-1})$. We assume that there is no observation error, thus an event that is taking place will be detected if it is being watched. However, zooming in on a region is at the expense of observing other regions covered exclusively by this camera.

We define the reward for time t as the sum of rewards over all monitored regions.

$$R(O_t) = \sum_{r=1}^{N} R(O_t^r) \tag{6.1}$$

The reward for each region is defined as follows:

$$R(O_t^r) = \begin{cases} \alpha & O_t^r = \text{High Resolution Facial Image} \\ \beta & O_t^r = \text{Low Resolution Facial Image} \\ 0 & Otherwise \end{cases}$$

This representation expresses a large family of reward functions. It is inspired by the task of recognizing entities using pan-zoom-tilt enabled surveillance cameras. High resolution facial images enable more features to be extracted and thus increase the likelihood of a correct identification. Entities can still be recognized from a low resolution image, but with less certainty [69]. The choice of α and β defines the application utility associated with each observation. For example, if we are 10 times more likely to identify a person using a high resolution facial image as opposed to a low resolution we should set $\alpha = 10 * \beta$.

Consider the case where $\beta = 0$ and $\alpha > 0$. Collecting low resolution images serves absolutely no purpose to the application. A greedy approach where the scheduler collects only high resolution images is, sometimes, not optimal. Low resolution images provide a more accurate state representation. This, in turn, increases the likelihood of the scheduler to make the right call and satisfy the application needs.

The scheduler that we design will automatically choose the best action to take based on the specific $\alpha, \beta \in \mathbb{R}^+$ values given as input to the process. If, for example, $\alpha \leq \beta$ the scheduler should always stay zoomed out and not take the risk of missing a low resolution image. The "hardest" cases occur when $\alpha > \beta$. In this case, the scheduler must find the optimal balance between zooming in and staying up.

Scheduling as a policy-learning: We wish to find a schedule that determines, for each possible belief state of the system, the best action to take. Recall that each belief state is a distribution over possible world states X_t. In POMDP terms, this mapping between belief states and actions is known as a policy $\pi : S \rightarrow A$. We wish to find a policy that maximizes the expected utlity, or accumulated rewards, over the next T time steps:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E[\sum_{t=0}^{T} R(O(X_t, A_{t-1})|\pi] \tag{6.2}$$

where $t = 0$ refers to the current time. We refer to the horizon T as the "look-ahead" of the scheduler. Intuitively, a schedule obtained with no lookahead ($T = 0$) should choose the "greedy" strategy of always zooming in. On the other hand, a scheduler with non-zero look ahead may sometimes decide to zoom-out, because it may increase the expected reward over the next T timesteps. Our POMDP formulation constructs an optimal schedule by balancing such constraints.

The Challenges of Looking Ahead

Naively computing the optimal policy from Eq.6.2 requires taking into account all possible outcomes of all possible actions for a horizon of T seconds. The problem with this approach is that in order to look ahead for T seconds, the number of states that must be generated is $O((2^N)^T) = O(2^{NT})$. Even for a modest sub-space of 4 regions, looking ahead for 10 seconds is already not tractable ($O(2^{40})$).

Directly applying a POMDP solver is not tractable -

"Off-the-shelf" POMDP solvers are available[4] and they are guaranteed to find the optimal policy. However, they can only solve problems with state space containing about 10 states [52]. None of the available techniques can be applied directly, due to the scale of our problem. We propose one solution which is optimized for the camera actuation problem. In our case, the size of the state space (the space of our X_t states) is exponential compared to the number of regions - 2^N. Thus, even for a modestly sized environment of half a dozen cameras with each covering 4 regions, the state space would be 2^{24} which is 6 orders of magnitude larger than the maximal state space solvable accurately.

This calls for significant optimizations in order to make the problem tractable. We develop an approximate solution, which is optimized for our camera actuation task. Our scheduler exploits the state transition characteristics of human motion in a building and the addive nature of reward functions (Eq. 6.1) of pervasive applications.

[4] see `http:\www.pomdp.org`

6.3 Approach

Our process is divided into two steps: **offline learning** and **real-time scheduling**.

In **Offline learning** we engage in a learning phase in which all cameras are zoomed out[5] and data is collected over a representative period (e.g., a typical day's activity). Once this information is collected, it is processed (offline) for two purposes: first, it is used in order to learn the parameters of the state transition model $P(X_{t+1}|X_{(t-M):t})$. Second, it is used to find a policy π that maps belief states to actions. This policy is created to optimize an application-specific expected reward (see Sec. 6.3.4).

In **real-time scheduling** the scheduler updates its current belief state (using the transition model and any partial observations) and then lookups the belief state in the policy table π in order to select an action A_t (see Sec. 6.3.6). Thus the "heavy" computations are done off-line while the decision making is extremely fast and light weight, as required from a real-time scheduler. A delayed scheduler decision will miss the activity as the real world transitions between states regardless of scheduling delays.

6.3.1 Architecture

Time is divided into equal length intervals (e.g., one second) during which cameras send their newly collected images for processing and observations are aggregated to create the observation O_t at the scheduler. The scheduler decides on an action A_t and sends the it back to the cameras which, in turn, are actuated to observe the space according to A_t at time $t+1$.

[5]If certain regions are not visible by the zoomed out configuration the training data needs to be collected by other means or the transition model be given as input to our scheduler. See Section 6.5 for a detailed discussion.

This ensures that the actions are selected based on the most recent available information, increasing the scheduler's agility and accuracy. The cameras are distributed but our scheduler is centralized. An alternative approach would be to run the scheduler at each camera locally. This would not necessarily make the scheduler more agile as the messaging time is negligible. The advantage of a centralized scheduler is that it can utilize information about activity that is about to enter the field of view of a camera that was observed by a neighbor camera.

In Table 6.1 we summarize the different challenges and our approach to address each one. The rest of the section discusses in details each of the above challenges.

6.3.2 Creating an Approximated State Transition Function

In this subsection, we address the exponential growth of the state space with the size of the camera network (curse of dimensionality) and its exponential growth with the number of past actions and observations (curse of history). We adupt a light weight model that has proven to work very well for modeling indoor human activity [60].

Consider the state transition function: $P(X_t|X_{(t-M):t-1})$. The conditional probability table on the right-hand-side has 2^{NM} entries, where N is the number of regions and M is the number of time units we collect history, so a naïve representation will require both exponential storage and time $O(2^{NM})$ when making predictions. Clearly this does not scale for any reasonably sized monitored space.

Recall that X_t^j is a binary variable indicating if there was an event in region R_j at time t. If we assume that people move independently throughout our camera network, we can intuitively "add" such probabilities across i to compute $P(X_t^j)$. We use a *noisy-OR* model to do this [43].

Table 6.1: Different Challenges and our Approach.

Challenge	Approach	Subsec.
Transition model is exponential $O(2^{NM})$	Assume that people move independently and use Noisy OR model that is linear $O(N^2M)$ (as in [60])	6.3.2
Scheduler belief state requires 2^N dimensions.	Assume that the scheduler belief state can be factored into a belief state for each individual region, which reduces the belief state to N dimensions.	6.3.3
Looking ahead by growing a search tree is exponential in time $O(N^t)$	Represent only certain scheduler states and use dynamic programming (value iteration) $O(NT)$	6.3.4
Hard to select which scheduler states to represent apriori	Based on the properties of our reward function we bound the error and dynamically add points to the representation if the error is not acceptable.	6.3.5
Scheduling decisions must be made at real-time, thus the	Partitioning the space by cameras so that only regions that are in direct competition be in the same partition. Pre - compute an approximated state action grid for each camera with constant space requirements.	6.3.6

Let α_{ij} be the probability that a person moves to region j in the next time step given that they are currently at region i. Formally, let us compute the binary state with a logical OR: $X_t^j = c_1 \vee \ldots \vee c_N$ where c_i are equivalent to X_t^i but are randomly flipped to 0 with probability $(1 - \alpha_{ij})$. The transition model now simplifies to:

$$P(X_t^j|X_{t-1}) = 1 - \prod_{i=1}^{N} P(c_i = 0) \tag{6.3}$$

$$P(X_t^j|X_{t-1}) = 1 - \prod_{i=1}^{N} (1 - \alpha_{ij} X_{t-1}^i) \tag{6.4}$$

The above model has $O(N^2)$ parameters, and so scales much better than a naïve representation.

M^{th}-Order Cross-Region Model - Assume we have an M-order Markov model, where the prediction of state X_t depends on the past M states. Again, a native implementation of $P(X_t^j|X_{(t-M):t-1}^{1:N})$ would require a table of size 2^{NM}. If we assume that people move independently, this suggests a noisy-OR model of temporal correlations.

$$P(X_t^j|X_{(t-M):(t-1)}^{1:N}) = 1 - \prod_{o=1}^{M} \prod_{i=1}^{N} (1 - \alpha_{ij}^o X_{t-o}^i) \tag{6.5}$$

where α_{ij}^o is the probability that a person moves to region j o-seconds later, given that they are currently at region i.

The above model states that if there is no motion activity in any region in the past M-seconds, there can be no current motion in any region currently. To overcome this limitation, we allow for a probability α_j that a person will spontaneously appear at region j - this is equivalent to a *leaky* noisy OR. We add in $(1 - \alpha_j)$ as an extra term in the product in (6.5). We are learning the temporal *and* spatial structure of our sensor network, formalized as a Dynamic Bayesian Network (DBN) [43].

6.3.3 Representing the Scheduler Internal Belief State

POMDP solvers maintain an internal belief state about the external world. This is captured by a probability distribution over all possible values X_t can take. As the scheduler executes the belief state changes. In our case, naïvely representing a single belief state requires a 2^N dimensional vector. Fortunately, the factored inference algorithm of the previous section represents the probability over states as a product of marginal probabilities for each of the N regions. Hence we represent a belief state $\hat{X}_t$ as a N dimensional vector where each entry $\hat{X}_t^j$ is the probability of an event taking place in region R_j at time t.

The Benefits of Looking Ahead

Figure 6.2: Illustrating the Benefits of Looking Ahead.

In order to illustrate the key idea behind the lookahead approach, consider Fig. 6.2. When the scheduler takes into account the long term effects of its actions, it will choose to stay in a states in which the sole purpose is to reduce uncertainty. This allows the scheduler to stay zoomed out and in the following timestamp zoom into the right region for a high resolution image. A greedy approach would always choose to zoom into the most likely region and is more likely to miss events that are taking place due to a limited representation of the current state of the world. The two frames

in Fig. 6.2 where extracted from a real execution of the scheduler with two different scheduling alternatives on the same recording of events.

The Challenges of Looking Ahead

In order to understand the "lookahead" approach let us consider a "toy example": Assume we have a network of a single camera with two zoom regions. X_t in this case is a two dimensional binary vector indicating the presence or absence of a face at time t in each of the regions. For the sake of presentation we assume that the two regions are independent and both regions transition based on the following model: $P(face \mid face) = 0.65$ $P(face \mid no\ face) = 0.05$. Thus, the probability of seeing a face in the following time step given that we see a face right now is 0.65 and 0.05 if no face is observed. The reward function sets the constants to $\alpha = 1\ and\ \beta = 0$, thus we are only interested in collecting high resolution facial images.

Assume that $\hat{X}_t = (0, 0)$ is the current belief state the scheduler is in. Thus the scheduler was zoomed out at time $t - 1$ and was able to observe that there was no motion in either region. In Fig. 6.3 we illustrate the look ahead tree for two seconds. We compute all possible actions the scheduler can take and all possible states it can end up in. For example, if it decides to zoom out (marked as "UP") - the scheduler can observe one of four different outcomes - (0,0) = see nothing in either region or alternatively: (0,1), (1,0) and (1,1). If, however, the scheduler decides to zoom in region 1 (marked as C1) - the scheduler can reach only two possible states - (0,0.05) and (1,0.05). Either the scheduler observes motion in region 1 or not. For region 2 - the scheduler does not get an observation, but based on the transition model, the scheduler will estimate the probability of motion as $P(face \mid no\ face) = 0.05$. From this example we can see how the action that the scheduler chooses to take affects its

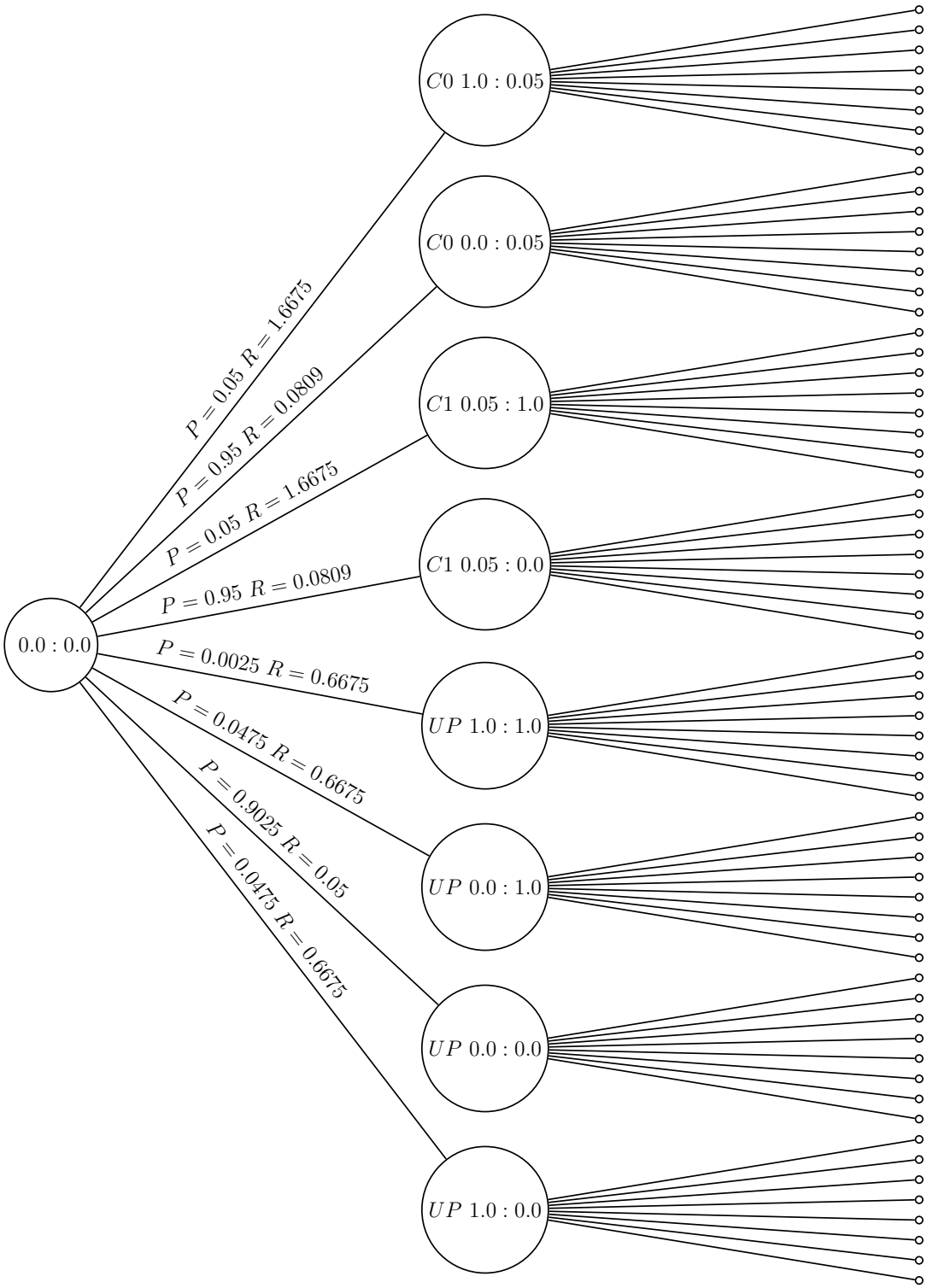


Figure 6.3: Two Second Lookahead Tree for our "Toy Example".

observation of the world. Each scheduler approximate state and action is associated with a reward.

For example, the expected reward of going "UP", taking into account two seconds of looking ahead is computed by taking into account the four possible ways the state of the world can evolve from state (0,0) and multiplying the probability of each state with its expected reward. For example, the transition (0,0) to (1,1) has a probability value of 0.05*0.05 and expected reward of 0.6675 when taking into account all eight possible outcomes that can follow from the system being in state (1,1) and are indicated as the small black dots.

The total expected reward of going "UP" is: $2 * 0.0475 * 0.6675 + 0.9025 * 0.05 + 0.0025 * 0.6675 = 0.112$ while the expected reward of zooming in to "C1" is: $0.05 * 1.6675 + 0.95 * 0.0809 = 0.16023$. Notice that the reward of going "UP" is greater than zero since the look ahead process also takes into account the future reward associated with an accurate state representation. Similar toy example with four zoom regions and $P(face \mid face) = 0.8$ $P(face \mid no\ face) = 0.1$ is the smallest example we could find in which the expected utility of going "UP" is greater than zooming into either of the regions. Given a scheduler state we would like to compute the expected utility of each action for the next T seconds and select the action with the highest expected utility - taking into account both the short term as well as the long term benefits of each action.

6.3.4 Value Iteration

In this section, we temporarily make the simplifying assumption that belief states $\hat{X}_t$ can be represented as discrete elements $s \in S$ such that $|S| = n$. In this case, our POMDP can be written as a MDP with discrete states, and the optimal policy

π can be represented as a table with $O(n)$ entries whos entries can be computed in $O(nT|A|)$ with dynamic programming (where T is the horizon and A is the set of possible actions). The full algorithm, described in Alg. 2, is known as value iteration. We refer the reader to [10] for complete details, but we briefly review it here as our final approximate solution (in Sec.6.3.5) builds upon it.

Algorithm 2: Dynamic Programming Value Iteration

Data: **S-** Set of possible scheduler states. **T-** Seconds of lookahead. **R-** Application specific reward function. **Action-** Set of possible scheduler actions. **Follow-** Returns the set of states reachable from a given state and action. **p-** returns the transition probability between two states.

```
1  begin
2      /* Initialization */
3      for s ∈ S do
4          for t ∈ 1..T do
5              opt[s,t] ⟵ 0
6      for s ∈ S do
7          for t ∈ (T − 1)..1 do
8              for a ∈ actions do
9                  /* Compute the expected reward for a at state s when t seconds
                   remain. */
10                 a_R ← 0
11                 for s_{t+1} ∈ follow(s,a) do
12                     a_R ← a_R +   p(s_{t+1}|s) * (R(s_{t+1}|a) + opt[s_{t+1}, t+1])
13                 opt[s,t] ← max(opt[s,t], a_R)
14 end
```

Alg. 2 computes the maximal expected reward by performing a "backward walk" in time. At time T, the time has already ended (we started from 0, and performed a look ahead of T seconds). Thus the maximal expected utility is zero. At time $t \leq T-1$ the process computes the expected reward of state s for each action $a \in A$ by taking into account all possible scheduler belief states s_{t+1} that can follow from s when action a is taken and looking up their expected reward in the dynamic programming table. The immediate reward of being in state state s_{t+1} while taking action a is $R(s_{t+1}|a)$.

The value stored in the table is that of the action with the highest expected utility *opt*.

One natural strategy is to naïvely apply the above approach using a discretized set of beliefs states S, obtained (for example) by sampling equally spaced grid points along probability vectors. However, future belief states s_{t+1} in line 11 in Alg. 2 will likely not be present in the finite set S. One can use the closest element in S, but this introduces errors that are compounded over time. To address this, we describe an adaptive strategy for iteratively adding discrete states to keep the error within some tolerance.

6.3.5 Adaptive Discretization with Bounded Error

We describe an algorithm for adaptively discretizing the state space S during Value Iteration such that the expected utility *opt* is computed within some user-provided tolerance. This bound is possible due to the additive and monotonic nature of our reward function, which in turn lets us compute upper and lower bounds on *opt* during dynamic programming. We begin by defining two scheduler belief state sets that will help us choose an approximated state:

$$FloorSet(s) = \{sf | \forall_j sf^j \leq s^j \; and \; s \in S\}$$

$$CeilingSet(s) = \{sc | \forall_j sc^j \geq s^j \; and \; s \in S\}$$

The ceiling set of a state s contains all scheduler approximated belief states that have a higher or equal probability of an event across all regions. Similarly the floor set has lower or equal probabilities.

Recall that O_t is the scheduler's observation at time t. Let $Z_t \in FloorSet(O_t)$. Assuming that the reward function is interested in collecting as many events as possible we can say that $R(O_t) \geq R(Z_t)$.

This is due to the nature of the reward functions:

$$R(O_t^i) = \begin{cases} \alpha & O_t^i = \text{High Resolution Facial Image} \\ \beta & O_t^i = \text{Low Resolution Facial Image} \\ \gamma & Otherwise \end{cases}$$

The family of reward functions that we are interested in has $\gamma = 0$. Thus there is no interest in observing an "empty" region of the space.

Reducing the probabilities of the scheduler's approximated belief state will reduce the probabilities of transitioning into any observation O_{t+1} that contains a person. If we repeatedly replace the scheduler's approximated belief state s with $\hat{s} \in FloorSet(s)$, the expected utility that will be computed will be a lower limit on the true expected utility:

$$\forall_{sf \in FloorSet(s)} R(sf) \leq R(s)$$

Similarly, increasing the probability of person being present across all regions, results in computing an upper limit on the true expected utility:

$$\forall_{sc \in CeilingSet(s)} R(sc) \geq R(s)$$

This property allows us to bound the true utility:

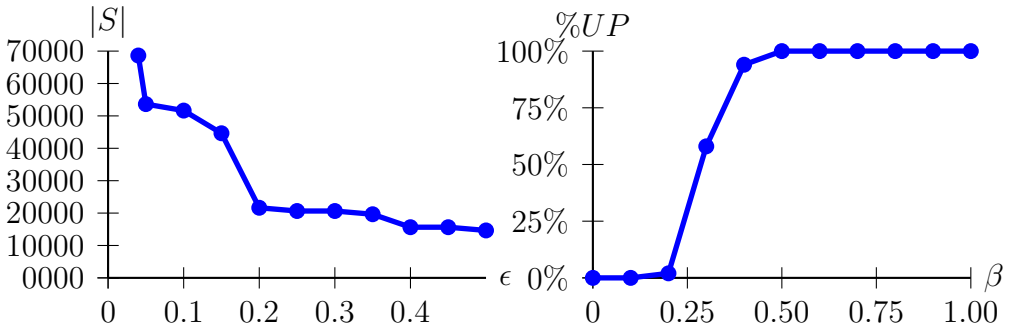
$$opt[sf, k] \leq opt[s, k] \leq opt[sc, k]$$

Whenever we need to approximate state s_{t+1} we take the closest state in $FloorSet(s) - sf$ and the one in $CeilingSet(s) - sc$. Each entry in the dynamic programming table maintains four values: the expected utility when consistently approximating states using their floor set as well as the ceiling set and the actions that were used to achieve the highest utility for the floor and ceiling states.

Before the dynamic programming routine processes the entries for the next time step (as part of the for loop in line 7 of Alg. 2 we verify that the error rates in the entires we just computed meet the desired error level. We can upper bound the error by the following expression:

$$\epsilon \leq opt[sc, k] - opt[sf, k]$$

If the upper limit on the error rate is below the desired error, the process terminates and returns the computed table. In the case where the error is above the user specified threshold, we must add states to our approximation of the state space - S and restart the process. The states are added prioritized based their expected contribution to reducing the error. To accomplish this, we keep track of the states that were missing from the current grid and had to be approximated at line 11 when s_{t+1} was computed. From within this set we iteratively add the top k states (in our case k = 1000). The states are ordered based on their Euclidean distance from the closest state currently in S. The rational behind this process is that once we add a state to the dynamic grid, certain states that were approximated before and introduced significant errors would now be approximated to the newly added state and the error their introduce



(a) Size of the State Grid as a function of ϵ

(b) % of states going "UP" as a function of β (when $\alpha = 1$)

Figure 6.4: The State Grid: Illustrating its growth rate and actions selected.

after this addition might prioritize them after other states. The reason we want to limit the number of states that are added in each iteration is to ensure that we don't increase the state space unnecessarily. Although there might be more than k states that were needed, once we add the top k states we might meet the ϵ required error bound. This allows us to control the growth rate of the state space such that it is within k states of the minimal size needed. Without this iterative selection process the state space explodes with states that are very close to each other and cause the process to exceed memory limits. The prioritization routine allows the process to find solutions for much smaller ϵ values. Moreover this allows the user to specify a timeout condition (as well as ϵ) that will return the best solution (perhaps exceeding ϵ error) that could be found within the timeout time frame. Fig. 6.4(a) illustrates the number of states that are needed for different ϵ values. We start from the "default" grid, which contains 11^4 entries, as each dimension is approximated to the set of points in {0,0.1,0.2,..,0.9,1}. The default grid can find a solution within error bounds for $\epsilon = 0.5$. However, when $\epsilon = 0.45$ the "default" grid has to be extended with $k = 1000$ states so that the dynamic programming process can construct a solution in which all actions have a guaranteed error rate, which is below 0.45. As the tolerated error rate decreases the number of states increases while our process of adding states limits the growth of the grid as close as possible to what really is needed.

Notice that ϵ is a limit on the error in terms of expected reward. Thus, an error of 0.1 means that each action taken by the scheduler is up 0.1 lower in terms of the expected reward when compared to the optimal scheduler.

The problem of finding an optimal scheduler remains not tractable, as a small ϵ will require an unbounded sized state space but solutions are iteratively improving and the processing duration and required state space can be controlled by changing ϵ. Furthermore, in real settings, the number of states that are actually needed in order to get a relatively low error rates is manageable as illustrated in Fig. 6.4(a) were we plot the size of the state space needed as a function of the desired error rate.

Efficiently Approximating s_{t+1}

Whenever a state is approximated we find its nearest neighbor in its floor and ceiling sets. This is done efficiently by using a range query over a KD-Tree. We start with S containing an equally spaced grid. For example, if we approximate a the state space of a camera with 4 regions then $S = G \times G \times G \times G$, where $G = \{0, 0.1, 0.2, ..., 0.9, 1\}$ Whenever a state is approximated we issue a range query from the points of the state to the upper grid lines for the set of its CeilingSet up to the grid lines. From this set we choose the nearest neighbor. KD-Range queries have time compexity of $O(\sqrt{|S|})$.

6.3.6 Distributed Real-Time Scheduling

The above approach, even given the approximations, is still intractable for networks with a large number of cameras. This is because n, the number of discretized states required to achieve a low error, empirically tends to grow exponentially in T[6] To ad-

[6]this is expected as we know that solving POMDP optimally is not tractable.

dress this limitation, we adopted a distributed scheduling approach, where we assume multiple local schedulers each "own" and actuate a collection of camera regions. This requires us to build policy tables for each local scheduler. This factored scheduler is similar to the factored representation of the state model (Subsection 6.3.2).

Because we wish to ensure that regions that are in competition be controlled by the same scheduler, we simply assign a local scheduler to each camera. This allowed us to build smaller (but bounded error) policy tables for each camera.

At run-time, our scheduler updates its beliefs about the world using the transition model $P(X_{t+1}|X_t)$. Beliefs are updated by standard prediction/correction equations [60] for temporal models. Notably, this model is **global** in that correlations across cameras are taken into account. These global beliefs are then marginalized into camera-specific beliefs and used to query **local** camera-specific policy tables π. The prediction/correction equations require $O(N^2)$ multiplications, while the policy lookup consists of a single table lookup per camera, making the overall scheduler extremely lightweight and fast.

To illustrate the action selection process, consider Fig. 6.4(b). We consider a single camera with 4 regions and solve the look ahead problem for 10 seconds up to an error of $\epsilon = 0.3$. We plot the fraction of states for which the best action is to zoom out (go "UP") as a function of β, the utility of a zoomed out event, where the utility of a zoomed in event is 1 (α=1). This table illustrates how the scheduler changes its behavior based on the characteristics of the reward function.

The higher the utility of a low resolution image, the more likely the scheduler is to zoom out. However, when the utility of a low resolution image is considerably lower than that of a high resolution image, the scheduler is highly likely to zoom in.

To illustrate the transition model, consider Fig. 6.5. The figure illustrates the con-

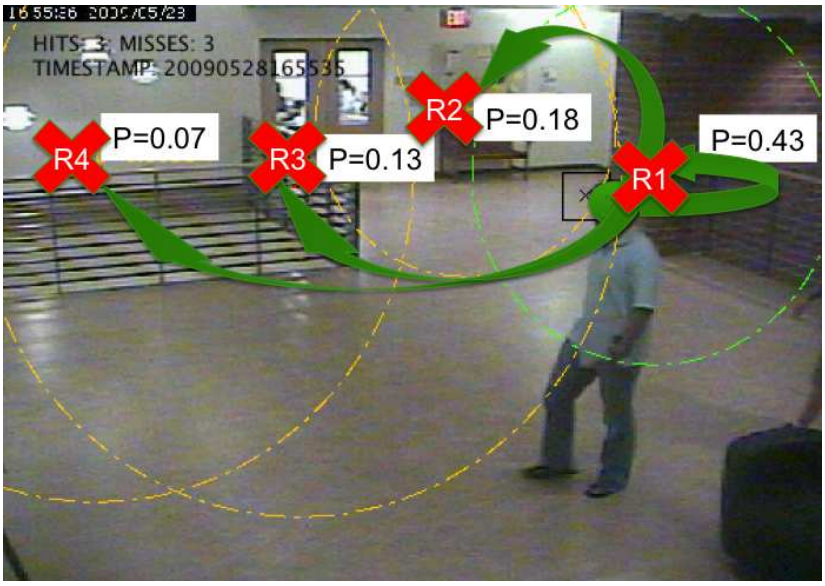


Figure 6.5: Illustrating the conditional probability transition model.

ditional transition probabilities of the face appearing in each of the regions in that camera given its current location. The probability values are computed based on real data that we have collected, as described in the following section.

6.4 Experimentation

Before we evaluate our scheduling approach let us consider the deterministic scheduling alternatives. Following a deterministic scheduler would require us to take into account various space specific characteristics such as the average time a person is likely to stay in a certain region, which region is likely to observe the person next and how much time it will take before the person appears in the next region. It is unclear how this information can be used to best select an action given the scheduler state, in a general setting. The probabilistic approach presents us with a way to overcome these challenges by selecting actions based on their expected utility. We have implemented our approach as well as different "base line" alternatives for comparison.

1) Consider a scheduler that for each camera, cycles between a zoomed out configuration and zooming into the regions covered by it, one by one in Round Robin. 2) Cycles in Round Robin between zoomed in configurations only in Round Robin. 3) Stays "UP" in all cameras at all times. 4) Select an action that has the highest expected utility based on the predicted probabilities. 5) Our approximated proposed approach with look ahead of two seconds. 6) A Tree based approach computing the exact utility by growing a tree of all possible states and actions from a given scheduler state, for each camera. 7) An "Oracle" approach which actuates the cameras to reach the highest possible utility achievable.

Experiment 1 - our approximated approach vs. the tree based, comparing latency

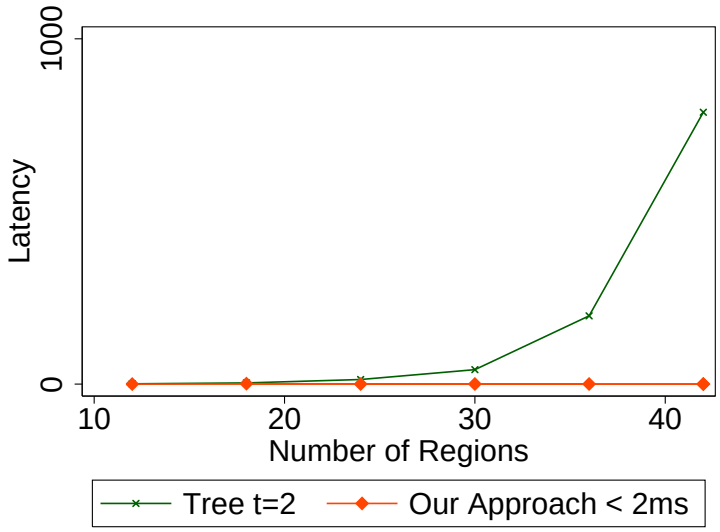


Figure 6.6: The latency of decision making.

To illustrate the impact of computing the exact utility value on the latency of the scheduler, consider Fig. 6.6. We plot the latency in milliseconds of the total time that it takes the scheduler to actuate all cameras, when the decision process is performed by our approach and by a tree based approach. We vary the number of regions in each

camera from 2 to 7 and thus the total number of regions in the camera network varies from 12 to 42. The latency of the tree based approach, even for a modest lookahead of 2 seconds, is not real-time and significantly higher than our approximated approach which takes less than 2 milliseconds for the largest case of 42 regions.

Experiment 2 - the benefit of looking ahead in real world setting

We have evaluated our techniques on a network of 6 cameras which cover the second floor of our computer science building whose fields of view are illustrated in Fig. 6.7. In the figure attached, each camera has 4 zoom regions. The area marked by an X is the center of a region that is likely to observe a face[7]. The region that the camera would zoom in to is marked by yellow circles.

Data was collected for the duration of two days from a single floor in the computer science building, collecting unscripted human activity of faculty and students walking around the hallways of the building. The evaluation was done by replaying the events that took place in the observed environment and computing the total reward that each alternative was able to achieve.

In Fig. 6.8 we compare the four different scheduling alternatives in our camera network where each camera has four possible zoom regions. For different application utility functions in which we increase the utility of a low resolution image. When the utility of a low resolution image is relatively high (over 0.3) the alternatives which are sensitive to the value of β are equivalent, basically reducing to a dimple scheduler which constantly actuates all cameras to zoom out. For the "hard" cases, where the utility of a low resolution image is low, looking a head for 2 seconds out performs all other alternatives and accomplishes the highest total utility. Notice that our ap-

[7]Created based on training data as follows: the locations at each camera that a face was detected are clustered into k clusters and each cluster becomes a zoom region.

Figure 6.7: Experimentation Setup- Six Cameras Actuating to Zoom Regions.

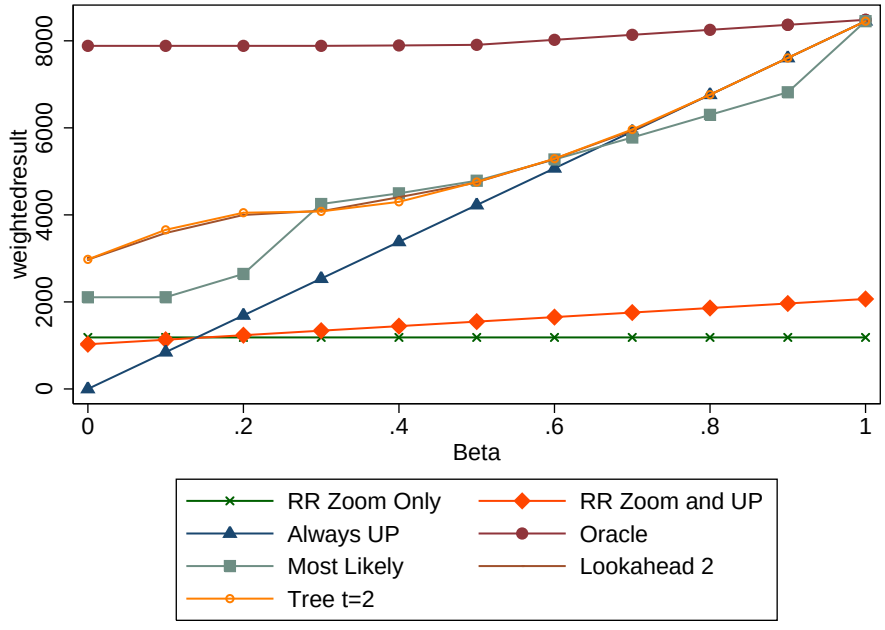


Figure 6.8: Total reward for different alternatives, 7 zoom regions.

proximated approach performs very closely to the exact approach, but with real-time latencies.

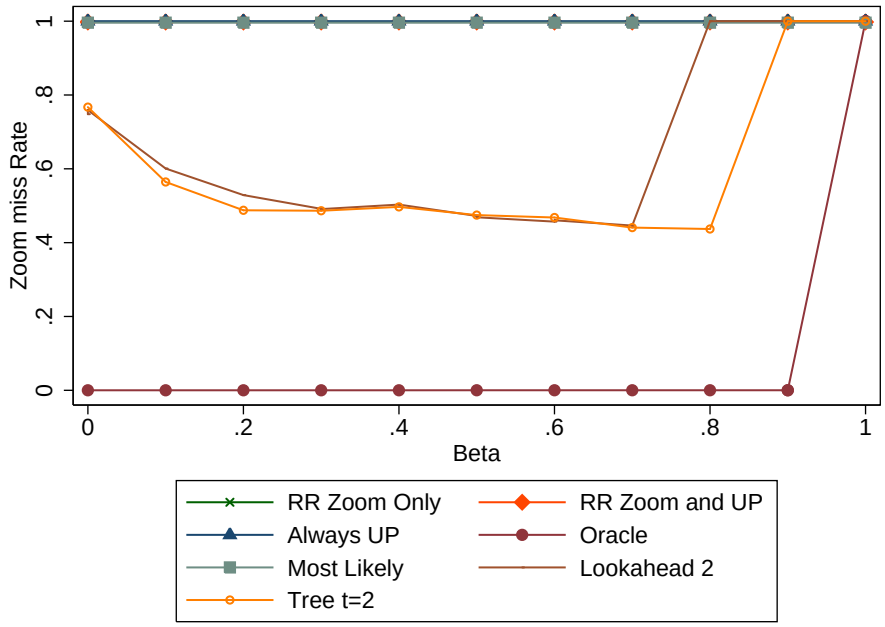


Figure 6.9: Rate of actuations not resulting in a facial image.

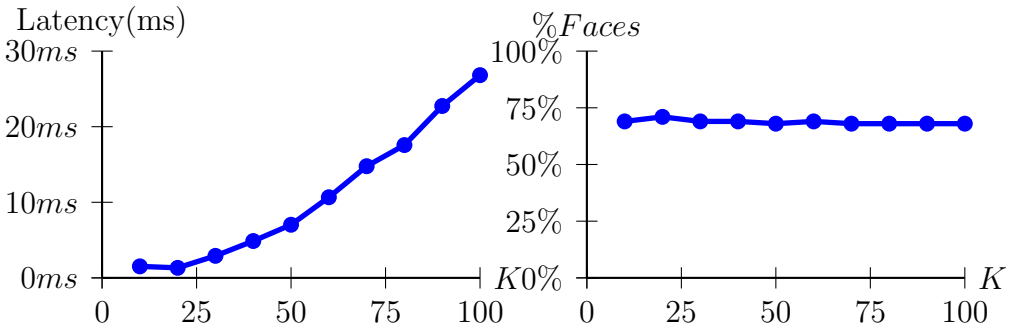
In Fig. 6.9 we plot the rate at which a high resolution image was collected but there was no entity in the region. We compute the ratio of high resolution images with a face over the total number of high resolution images collected. We have defined the rate to be 1 if no high resolution images were collected. This experiment captures the need for a low rate of "false-positives". Since the captured observations are later processed by a heavy operator (e.g., face recognition), we would like to minimize the number of frames that it has to process. The lookahead approach both collects a larger total of observations of interest to the application and reduces the number of false positives as compared to the other alternatives.

Our approximated approach is comparable to the tree based approach and both lookahead alternatives vastly dominate the naive alternatives.

Experiment 3 - scalability in a very large camera network

Our approach scales with the number of cameras as described in Subsec. 6.3.6. To experiment with a very large camera network we have simulated a network of $K = 10$ to 100 cameras, monitoring in a (very) long hallway as illustrated in Fig. 6.1 for the case of $K = 3$ cameras. Each camera has 3 zoom regions and people walk in a single direction from left to right. We have simulated the walk trajectories generated by 10 people who arrive at the entry point of the pipeline according to a Gaussian distribution with mean of 100 seconds and standard deviation of 25 seconds. Each person walk all the way to the end of the hallway in their own walk speed, specified as the time it takes the person to walk from region to region, which is also a Gaussian with mean of 4 seconds and variance of 2 seconds. Our scheduling scheduling algorithm automatically learnt the transition probabilities from a generated script used as training data. We have experimented with an application utility interested in high resolution facial images ($\beta = 0$, $\alpha = 1$). We plot the latency of reaching a scheduling decision as a function of K in Fig. 6.10(a) and the percentage of recalled faces in Fig. 6.10(b). The latency for a network of $K = 100$ cameras is less than 30 ms and linearly increasing with the size of the camera network, which makes the scheduler good fit for real-time applications. The recall of the faces remains high as the number of cameras increases. In the simulated results the recall of events is around 70% due to the fact that in about 1 of 3 cases the person will start moving and the scheduler will miss a single frame until it catches up in one of the following 3 zoom regions.

Future work should consider applying the techniques discussed in this thesis to other application domains, detecting other types of events and using different types of sensors. As a final note, we would like to conclude with overview of our algorithm to address two important problems: actuating cameras with overlapping regions and



(a) Latency of Scheduling as a function of number of cameras

(b) % of faces collected at high resolution as a function of the number of cameras)

Figure 6.10: High Recall Rates and Real-Time Latencies for Large Camera Networks.

optimizing multiple applications. Consistent with our approach throughout the thesis we address these two problems by utilizing the phenomena semantics which let us take into account the future states the phenomena is expected to be at.

Cameras with Overlapping regions

In certain cases, cameras can have overlapping regions. If this is the case, selecting actions for each camera independently might result in a sub-optimal actuation. For example, it choose to zoom two cameras to the same region.

We suggest two ways to overcome this limitation. First, we can schedule several cameras at the same time and find the joint actions which has the highest expected utility. Second, we can select actions for cameras according to a pre-defined order and for each camera choose an action which has the highest expected utlity given the actions that were already selected.

The first alternative is guaranteed to find the best action to take, but is not tractable for most camera networks. The challange is that the number of actions that are possible for a joint of K cameras where each camera has A actions is K^A. Furthermore, all regions that are covered by the K cameras need to be taken into account when

performing the value iteration, and as a result we need to approximate a blief state of a larger dimentionality.

The second alternative is tractable for large scale camera networks and provides an approximated solution. In order to support this approach, the value iteration procedure needs to keep track of the expected utility, not only of the best action, but of all actions. This allows the real time process to select actions acording to Alg. 4.

The main idea is to select the actions, considering the added utility of the action given all the actions that were selected so far. This will enable the scheduler to avoid having two cameras zooming into the same region, for example. Note that this will not take into account the long term effects of actions but only their immediate reward. The first alternative is guaranteed to find the optimal joint schedule taking both immediate reward as well as long terms reward of each (joint) action.

Algorithm 3: Selection actions for cameras with overlapping regions

Data: **ER-** Returns the expected reward associated with taking an action for a state for a given camera given all actions selected so far. **C-** Set of cameras with overlapping regions. **s-** returns the current state for a given camera. **actions-** returns the set of actions for a given camera.

Result: **Plan** of actions selected for each camera.

```
1  begin
2      Plan ← {}
3      for c ∈ C do
4          bestAction ← UP
5          bestActionScore ← 0
6          for a ∈ actions(c) do
7              /* Compute the expected reward for a at state s given Plan. */
8              utility ← ER(s(c), a, Plan)
9              if utility then
10                 bestAction ← a
11                 bestActionScore ← utility
12         Plan ← Plan ∨ bestAction
13 end
```

Selecting Actions to Optimize Multiple Applications

One of the limitations of the current approach is that we need to compute the policy π for each application based on its α and β parameters. In a more dynamic setting there might be several possible applications which any subset of these can be running at the same time.

Formally, given a set of applications Q we would like to find a aduptively change the selected actions based on the current set of registered applications $q_1, \ldots, q_p \in Q$. Doing so explicitely, would require $2^{|}Q|$ different policies π being pre-computed, one policy for each combination of applications currently registered at the system.

To address this, we suggest a more scalable approach in which we dynamically generate the optimal policy for the set of currently registered applications based on $|Q|$ policies computed for each query individually. For each state we "enhance" our policy table from Section 6.3 and store with each state not only the score of the best action but also the score of all other actions. Choosing the best action for the currently registered applications is resolved by computing the total expected utilty of each action according to the policies of registered applications, as described by Alg. 4.

6.5 Conclusions and Future Work

In this chapter, we address the challenge of actuating camera sensors to maximize application utility in terms of high resolution images. We designed and implemented a fully functional system using available off-the-shelf cameras to learn motion semantics in our building and evaluated different scheduling algorithms based on non scripted motion. Our algorithm takes into account all available semantic information and performs a look-ahead. Our approach introduces a significant improvement in the

Algorithm 4: Selection the best action for multiple queries

Data: **c-** the curernt camera to select an action for. **s-** returns the current state for a given camera. **AER-** Returns the application expected reward associated with taking an action from a given state. **Q-** Set of Applications currently registered. **actions-** returns the set of actions for a given camera.

Result: **Action** for camera c which optimizes all current applications.

```
1  begin
2      Plan ← {}
3      bestAction ← UP
4      bestActionScore ← 0
5      for a ∈ actions(c) do
6          currentActionScore ← 0
7          for q ∈ Q do
8              /* Compute the total expected reward for a at state s given for all
                  current applications. */
9              currentActionScore+ = AER(s, a, q)
10         if currentActionScore > bestActionScore then
11             bestAction ← a
12             bestActionScore ← currentActionScore
13 end
```

number of events of collected which are of interest to the application.

We have proposed a scheduler that can dynamically actuate cameras to satisfy an application requirement, e.g., collect as many high resolution facial images as possible. The main challenge is scalability as the state space of the environment is significantly larger than typical problems with similar formulations. Table 6.1 summarizes the different challenges and our approach. The heuristics and approximations that we followed made our scheduler tractable. The real time component is extremely light weight and fast. And, most importantly there was significant improvement over alternative approaches.

As part of future work we would also like to consider a hybrid push/pull approach that will benefit from the advantages of computation at the sensors while meeting the deadlines of a real-time system and supporting the dynamic nature of a surveillance

task. Furthermore. We would like to be able to learn the transition model when the zoomed out configuration can only observe part of the space assigned to a camera. The challenge is that the offline process would have to deal with partial observability, the cameras cannot "zoom out" to observe the entire environment. Merely collecting the training data for the transition model becomes not tractable. Next, we are planning to address the problems that arise when multiple application queries need to be optimized at the same time. Currently, we must generate a lookahead table in advance for a single reward function. For example, assume applications interested in both face recognition and motion detection are running at the same time, we want to find the action that has the highest expected utility for both applications based on the pre-computed tables for each query which computing the lookahead table for the combination of both. Finally, we are planning to experiment with approximation techniques for cameras with overlapping zoom regions where the utility of a single camera action depends on the actions selected by other cameras.

Chapter 7

Using Semantics for Re-Calibration

7.1 Introduction

With increased awareness among public, private and government sectors for need of greater security, surveillance technologies (especially video surveillance) have recently received a lot of attention. Video surveillance systems are being used in a variety of public spaces such as metro stations, airports, shipping docks, etc. The challenge of achieving high Accuracy of Detection (AoD) from large numbers of cameras increases as more cameras go up in more places.

Aside from the numbers of sensors currently being used, some applications require the deployment of sensors in distant and isolated locations. According to Liljegren el. al [9], the U. S. Department of Energy (DOE) Atmospheric Radiation Measurement (ARM) Program has deployed dual-channel microwave radio-meters in rural Oklahoma and Kansas, the north slope of Alaska, and on islands in the tropical Pacific Ocean. These radio-meters provide continuous measurements of integrated water vapor (IWV) and integrated liquid water (ILW) amounts. Due to the remote nature of

these locations, several weeks or months may elapse between maintenance visits by operations personnel. Even then, subtle problems that can adversely affect the instrument calibrations may go undetected. Achieving high accuracy of data generated from sensors of this nature is a real challenge.

In this chapter, we propose a general approach to ensure AoD for systems modeled as Finite State Machines (FSM). The challenge lies in designing a robust detection system for specific use when manual intervention is impractical due to the distributed nature of the sensors or their remote locations. We have designed a novel approach to ensure AoD by using semantics of the tracked system for the purpose of automatic re-calibration of the detection algorithms used by the sensors.

For the sensor to ensure AoD, it must accurately interpret the captured feature space to application specific information. Usually, this process involves setting several parameters. For example, OpenCV needs a CASCADE[1] as a parameter to perform face detection and a video camera needs a background model as a parameter. The detection algorithm in the ARM microwave radio-meter has to adjust to changes in the physical location of its mirrors that tend to slip as much as 1 degree on their stepper motor shafts due to continuous use.

Because they are deployed in uncontrolled environments the sensors must overcome problems in order to achieve better AoD. For example, in the case of computer vision, cameras might be replaced, lighting conditions may change and camera views might be altered. Physical changes could also occur due to continuous usage, e.g., the ARM microwave radio-meter example. In order to overcome these difficulties the sensor follows a pre-defined parameterized prediction model (we will clarify this in sec. 7.2). This model is used to perform the detection based on the sensed feature

[1]OpenCV [17] uses a statistical approach for object detection, an approach originally developed by Viollan and Johns [62] where features extracted from training sets of data are compressed into a statistical model stored as an XML document.

space but it is inherently imperfect. Specifically, changes due to the nature of the uncontrolled environment in which the sensor operates or due to imperfect matches between the learned model and the actual observed system will result in reduced AoD for that sensor. The parameters are set to reduce the uncertainty of the uncontrolled environment.

Although these parameters enable better state detection by the sensor when set correctly, when they don't match the current situation in the sensed environment they become a problem. Re-calibrating these parameters is necessary to ensure that the detection process meets AoD requirements. We propose a novel approach to re-set the detection algorithm to the new environment by searching the space of parameters to maximize detection consistency with a semantic model.

To the best of our knowledge, the proposed approach is first of its kind that allows parametrization in real time of a given imperfect model based on known system semantics to achieve better AoD. This approach was implemented and evaluated in a real setting and was found to be able to parameterize a model within a very large parameter space.

This chapter is structured as follows: In Sec. 7.2 we give a mathematical definition for the observed system and collected semantics. sec. 7.2 describes the architecture and outlines the role of semantics in the overall framework. Sec. 7.3 presents the algorithm for exploiting semantics for sensor re-calibration and improved AoD. Sec. 7.4 shows the instantiation of our semantic based detection model applied to an appliance: coffee pot level detection based on video data. Sec. 7.5 presents the results of applying the algorithm on our example application. Sec. **??** discusses related work. Future work and conclusions are presented in sec. 7.6.

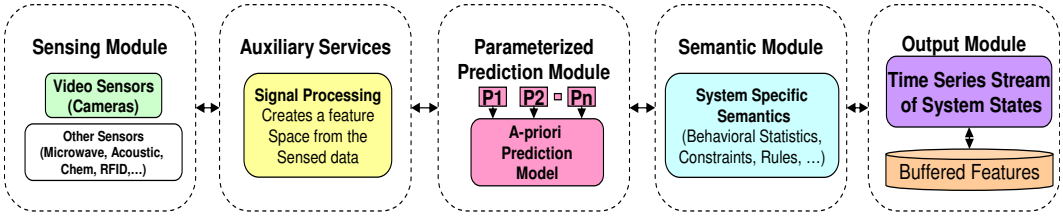


Figure 7.1: Main Modules Involved in the Detection Process.

7.2 System Architecture

Fig 7.1 depicts a high-level outline of the main modules that are involved in the detection process.

- **Sensing Module:** Processes data from incoming sensors. The sensing module is in-charge of the translation of the physical signal sensed by the sensor to the digital signal processed by the processor.
- **Auxiliary Services:** A service library containing modules that provide auxiliary services necessary to create the feature space from the digital signal made by the sensing module. The auxiliary services transform the data created by the sensor to form a higher level representation of the system that can be processed by the prediction module. For example, a service that translates the Jpeg image to the corresponding RGB color histogram is considered an auxiliary service.
- **Parameterized Prediction Module:** An a-priori model used for output generation by the sensor. For the sensor to satisfy AoD requirements the prediction module is consulted for prediction based on the features created by the auxiliary services. Since the prediction model performs differently under varying conditions, the model uses the notion of parametrization of the prediction. The added parameters, enable the prediction model to exploit some of the special characteristics of the specific conditions under which it is required to operate.

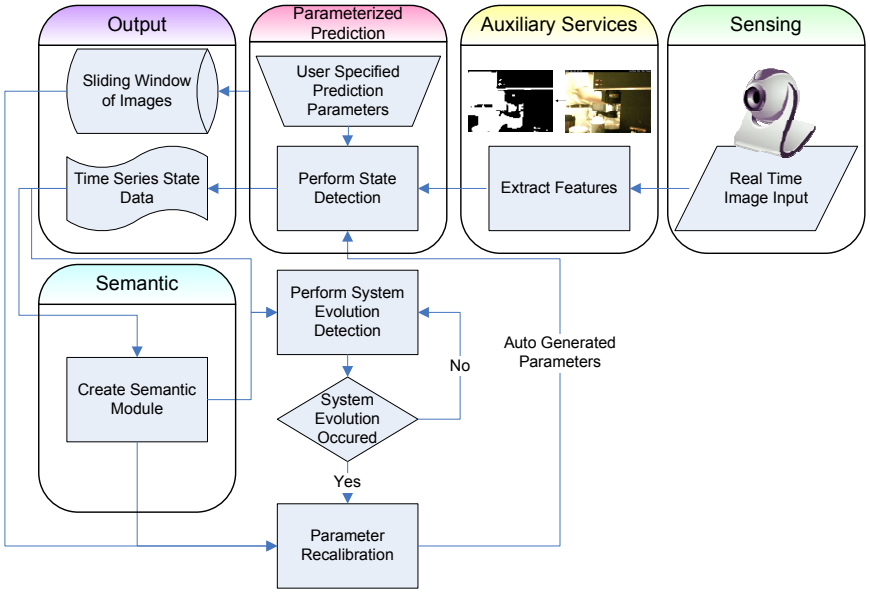


Figure 7.2: A flowchart of the execution of the algorithm. The modules involved are the same modules described in sec. 7.2 as well as the processes that execute the algorithm.

- **Semantic Module:**

  A model that describes the normal behavior of the system under surveillance, i.e., system semantics. This module is used to monitor the detection process and decide when it deviates from a normal operation expected from the system.

  This module is used for two main tasks: First, it detects the changes in the conditions that causes the system to stop detecting states correctly. Second, it finds new parameters for the new conditions. Once the new parameters are set the detection becomes once again consistent with the semantic model.

- **Output Module:** The output module generates a time series where each entry represents the time instant at which a state transition takes place. This module is also responsible for populating a buffer of sensor readings which is used for re-calibration when the conditions in the sensed environment change.

The flow of execution is presented in Fig. 7.2. The following are the five main steps of execution:

1. First, a monitored phase of detection is deployed. The user, specifies the prediction parameters and verifies the output of the detection algorithm. Output: a time series data of system states that was verified for accuracy.

2. Generate the semantic model. Output: a reference model used for system evolution detection and re-calibration.

3. Based on a parameterized prediction model the system executes and detects state as long as the series of states detected complies with the semantic model. Output: a time series data of system states verified against the semantic model.

4. When the states that were detected are no longer consistent with the semantic model, it is assumed that the system had evolved. This means, that the detection can no longer be performed based on the current parameterized prediction model. Output: a notice that the system had evolved and triggering the re-calibration phase.

5. The system tries to build a new reference model based on the semantic model (re-calibration phase). Output: success iff a new model was created that was verified for accuracy against the semantic model.

Step 5, is the key step in the algorithm where the previous model was detected as invalid and a new model is constructed. A successful outcome of this step enables the re-calibration of the detection model based on the semantics of the system. The next section, discusses this step in details.

7.3 Semantic Based System Evolution Detection and Re-Calibration

Definition 1. *Systemevolution refers to the change in the configuration of the system.*

Definition 2. *$time_{evolution}$ is the time instant at which the system changes completely from the old state to a new, evolved state.*

For example, changes in the physical location of the ARM microwave radio meter's mirrors will fall under our definition of system evolution. Changes in the view of the camera is also an instance of system evolution. We assume that system evolution occurs rarely but instantaneously. Studying the re-calibration of systems that do not follow these two assumptions is out of the scope of this thesis[2].

The following algorithm, scores a sequence of perviously detected time series stream of system states: ψ. The fraction of transition times that take more then two standard deviation units from the average time is returned as the score for ψ.

procedure $evaluation(\psi)$ **returns** score

$FaultCount \leftarrow 0$

for $s[i] \in \psi = \langle s[1], s[2], .. \rangle$ **do**

$trns \leftarrow t[i] - t[i-1]$

$class \leftarrow class[i]$

$\bar{x} \leftarrow GET\_AVG\_FROM\_MODEL(s[i], s[i-1], class)$

$\sigma \leftarrow GET\_STDDEV\_FROM\_MODEL(s[i], s[i-1], class)$

$range[A, B] \leftarrow [\bar{x} - 2\sigma, \bar{x} + 2\sigma]$

if $trns \notin range[A, B]$ **then**

[2]Systems that change patters of behavior for prolonged periods may cause the algorithm to start a re-calibration phase. That can be accounted using a larger stream ψ or by checking for outlier behavior before re-calibration as described by Scott(2000) [50].

$FaultCount \leftarrow FaultCount + 1$

end if

end for

return $\frac{FaultCount}{|\psi|}$

end procedure

Definition 3. *$Threshold_{Consistent}$ is a constant, defined by the user. This threshold defines the boundary between fault level that is consistent with the model, to the inconsistent fault level.*

7.3.1 System Evolution Detection

System evolution is declared iff $evaluation(\psi) > Threshold_{Consistent}$. The length of the stream $|\psi|$ will be referred to as $Window_{Buffer}$.

Definition 4. *$Window_{Buffer}$ is a first-in-first-out queue (sliding window) of features and series of system states (ψ).*

Definition 5. *$Buffer_{Transitions}$ is the number of actual state transitions of the system, captured by the features collected in the window buffer.*

Choosing the correct value for $Threshold_{Consistent}$ and the number of state transition in the window buffer ($Buffer_{Transitions}$) is discussed in sec. 7.3.3.

7.3.2 Model Re-Calibration

The re-calibration process is essentially a search for a new set of parameters that optimizes the consistency of the stream generated by the parameterized prediction model with the system semantic model:

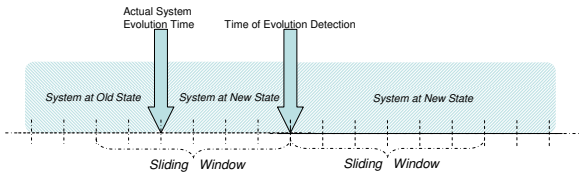


Figure 7.3: The time gap between the actual system evolution time and the evolution detection time. Before the re-calibration phase begins the algorithm ensures that the sliding window contains only observations that relate to the new system state.

procedure $Recalibrate\_prediction\_model(Prediction\_model, Window_{Buffer})$

returns $Parameters\ p1, p2, .., pn\ for\ the\ evolved\ state.$

$min\_score \leftarrow 1.0$

$\setminus\setminus\ \ P1, P2, .., Pn\ are\ the\ domains\ for\ p1, p2, .., pn$

for $p1, p2, .., pn \in P1, P2, .., Pn$ **do**

$score \leftarrow evaluation(\text{generateStream}^3(Prediction\_model(p1, p2, .., pn), Window_{Buffer}))$

if $score < min\_score$ **then**

$min\_score \leftarrow score$

$best\_parameters \leftarrow p1, p2, .., pn$

end if

end for

return $best\_parameters$

end procedure

The above algorithm assumes that $Window_{Buffer}$ only contains features that are of the evolved state, otherwise any single set of parameters will fit only a part of the $Window_{Buffer}$. However, although the evolution occurs in a time instance[4] this does not mean that the buffer includes only features of the evolved state. Fig. 7.3 illustrates the problem that arises when the exact evolution time cannot be determined. After a system evolution is declared at $time_{detected}$, we know evolution happened before

[3]generateStream generates the stream ψ using Prediction_model(p1,p2,..,pn)

[4]see def. 1 in sec. 7.3

$time_{detected}$. The problem is we don't know exactly when. Since we want to search for new parameters only for the evolved state, we propose waiting until $Window_{Buffer}$ contains only features that were recorded after $time_{detected}$ before engaging in the re-calibration phase. The features received after $time_{evolved}$ and before $time_{detected}$ will be assigned states based on the previous model, and thus the detection in that case will be inaccurate. Achieving correct state detection for this time period is outside the scope of this chapter. The next section shows how we applied the ideas presented in this section to our Coffee-Level detection example to achieve re-calibration when the view or zoom changed or the camera was replaced.

7.3.3 Choosing the Consistency Threshold and Buffer Size

Let p be the probability of a given transition to be *in*consistent with the statistical model (thus, falling outside the range of two standard deviation units from the average). Let us further define $InConsistent(\psi)$ as the number of transitions that are inconsistent with the system semantic model that belong to ψ.

The probability that the algorithm does not incorrectly go into the re-calibration phase when the correct parameters are being used is the probability that $evaluation(\psi)$ returns a score smaller than $Threshold_{Consistent}$. The following formula is the probability of observing up to $Threshold_{Consistent} * Buffer_{Transitions}$ inconsistencies[5] in independent trials, where the probability of inconsistency in any given trial is p:

$$\sum_{j=0}^{floor(Threshold_{Consistent} * Buffer_{Transitions})} \binom{Buffer_{Transitions}}{j} p^j (1-p)^{Buffer_{Transitions}-j}$$

Fig. 7.4 plots this probability for the two parameters: $Buffer_{Transitions}$ and $Threshold_{Consistent}$. p was set to 4.4% which is the percentage of data that will be outside this range for

[5] $\frac{InConsistent(\psi)}{Buffer_{Transitions}} < Threshold_{Consistent} \rightarrow InConsistent(\psi) < Threshold_{Consistent} * Buffer_{Transitions}$

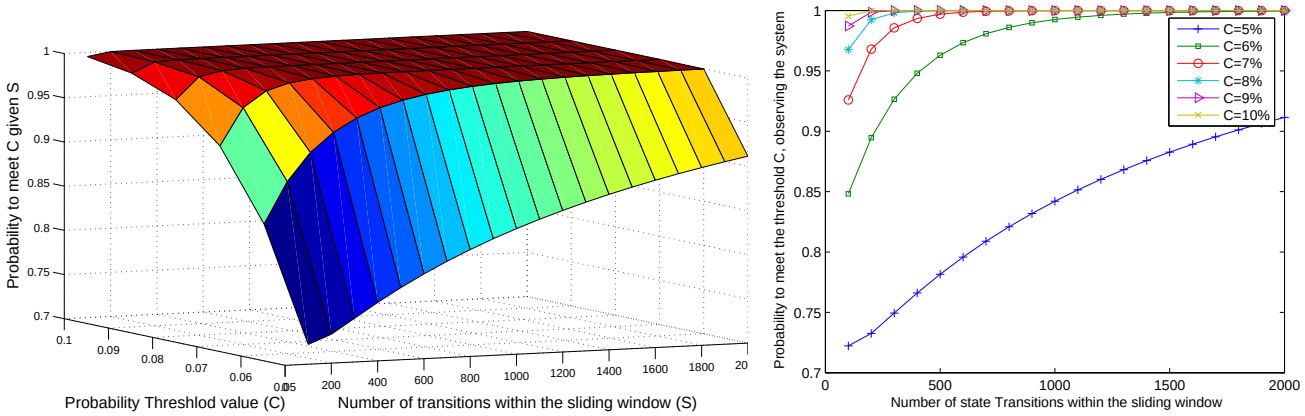


Figure 7.4: Probability of the observations collected in the $Window_{Buffer}$ to have less than C=$Threshold_{Consistent}$ percent of deviations ($p = 4.4\%$).

a normal distribution. For example, in the case of our coffee-machine we found that $p = 4.59\%$ for our test data. Choosing a high $Threshold_{Consistent}$ value will also increase the probability for false-positives when performing re-calibration (see sec. 7.3.2).

For a given $Buffer_{Transitions}$, say B_t, the length of time for which the system needs to be observed (i.e., to collect B_t distinct state transitions with a high probability) is decided based on the following two criteria:

1. A minimum of k instances of each state to retrain the prediction model.
2. $Buffer_{transitions}$ number of states needed to reliably re-calibrate the model, see fig. 7.4.

Therefore, the observation time interval can be computed as follows:

Definition 6. *$Buffer_{TimeLength}$ = MAX(Time period for k occurrences of the most infrequent state, Time period for $Buffer_{transitions}$ occurrences of the most frequent state).*

7.4 Applying the Semantic Based State Detection Model

We applied the model proposed to utilize coffee level detection based on simple set of color layout features that were extracted from a camera that had the coffee pot in view. In this section, we will show how our model was instantiated to a real application. This application should, however, be regarded as a proof of concept only. 7.5 presents the evaluation results of our appliance system after following the instantiation described in this section.

7.4.1 Sensing Module

For the experiments we collected a set of images from 10am on June 2007 until 10am June 13th 2007 from two cameras that were focused on the coffee pot ($Buffer_{TimeLength}$=3 days). The images were intended to simulate the buffer of images that would have been created by the proposed algorithm in real time.

If images would have been saved every time unit ($1sec$ in our case) the storage needed would have exceeded 14G[6]. We therefore followed a more compact approach and saved an image every time a change in the color layout was detected[7]. This resulted in a set of 1174 images = $Window_{Buffer}$ saved and less then 60Mb of storage used. Later the found that the number of transitions within this buffer was $Buffer_{Transitions}$=203.

[6] $14G = 3_{days} * 24_{hours} * 60_{minutes} * 60_{seconds} * 60k_{jpegSize}$.

[7] The change detection process itself is very important to the performance of the detection algorithm. All system states need to be captured while only changes that reflect "outliers" should be discarded (see sec. 4.2.1 for details). We saved images based on the change percentage CP in the color layout features extracted. The CP value was chosen s.t. all possible state transitions will result in a change in the color layout features that is larger then CP

7.4.2 Feature Extraction Using Auxiliary Services

In sec. 4.2.1 we defined the set of possible states of our appliance example as Coffee-Pot off, Empty, Half Full and Full. The auxiliary services used for detection of these states was Java Advanced Imaging (JAI [57]) library for color layout feature extraction. The image was transformed into a matrix $M(x, y)$ where each cell contains the average color of pixels included in the bounding rectangle ($upperleftX = x * C, upperleftY = y * C, width = C, height = C$) (C defines the number of grid cells extracted from the image). The grid of cells using black and white as the average colors is illustrated in Fig. 4.3.

7.4.3 Building the Parameterized Prediction Model

Once the features were extracted, we were able to perform prediction for the actual state represented by each image.

The prediction module we chose was based on color layout similarity. We clustered the images into four different groups based on their color layout. Specifically, we defined a distance function that computed the euclidian distance between the two matrix representations of two images. Based on this distance the images were clustered into $n = 4$ groups (one for each state). A centroid for each group was calculated and manually labeled according to the state it represented. Each image in the $Window_{Buffer}$ was assigned a state based on the label of the nearest centroid found to that image.

Two parameters were required for our prediction model. First, since the data was noisy, in that there were many changes that occur in the background scene, the prediction would have performed better had a background subtraction module being used. A bounding rectangle representation for selecting the system from the background

was the first parameter. The second was the actual labeling of the groups of images that were clustered together.

Thus, assuming that the parameters selected expressed most of the possible changes in the uncontrolled environment, when the system evolved, it is a matter of finding the correct new set of parameters to resume detection to meets the AoD requirements.

7.4.4 Building the Semantic Model

We performed state detection on the set on images collected based on a prediction module where the two parameters were manually seleted. The time series system state stream that was generated was loaded on to a PosGres Database using the following schema:

$Time$	$Class$	$CurrectState$	$PreviousState$	$TransitionTime$
τ	WeekendWorkTime	E	F	4
$\tau + \Delta$	WeekendWorkTime	F	E	4
$\tau + 2\Delta$	WeekendWorkTime	E	F	39
$\tau + 3\Delta$	WeekendWorkTime	H	E	177
.	.	.	.	.
.	.	.	.	.

7.4.5 Performing Re-Calibration

The algorithms below, perform the search for the bounding rectangle and correct labeling. Further details about the two parameters are presented in sections 7.4.5 and 7.4.5 respectively.

procedure *evaluate_rectangle*(*rectangle*) **returns** *score*

```
clusters ← clustering(rectangle)
min_score ← 1.0
min_labeled_clusters ← null
for permutation ∈ all k! labeling possibilities do
    labled_clusters ← labeling(clusters, permutation)
    score ← evaluation(time_series(labeled_clusters))
    if score < min_score then
        min_score ← score
        min_labeled_clusters ← labeled_clusters
    end if
end for
return min_score
end procedure
rectangle ← old_rectangle
while search continues do
    min_score ← evaluate_rectangle(rectangle)
    if min_score < Threshold_Consistent then
        return rectangle
    end if
    min_score ← 1.0
    min_rectangle ← null
    for rectangle ∈ all possible rectangle movement do
        score ← evaluate_rectangle(rectangle)
        if score < min_score then
            min_score ← score
            min_rectangle ← rectangle
        end if
    end for
    rectangle ← min_rectangle
```

end while

Parameter1: Bounding Rectangle

Features associated with current bounding rectangle will be extracted to form a set of feature vectors, one for each images from the learning window buffer. A grid divides the rectangle into m by n cells with the same size for each image. Average color is calculated for all pixels inside each cell for every image.

We used classic k-Means [36] algorithm with euclidian distance function for clustering the feature vectors. One thing we did that was different from a standard set-up is that we did not choose the initial k seeds of clusters randomly, rather we used a greedy algorithm which picks initial seeds in a way that none of them is too close to the other. The algorithm is proceeds as follows: choose the pair of points with maximum distance from all pairs of points as the first two seeds s_0 and s_1; $s_i(i > 1)$ is picked from all points such that the minimum distance from $s_j, (0 \leq j < i)$ is maximized. We found that in our setting this greedy algorithm performed better. The reason is that choosing seeds which are distant from each other will increase the probability of each seed representing a different state. Having many observations of one state (in our case, coffee pot empty) caused the centroids generated from a standard set-up to drift towards the random seeds, that were too of the frequently occurring state.

Labeling

Let there be k unlabeled clusters $c_0, c_1, ...c_{k-1}$ and k different labels $l_0, l_1, ...lk-1$ which represent k different states. The labeling procedure is to find the most likely mapping from c_i to l_i. Time series data generated for all different $k!$ labeling possibilities, each labeling instance is evaluated separately. The labeling that generates the

Figure 7.5: The initial and the resulting bounding rectangle for the two cameras.

lowest score will be considered the most likely labeling. The next section presents the results we achieved when we performed a greedy search in the space of parameters after the system had evolved. The results presented in the next section show, that re-calibration based on a trained model can actually improve AoD.

7.5 Evaluation and Results

We simulated three different situations that might occur: change in the location of the coffee-pot, change in the size of the coffee-pot (zoom in) and change in the camera being used. Fig. 7.5 shows how, in both our test cases, the starting rectangle was realigned to a position that enabled better detection. The second parameter, labels of the different clusters, was also selected correctly for all groups.

In our experiment we assumed that the new bounding box will be in proximity to the old one. Following this assumption we began our search for a new bounding box from the previous one and expended it iteratively in one of four possible directions: South, North, East and West, according to the greedy algorithm described in sec. 7.3. Our search thus counted only for the situations where the new bounding box contained the old.

We found that the percentage of transition times that fall outside the range $[avg - 2 * stddev, avg + 2 * stddev]$ for our test data was 4.59%. This value was calculated

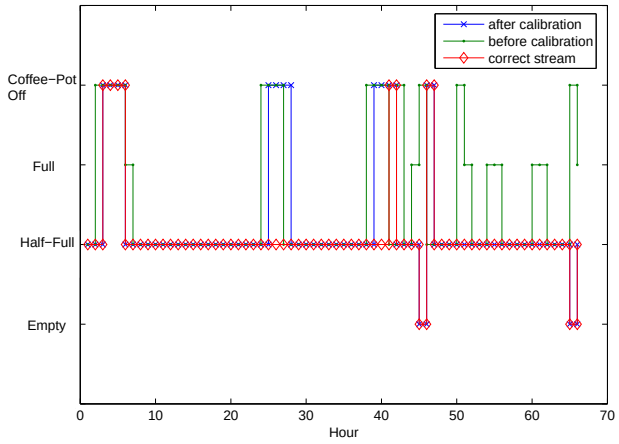


Figure 7.6: The state at with the system was at for the longest time, for each hour according to the different streams.

based on the stream generated in a controlled environment where both the bounding box and the labeling were manually specified. The $Threshold_{Consistent}$ value we used was 0.089 and, along with the 203 state transitions within our buffer, we understood that the probability of the algorithm that meet the threshold was:

$$\sum_{j=0}^{floor(0.089*203)} \binom{203}{j} 0.0459^{j}(1-0.0459)^{203-j} \cong 0.9972$$

Not only were the parameters correctly selected, but also the detection itself also improved. Fig. 7.6 compares the streams generated before and after the re-calibration took place.

7.6 Future Work

Due to the abstract nature of the ideas presented in this chapter, many applications can benefit from their use, despite the fact that the application of these ideas was executed in only one domain.

The work should be extended in the following five ways: we showed the applicability of our ideas for our Coffee appliance level detection example only. However, evaluating these ideas on other domains could be studied. Second, we followed a strait forward statistical approach towards the semantics generation. Other models (e.g., Hidden Markov Models (HMM) and rule based model, ..) should be studied in different domains. Third, we only used semantics to validate a given instantiation of parameters, basically performing a greedy search in the parameter space. Optimizing the search using a semantics based heuristic or semantic based search is required. Fourth, the case where the detection is performed using multiple sensors needs to be addressed. Fifth, the semantics collected could be used for prediction as well. We only used the semantics for re-calibration purposes.

It is our belief that building a general-case model for a general system using a general purpose semantic model following a general search strategy is, at the very least, a difficult task. The actual instantiation of these three modules will continue to vary from application to application.

We have presented a new approach for utilizing semantics to achieve better AoD and have proposed the parameterized model concept to improve performance of general case prediction algorithms.

We believe that this approach opens new windows of opportunity in the age of pervasive computing and will prove to be an important building block in a world of unconstrained environments and imperfect prediction models.

Chapter 8

Future Work and Conclusion

In this thesis we presented a set of algorithms to address adaptivity challenges in sentient spaces. The proposed algorithms utilize semantics of the phenomena being monitored by modeling the way it evolves over time and space in the monitored environment. First we address the scalability challenge by choosing the set of sensors that are most likely to observe an event of interest. Second we utilize the same semantics for the challenge of actuation, by computing the expected utility of different actions and choosing the action with the highest expected utility. Finally, we have proposed an effective and efficient technique for parameter selection used to re-calibrate sensors based on the expected "normal" way that the phenomena evolves over time.

As a motivating example, we have used MeerkatU - a real time notification system for a University office space. We have shown how low level events (motion) as well as high level events (identity recognition) can benefit from a principled approach to address middleware level challenges. We believe that building sentient systems, such as MeerkatU, can benefit from techniques that abstract sensor level challenges from applications writers.

The thesis started with the hypothesis that semantics can help guide the middleware to address adaptivity challenges in sentient spaces. We have developed a principled approach for using semantics for data collection, actuation and re-calibration. While our approach was evaluated in the context of camera based systems, the models that we have developed and the way that we use these models is much more general. For example, a very different type of sensor enabled application wherein loop sensors are used to measure traffic flows on various freeways/road network, and the data captured is used to build applications such as route planning, traffic jam determination, etc. We can use the techniques described in this thesis as follows: The phenomena that we monitor would be the traffic flow at each sensor discretized to a finite set of flow states, e.g., average speed rounded to the nearest value in 10 miles per hour intervals. The phenomena semantics capture the nature in which traffic flows are correlated between different locations. The techniques that we have developed in this thesis apply to this setting and can be used in order to schedule the data collection from the different sensors in order to detect an event of interest - slow traffic flow (i.e., speed of less than 10 miles per hour). The data collection process would increase the number of slow traffic flow events collected. Furthermore, we can reason about the expected affects of actions such as controlling the delay in traffic light signals located at entry points to highways and actuate the traffic light to reduce the chance of a traffic jam.

Reasoning about the future state of the world based on phenomena semantics is a general concept, which applies for different types of sensors and numerous sentient space application domains. Our results suggest that sensor level challenges can be abstracted and effectively addressed by a middleware layer. We believe that our approach will serve as a fundamental building block for building the next generation sentient systems.

This work can be extended in multiple directions. We discuss the possible extensions in the relevant chapters discussing data collection, actuation and re-calibration. Other directions of future work include addressing the heterogeneity, uncertainty and privacy challenges at the middleware layer. Applying the techniques developed in this thesis to other sensor networks, in which multiple types of sensors are available is a promising direction and can lead to techniques for addressing the heterogeneity challenge. In this thesis we have focused on problems that require collecting observations of a phenomena (e.g., collect motion, event start, high resolution facial image). Another approach would be to collect observations such that the system's representation of the world is as accurate as possible. In this case, reward is not necessarily associated directly with an observation but with the observation's effects on the system's internal representation of events that are taking place in the real world. This can lead to techniques for addressing the uncertainty challenge. Finally, future work should consider experimenting with more complicated cost function. For example, if both a camera and a motion sensor are available to observe motion in a certain region of the monitored environment, collecting an observation from a motion sensor has a lower cost, for example, in terms of violating the privacy of individuals. This can lead to techniques to incorporate privacy preserving considerations to the data collection process and in turn addressing the privacy challenge.

Bibliography

[1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[2] P. Anandathirtha, K. Ramakrishnan, S. Raja, and M. Kankanhalli. Experiential Sampling for Object Detection in Video. *Multimedia Content Analysis: Theory and Applications*, page 175, 2009.

[3] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. *Book chapter*, 2004.

[4] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB JournalThe International Journal on Very Large Data Bases*, 15(2):121–142, 2006.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas. Operator scheduling in data stream systems. *The VLDB JournalThe International Journal on Very Large Data Bases*, 13(4):333–353, 2004.

[6] J. Baxter, P. Bartlett, and L. Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15(1):351–381, 2001.

[7] M. Botts and A. Robin. Sensor model language (sensorml). *Open Geospatial Consortium Inc., OGC*, pages 07–000, 2007.

[8] R. Brown and P. Hwang. Introduction to random signals and applied Kalman filtering. 1997.

[9] L. J. C. Automatic self-calibration of the arm microwave radiometers, 2000.

[10] A. Cassandra, L. Kaelbling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1023–1023. Citeseer, 1995.

[11] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668. ACM, 2003.

[12] C. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 161–172, 1994.

[13] E. Daeipour and Y. Bar-Shalom. An interacting multiple model approach for target tracking withglint noise. *Aerospace and Electronic Systems, IEEE Transactions on*, 31(2):706–715, 1995.

[14] M. Di Francesco, K. Shah, M. Kumar, and G. Anastasi. An adaptive strategy for energy-efficient data collection in sparse wireless sensor networks. *Wireless Sensor Networks*, pages 322–337, 2010.

[15] D. Forsyth, O. Arikan, L. Ikemoto, J. O'Brien, and D. Ramanan. Computational studies of human motion: part 1, tracking and motion synthesis. *Foundations and Trends® in Computer Graphics and Vision*, 1(2-3):77–254, 2005.

[16] Z. Fu and N. Venkatasubramanian. Adaptive parameter collection in dynamic distributed environments. *icdcs*, 00:0469, 2001.

[17] V. P. G Bradski, A Kaehler. Learning-based computer vision with intels open source computer vision library, 2005.

[18] P. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a worldwide sensor web. *Pervasive Computing, IEEE*, 2(4):22–33, 2003.

[19] M. Grossglauser and D. Tse. A framework for robust measurement-based admission control. *SIGCOMM Comput. Commun. Rev.*, 27(4):237–248, 1997.

[20] V. Gupta, T. Chung, B. Hassibi, and R. Murray. On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage. *Automatica*, 42(2):251–260, 2006.

[21] Q. Han, S. Mehrotra, and N. Venkatasubramanian. Energy efficient data collection in distributed sensor environments. *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, 39:590–597, 2004.

[22] A. Ihler, J. Hutchins, and P. Smyth. Adaptive event detection with time-varying poisson processes. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 207–216, New York, NY, USA, 2006. ACM Press.

[23] V. Isler and R. Bajcsy. The sensor selection problem for bounded uncertainty sensing models. *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 151–158, 2005.

[24] D. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM Data Mining (SDM) Conf*, 2005.

[25] B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. *IEEE Transactions on Parallel and Distributed Systems*, 8(12):1268–1274, 1997.

[26] J. King. Quist: A system for semantic query optimization in relational databases. In *Proceedings of the seventh international conference on Very Large Data Bases-Volume 7*, pages 510–517. VLDB Endowment, 1981.

[27] V. Krishnamurthy. Algorithms for optimal scheduling and management of hidden Markovmodel sensors. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 50(6):1382–1397, 2002.

[28] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. 2003.

[29] B. Liu, A. Gupta, and R. Jain. Medsman: a streaming data management system over live multimedia. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 171–180. ACM, 2005.

[30] W. Lovejoy. Computationally feasible bounds for partially observed markov decision processes. *Operations research*, pages 162–175, 1991.

[31] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2003.

[32] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, 2005.

[33] D. Makris, T. Ellis, and J. Black. Bridging the gaps between cameras. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 2, 2004.

[34] D. Massaguer, S. Mehrotra, R. Vaisenberg, and N. Venkatasubramanian. Satware: A semantic approach for building sentient spaces. *Distributed Video Sensor Networks*, pages 389–402, 2011.

[35] S. McIlraith, T. Son, and H. Zeng. Semantic web services. *Intelligent Systems, IEEE*, 16(2):46–53, 2001.

[36] J. McQueen. Some methodes for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, CA, USA, 1967. University of California Press.

[37] K. Murphy. A survey of pomdp solution techniques. *environment*, 2:X3, 2000.

[38] K. P. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in dbns. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 378–385, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[39] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 133. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[40] A. Ng and M. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 406–415. Citeseer, 2000.

[41] J. Nieh and M. Lam. The design, implementation and evaluation of SMART: A scheduler for multimedia applications. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 184–197. ACM New York, NY, USA, 1997.

[42] N. R. C. U. C. on Networked Systems of Embedded Computers. *Embedded, everywhere: A research agenda for networked systems of embedded computers*. National Academies Press, 2001.

[43] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[44] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International joint conference on artificial intelligence*, volume 18, pages 1025–1032. Citeseer, 2003.

[45] M. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.

[46] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[47] R. Report. Wireless sensor networks and home control. *ABI Research*, pages 1–60, September 2010.

[48] A. Sang and S. qi Li. A predictability analisys of network traffic. *Computer Networks*, 39:329–345, 2002.

[49] D. Schulz, D. Fox, and J. Hightower. People tracking with anonymous and id-sensors using rao-blackwellised particle filters. *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[50] S. Scott. Detecting network intrusion using the markov modulated nonhomogeneous poisson process.

[51] R. Shah and J. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE*, volume 1, pages 350–355. Ieee, 2002.

[52] G. Shani, R. Brafman, and S. Shimony. Prioritizing point-based pomdp solvers. *Machine Learning: ECML 2006*, pages 389–400, 2006.

[53] C. Shu, A. Hampapur, M. Lu, L. Brown, J. Connell, A. Senior, and Y. Tian. Ibm smart surveillance system (s3): a open and extensible framework for event based surveillance. In *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pages 318–323. IEEE, 2005.

[54] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(12):1349–1380, 2000.

[55] B. Song and A. Roy-Chowdhury. Stochastic Adaptive Tracking In A Camera Network. *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.

[56] C. Stauffer and W. E. L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, 2000.

[57] Sun. Java advanced imaging. *http://java.sun.com/javase/technologies/desktop/media/jai/*.

[58] L. Sweeney and R. Gross. Mining images in publicly-available cameras for homeland security, 2005.

[59] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[60] R. Vaisenberg, S. Mehrotra, and D. Ramanan. Exploiting semantics for scheduling data collection from sensors on real-time to maximize event detection. In *Proceedings of SPIE. SPIE*, 2009.

[61] P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo. Letting loose a spider on a network of pomdps: Generating quality guaranteed policies. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8. ACM, 2007.

[62] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *cvpr*, 01:511, 2001.

[63] C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[64] Wikibook. *The Wikibook of: Control Systems and Control Engineering*. Wikibooks, 2012.

[65] J. Williams, J. Fisher, and A. Willsky. Approximate dynamic programming for communication-constrained sensor network management. *Signal Processing, IEEE Transactions on*, 55(8):4300–4311, 2007.

[66] J. Williams, J. Fisher III, and A. Willsky. An approximate dynamic programming approach to a communication constrained sensor management problem. *Proc. Eighth International Conference of Information Fusion*, 2005.

[67] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 38(4):13, 2006.

[68] X. Yu, K. Niyogi, S. Mehrotra, and N. Venkatasubramanian. Adaptive target tracking in sensor networks.

[69] W. Zhao, R. Chellappa, P. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys*, 35(4):399–458, 2003.