

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Automated crowd-counting system upon a distributed camera network

Permalink

<https://escholarship.org/uc/item/2kd829gf>

Author

Morrow, Mulloy

Publication Date

2012

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Automated Crowd-Counting System upon a Distributed Camera Network

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

in

Electrical Engineering
(Intelligent Systems, Robotics, and Control)

by

Mulloy Morrow

Committee in charge:

Professor Nuno Vasconcelos, Chair
Professor Kenneth Kreutz-Delgado
Professor Truong Nguyen

2012

Copyright
Mulloy Morrow, 2012
All rights reserved.

The Thesis of Mulloy Morrow is approved and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2012

DEDICATION

To my family.

Mom, Pop, Kelland, Mollie, and Bi

EPIGRAPH

Just Keep Swimming.

—Dory

TABLE OF CONTENTS

| | |
|--|------|
| Signature Page | iii |
| Dedication | iv |
| Epigraph | v |
| Table of Contents | vi |
| List of Figures | ix |
| List of Tables | x |
| Acknowledgements | xi |
| Vita and Publications | xii |
| Abstract of the Thesis | xiii |
| 1 Introduction | 1 |
| 1.1 Prior Work | 4 |
| 1.1.1 System I | 4 |
| 1.1.2 System II | 8 |
| 1.2 Current Work | 9 |
| 1.2.1 System Overview | 10 |
| 2 Acquisition Methods | 14 |
| 2.1 Surveillance Network Hardware | 17 |
| 2.1.1 Cameras | 17 |
| 2.1.2 System Manager (SM) | 20 |
| 2.1.3 Network Video Recorders (NVRs) | 20 |
| 2.2 Surveillance Network Software | 20 |
| 2.2.1 Pelco SDK Libraries | 21 |
| 2.2.2 Relevant Functions | 23 |
| 2.3 Acquisition Module | 27 |
| 2.3.1 Exporter Application | 27 |
| 2.3.2 Timing Schema | 28 |
| 2.3.3 Video Stream Parameterization | 29 |
| 3 Analysis Methods | 34 |
| 3.1 Crowd-Counting Overview | 34 |
| 3.2 Computational Tools | 35 |
| 3.2.1 Dynamic Texture Model | 36 |

| | | |
|-------|---|----|
| 3.2.2 | Motion Segmentation | 39 |
| 3.2.3 | Features | 41 |
| 3.2.4 | Regression | 41 |
| 3.3 | Automation Implementation | 42 |
| 3.3.1 | Scene Setup | 42 |
| 3.3.2 | Count Functions | 43 |
| 3.3.3 | Experiment Setup | 43 |
| 3.4 | Known Issues and Notes | 45 |
| 4 | Visualization Methods and Results | 47 |
| 4.1 | Core Python Scripts | 47 |
| 4.1.1 | ShowTables | 47 |
| 4.1.2 | CreateTable | 47 |
| 4.1.3 | AppendTable | 48 |
| 4.1.4 | Table2XML | 50 |
| 4.2 | Visual Demos | 50 |
| 4.2.1 | Count Vectors | 50 |
| 4.2.2 | Realtime Map and Trends | 51 |
| 4.2.3 | Zoomable Line Charts | 51 |
| 4.2.4 | Known Issues | 55 |
| 5 | Future Work | 56 |
| 5.1 | Code Optimization | 56 |
| 5.2 | Other Counting Methods | 56 |
| 5.3 | Suppressing Training Costs | 57 |
| 5.4 | Suppressing Count Variance | 57 |
| 5.5 | Control Theory | 58 |
| 5.6 | Viewpoint Invariance | 58 |
| | References | 60 |
| A | Appendix - People Counting System | 62 |
| A.1 | Compiled Matlab Programs | 62 |
| A.2 | Options File for Training People Counting Model | 62 |

LIST OF FIGURES

| | | |
|-------------|--|----|
| Figure 1.1: | Past Work - Event-Recognition Scenes | 5 |
| Figure 1.2: | Past Work - Example Scene from PETS 2009 Dataset S1 | 8 |
| Figure 1.3: | Current Work - Simplified System Flowchart | 11 |
| Figure 1.4: | Current Work - System Diagram | 13 |
| Figure 2.1: | Acquisition Module Diagram | 16 |
| Figure 2.2: | Camera Locations - Campus Map | 18 |
| Figure 2.3: | Camera Locations - Price Center Map | 19 |
| Figure 2.4: | Export Schedule | 29 |
| Figure 3.1: | Crowd-Count System Flowchart | 35 |
| Figure 3.2: | Dynamic Texture Model (DTM) Diagram | 38 |
| Figure 3.3: | Motion Segmentation Flowchart | 39 |
| Figure 3.4: | Analysis Module Diagram | 42 |
| Figure 3.5: | Analysis Schedule | 43 |
| Figure 4.1: | Visual Demo 1 - Count Vectors | 52 |
| Figure 4.2: | Visual Demo 2 - Realtime Map and Timeline (All Scenes) | 53 |
| Figure 4.3: | Visual Demo 3 - Zoomable Timelines. | 54 |

LIST OF TABLES

| | | |
|------------|--|----|
| Table 2.1: | Pelco Camera Features | 17 |
| Table 2.2: | Pelco SDK Libraries | 22 |
| Table 2.3: | Configuration File Datatypes | 29 |
| Table 2.4: | Configuration File Format | 30 |
| Table 2.5: | Acquisition Module Export Parameters | 31 |
| Table 2.6: | Acquisition Module Scene Settings | 32 |
| Table 3.1: | Segmentation Features Table | 40 |
| Table 3.2: | Core Analysis Automation Functions | 44 |
| Table 3.3: | Required Files for Count Training | 44 |

ACKNOWLEDGEMENTS

Thank you to Jack, Patrick, Jan, Noah, Jen, Emily, Kim, Reza. A special thanks to Jen for all her support, patience and love. Also, thank you to all the cafes in San Diego, the tangueros(as), Biagi, Pugliese, D'Arienzo, Fresedo, Demare, and others.

VITA

- 2008 B.S. in Engineering Physics, University of California, San Diego
- 2012 M.S. in Electrical Engineering (Intelligent Systems, Robotics,
and Control), University of California, San Diego

PUBLICATIONS

A.B. Chan, M. Morrow, N. Vasconcelos, “Analysis of Crowded Scenes using Holistic Properties”, *In 11th IEEE Intl. Workshop on Performance Evaluation of Tracking and Surveillance (PETS)*, 2009.

ABSTRACT OF THE THESIS

Automated Crowd-Counting System upon a Distributed Camera Network

by

Mulloy Morrow

Master of Science in Electrical Engineering
(Intelligent Systems, Robotics, and Control)

University of California, San Diego, 2012

Professor Nuno Vasconcelos, Chair

Automated and Distributed-Camera Crowd Analysis is an impressive and important research challenge that has recently gained prominence in our society. Its applications include increased security and efficiency of public environments, research in herd and flocking behavior, population monitoring, urban architecture, and also marketing. However, there exists a striking difference between the environments where we deploy and where we develop these analytics, resulting in non-robust analytics.

For the purpose of elucidating prevalent challenges faced by SVCL video crowd analytics, we develop a scalable, distributed and automated research platform composed of three sub-solutions: (1) Acquire data from a dynamic and distributed-camera environment; (2) Automatically compute crowd-count estimates based on privacy-preserving

holistic motion segmentation; and (3) Visualize results geo-spatially and temporally on interactive maps.

Rather than placing comparative-emphasis on computation methods, however, we consider the influence and limitations our research community's video databases pose. Most pronounced is their static and finite nature, which may be a myopic characteristic constricting our research efforts. Therefore, we contrast results and note the added dynamic and long-term utility provided by our automated platform.

Our current video database yields geo-spatial real-time statistics of pedestrian traffic as well as long-term temporal trends over a distributed and connected geographic area. By designing a rich, scalable, and common experimental environment, we can more rigorously evaluate machine vision techniques and crowd dynamics. Attention may be shifted away from evaluation based solely on accuracy and more readily towards the inclusion of technique break-down.

1 Introduction

Our research project was conducted at the Statistical Visual Computing Laboratory (SVCL) at UC San Diego. SVCL performs research in both fundamental and applied problems in computer vision and machine learning. SVCL focuses on the development of intelligent systems, which combine image-understanding capabilities with any available additional information to enable sophisticated recognition and modeling, amongst other tasks. Strong emphasis is given to formulations that can deal with noise and uncertainty and solutions that are provably optimal under suitable optimality criteria.[1]

One technique that followed this SVCL criteria, developed by SVCL alumnus Prof. Antoni Chan, was the re-representation of video as a linear dynamic system (LDS); a representation that has lent itself to robust and privacy-preserving event-recognition and crowd-counting based on holistic motion.[2, 3, 4] (details in Chapter 3, Analysis Methods.) This work produced a crowd analytics module and laid the foundation for our automated crowd monitoring and surveillance system.

Automated crowd monitoring and surveillance is a very interesting challenge for vision analytics of today. Albeit challenging, the computational foundations for many useful solutions exist. Some of our environments that would benefit immediately from automated crowd analytics include areas vulnerable to security threats from crowds and/or terrorists such as schools, airports, walkways, sports facilities, hospitals, and amusement parks. Furthermore, crowd analytics has very interesting and promising applications in areas of research in herd and flocking behavior of animals, population monitoring, entertainment, urban architecture, and also marketing. Both immediately in our public lives and in our research communities, crowd analytics provides an interesting avenue for automated monitoring, management, and safety.

Naturally, robust and practical solutions necessitate a rich video corpus in which to develop. However, a corpus requires information to be highly structured. In our visual environment, structure with which to disentangle symbolic information is not as apparent. We have not a highly structured written language analogue in the visual world as we do in the audible. Despite this disadvantage, many highly specific solutions have risen from the cataloging of common significant visual patterns. However, crowds exist outside of these highly controlled and visually sterile environments. They exist in a complex visual world, such as an airport. How do we begin to disentangle the plethora of symbolic visual information? Furthermore, how do we do so scientifically so that objective comparisons may be drawn between evaluations of unique solutions.

As we will discuss in subsection 1.1.2, in our science community there exist conferences and workshops that solicit this discussion internationally and provide static and closed datasets (corpora) as a common development and testing environment.

Here I introduce our usage of two terms: static and closed. Keeping in mind that a visual corpus is composed of a set of structured visual data, we borrow the signal processing term *static* to imply a sense that the information is unchanging in a temporally local sense. Similarly, we borrow the physics term *closed* to imply the corpus contents as comprising a system remain constant and finite.

To use a simple metaphor, inspecting the same group of ants in a Petri dish defines a closed system. Although there may be a temporal component, our collection of subjects is unchanging. Observing the ants outside in the dirt where ants may come and go from our scope of observation describes an open environment. If we now turn our attention to our observation device, perhaps a magnifying glass, static implies a corpus developed using magnifying glass(es) with no change to their dimensions or position. However, if our magnifying glass(es) were somehow able to change their dimensions, this would alter the scope of our observation and give rise to a dynamic corpus. At all times, we presume the ants remain ants and not transform into gorillas. An equally fascinating animal with curious herding habits. However, our subjects and their behavior, which we represent as visual signals, give rise to statistically stationary patterns. Furthermore, the structure with which we catalog information in our corpus is unchanging as well.

To further characterize the majority of working databases in the community at present, the approach thus far has been mostly bottom up, to focus on specialized fundamental problems and work our way outward to comprehensively survey the entire visual universe, developing pattern disentanglement tools along the way. As we discuss in subsection 1.1.1, this rather brute force development of a large and comprehensive visual database is costly and only effective to the extent that our asexual algorithms may be optimized to our finite corpora. Continuing on this endeavor, in subsection 1.1.2 we follow another strategy based on a standardized multi-view database. The advantage of this strategy is the ability to compare the performance of solutions in a rather fair and normalized method that uncovers common as well as solution specific pitfalls.[5]

However, these pitfalls are little help without enough information to provide for the statistical insight that would reflect their source. Furthermore, if our goal is also to elucidate new challenges, a static database may not be expansive enough to include sparse and/or subtle patterns. In other words, although this bottom up approach has yielded many excellent practical solutions, we will discuss how this piecewise bottom-up approach suffers from a systematic error leading to myopia and how our project seeks to overcome these limitations. First, by paving the path to greater statistical insight from the use of visual analytics on a real-time distributed surveillance system. And secondly, by overcoming fundamental pitfalls to build robust crowd analytics.

In section 1.2, we will begin to discuss our general-purpose real-time analytics platform and our approach to exploring strengths and weaknesses of our analytic module; in particular uncovering novel and statistically significant obstacles to robustness.

Our goals with this insight are to reflect pitfalls in algorithm robustness as well as elucidate new challenges for the visual analytics fields. This necessity for an improved data source and testing platform comes at a time when our definitions of robustness are too narrow in scope; particularly in contrast to biological analogues, which are a growing inspiration.

This project 1) implements a platform for automated monitoring and surveillance of crowded scenes, 2) provides a common experimental environment in which to test crowd analytics continuously and in real-time, all while 3) overcoming common limiting constraints such as static databases applicable only to specialized subsets of

problems. This project paves a pathway for new and extended crowd analytic evaluation, including: visualizing distributed crowd dynamics across an expansive spatial area as well as temporally yielding trends over extended durations. Furthermore, this project elucidates new avenues of crowd research such as 1) crowd interpolation of unmonitored network pathways, 2) object and person tracking across fields of view, and 3) crowd analysis of areas with simultaneous multiple perspectives.

1.1 Prior Work

The discussion covering prior relevant work will be divided into two parts describing separate projects. In subsection 1.1.1, the first project consisted of crowd and traffic database research and development followed by event recognition experiments. In subsection 1.1.2, the second project turned into a IEEE paper submission and consisted of event recognition and crowd counting results. This second project was a small part of a larger project completed by SVCL Alumnus, Dr. Antoni Chan.

1.1.1 System I

We begin our endeavor with brute force, by attempting to create a large and diverse database consisting of both pedestrian and vehicular traffic via recording data with a single camera and tripod. Subsequently, this data is used to perform event recognition experiments, which will give us an idea of technique performance based on accuracy and robustness.

In Figure 1.1 on page 5, there is a list of images representing classes of events and accuracy results in predicting each based on techniques [2]. In most cases, 3 classes were discriminated based on traffic level/congestion (i.e. high, medium, low). In Figure 1.1 row (d), the exception is an experiment containing 7 classes differentiated along vehicular traffic-light system state (i.e. (a) north-through and south-through, (b) east-through and west-through, (c) north-through and north-turning-left, and (d) south-through and south-turning-left are four examples). In a significant number experiments, accuracy drops to the 50% level, indicating robustness pitfalls. Due to numerous noise sources, causes for pitfalls is inconclusive.


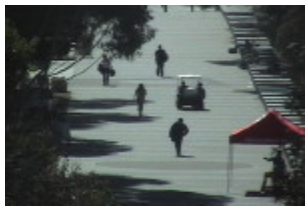


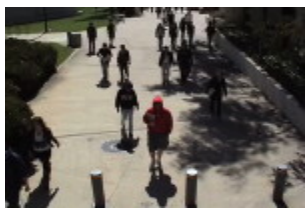
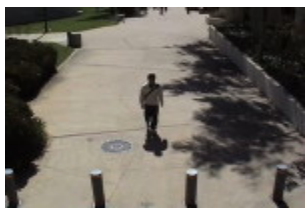



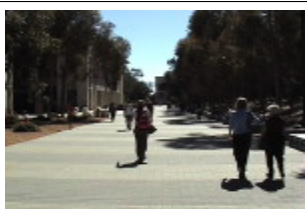

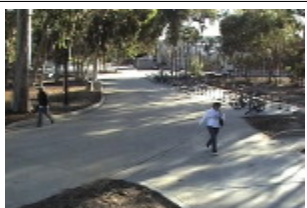
| | | | |
|-----|---|--|---|
| (a) |  |  | 82% Martin NN 82% State KL NN 83% State KL SVM 62% Image KL NN 87% Image KL SVM |
| (b) |  |  | 70% Martin NN 71% State KL NN 73% State KL SVM 73% Image KL NN 72% Image KL SVM |
| (c) |  |  | 56% Martin NN 54% State KL NN 57% State KL SVM 54% Image KL NN 55% Image KL SVM |
| (d) |  |  | 83% Martin NN 84% State KL NN 82% State KL SVM 68% Image KL NN 73% Image KL SVM |
| (e) |  |  | 64% Martin NN 64% State KL NN 58% State KL SVM 62% Image KL NN 70% Image KL SVM |
| (f) |  |  | 84% Martin NN 83% State KL NN 89% State KL SVM 67% Image KL NN 87% Image KL SVM |

Figure 1.1: Past Work - Event Recognition. *Left two columns:* examples of class differences. *Right column:* accuracy results.

Reviewing the results, little was gained at a great cost of database authoring. What *was* gained was insight of the arbitrariness of database design and the difficulty in capturing pre-determined scenes without a full scale movie production team to control all aspects of the environment. In other words, in scripting experimentally-salient scenes, there is a framing-out of many patterns, whether it be to minimize costly resources or to reduce extraneous data considered noise.

The University of California San Diego campus contains a variety of pathways and traffic scenarios, which provide an abundance of visual crowd patterns that the computer vision and machine learning science communities have addressed. As such, the campus provides for a good source of normal data for the development and comparing of solutions. For anomalies, weekly and monthly events such as fairs, campus tours, and the occasional protest provide interesting changes in crowd dynamics. At a finer level, crowd dynamics are affected by the numerous golf carts in service and which are allowed to use the same network of paths as pedestrians.

In addition to the thousands of visitors and staff on campus everyday, nearly $30 \cdot 10^3$ students are in attendance each quarter.[6] Although only a fraction of these students are present on campus at any instance of time, the number of persons typically on campus, the size of walkways and size of facilities supporting their activities is sufficient to provide multiple scenarios of consistent crowds with stable behavior. This provides for the collection of data comprising crowd classes of varying congestion levels, counts and holistic motion. Nonetheless, although we may observe these classes occasionally they may not all be present during our small recording time.

With that said, the strengths of this crowd analysis approach is in discriminating holistic crowd behaviors. In other words, our use of a generative model based on stationary properties of a stochastic process facilitates discrimination across these stationarities. Furthermore, these stationarities are dependent on camera properties such as field of view, orientation, focal distance, and perspective. Therefore, scene consistency is necessary. *Consistency* ideally presumes scene data was collected (a) using the same camera, (b) the camera was stationary, (c) lighting conditions were consistent. (a) using the same camera ensures focal distance and image contrast is constant. (b) stationary camera ensures holistic motion, e.g. from-left-to-right, doesn't transform into a contrast-

ing motion, i.e. from-top-to-bottom. (c) lighting conditions greatly affect the existence of shadows, lens flares, motion seen. These features greatly affect the segmentation area of DTMs and consequently may affect analytics dependent on these area features. Although these types of features do not greatly affect analytics such as event recognition, other analytic systems such as crowd prediction (our analytic of focus discussed and employed throughout the rest of this thesis) *is* highly dependent on such features.

Considering these constraints, developing a *complete* database proved difficult. Pre-production work included selecting scenes that would support data collection from orthogonal view points and the pre-determined levels or classes of traffic. For foot traffic, this meant shooting a single scene and collecting all desired classes in the span of hours to ensure lighting consistency and class consistency from all angles. Many scenes will support consistent crowd dynamics. However, the entire range of desired classes existing in that span of hours was not always a sure bet. This resulted in a database with many incomplete scenes, i.e. non-existing classes on which our experiments depended.[7]

To fill in these missing classes, in a few cases we revisited the scene at a later date in attempt to capture the missing contrasting crowd patterns. Great attention was given to placing the camera in the same position and orientation. However, despite this great effort the original camera position could never be exactly regained. Any changes of this type introduce an unpredictable bias to our signal. Other sources of bias resulting from this revisiting are changes in lighting conditions, differing collective energetics of the individuals comprising the crowds (different times of day), and sometimes a slight change in pathway layout.

Despite these inconsistencies, event-recognition based on holistic motion proved rather robust to some biases. For instance, strong direct light yielding strong shadows and sparse light at night yielding low contrast did not greatly affect classification as long as these extreme lighting conditions were present in training data. However, as we will see in the subsequent sections, this robustness to lighting conditions is a strength unique to event-recognition. This can be answered by or can be accounted for by the fundamental assumption that a single DT suffices to describe each frame. However, when this assumption is discarded and replaced by describing each frame with a mixture

of textures, or a Dynamic Texture Mixture (DTM), these sources of bias become much more apparent, as we will see in the next section.

1.1.2 System II

IEEE's PETS 2009 workshop, formally known as the Eleventh IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, was held in Miami 2009 in conjunction with the CVPR 2009 conference. The workshop aimed at bringing together and comparing performance of systems designed for the purpose of visual tracking and surveillance. Unique to this workshop was the usage of a static multi-viewpoint dataset used for all experimental result submissions, which normalized evaluation and performance comparisons to an arguable degree.

The theme of the workshop was on multi-sensor crowd analysis and event recognition in public areas. There were three levels of analysis sought after: low-level crowd counting; mid-level tracking of individuals within a crowd; and high-level event recognition and stream detection.[8] The multi-sensor aspect of the theme referred to the crudely synchronized multiple camera viewpoint setup. The crowd behavior captured in these datasets was produced using multiple actors.



Figure 1.2: Example Scene from PETS 2009 Dataset S1

In our workshop submission, we explained how we conducted Crowd Counting and Event Recognition experiments and submitted results based on the viewpoints that were most amenable to our methods. That is, data captured by cameras from high above

the ground with a bird's eye view of the outdoor stage, scene in Figure 1.2. In many scenarios, our methods performed very well with acceptable error. However, a few scenarios were difficult. Not due to the complexity of the crowd dynamics, however. The training of methods was strained and poor for many scenarios due to the scarcity of data with which to train and test. However, performance hampered by limited data is not of significance to the community and reveals no insight suggesting new research directions.[9]

Although there are many efforts to create a rich common experimental environment, these efforts often are costly and fall narrow in their attempts to encompass an expansive set of patterns applicable to many of our community's disentanglement efforts of visual symbols. This was the leading motivation to create a more expansive database to serve as a common experimental platform.

1.2 Current Work

The automated monitoring and surveillance of crowded scenes is a remarkable challenge for current image and video understanding technology. It has environmental application in areas such as security, natural disaster prevention, research in herd and flocking behavior, population monitoring, entertainment, urban architecture, and marketing. It has recently acquired strong societal significance, due to the possibility of terrorist attacks on events involving large concentrations of people, a problem for which there are currently no effective solutions.

At best, available databases are comprised of synchronized multiple perspectives. [9] However, they remain static, i.e. not live and not adaptable to problem-type needs. Data has been pre-selected as salient for the problem at hand. Data of this nature tends to cultivate champion techniques that master the challenges laid forth via rigorous and thorough efforts to create an expansive database. However, due to increasing optimality criteria the definition of robustness is quickly expanding laterally. How should our approach change to encompass overcoming more with less force?

A common limitation of databases, as we have mentioned previously, are their finite and static nature. This is acceptable when solving simple and independent prob-

lems. However, databases of this nature become too limiting when it is our goal to discover new problems rather than better solve existing ones.

Database design is rather arbitrary and is therefore susceptible to be limited to what we humans find experimentally salient. This is not unreasonable. After all, we are also defining the problem. However, at some point databases are being created for existing problems rather than for unknown problems. Thus, a need for more unbiased data exists so that we may discover new problems. In other words, our solutions tend too often to be akin to the proverbial Lamppost Theory, i.e. looking for our lost keys under the lamppost where there is light. This project serves as a tiny flashlight under that lamppost to expand problem-solving attention beyond and into the darkness. In other words, the intention of this project is to create an abundant yet common research environment. A database should represent an unbiased and true universe. And as we begin to draw inspiration from biological systems to solve information processing problems, we too need to make accessible to machines that in which these biological systems live and breath.

This project 1) implements a platform for automated monitoring and surveillance of crowded scenes, 2) provides a common experimental environment in which to test crowd analytics continuously and in real-time, all while 3) overcoming common limiting constraints such as static databases applicable only to subsets of problems. This project paves a pathway for new and extended crowd analytic evaluation, including: visualizing distributed crowd dynamics across an expansive spatial area as well as temporally yielding trends over extended durations. Furthermore, this project elucidates new avenues of crowd research such as 1) crowd interpolation of unmonitored network pathways, 2) object and person tracking across fields of view, and 3) crowd analysis of areas with simultaneous multiple perspectives. By making data more accessible, we expose techniques to many more signal nuances that were previously treated as noise.

1.2.1 System Overview

Seen in Figure 1.3, we have a simple flowchart illustrating our three main modules that, separately, acquire video, analyze the crowds, and visualize the results. In Figure 1.4 on page 13, we see these modules in detail.

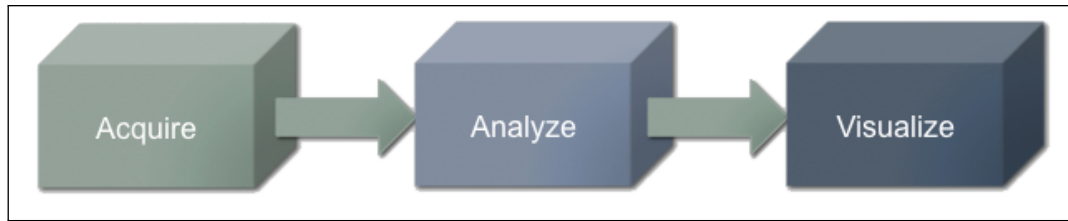


Figure 1.3: Simplified System Flowchart.

The fundamental system constraint was working with two virtual local area networks (VLANs). The first network is an industry standard Police Surveillance System manufactured by Pelco[®], and operated by the UC San Diego Police Department (VLAN 318). The second is our SVCL network (VLAN 10) containing a cluster of linux workstations for high computational capabilities. These networks are both depicted in Figure 1.4 as boxes with dotted perimeters. Creating a tunnel between them is disallowed due to university policy at the time of writing this thesis. Therefore, the Acquisition System was designed to serve as a gateway or security buffer between the two networks. In other words, the Acquisition System interfaces our analysis network with the police surveillance network. As such, it was necessary to develop this module to run on a workstation that communicates with both networks, serving as a security buffer or gateway.

As seen in Figure 1.4 below, at the top are our probes or measurement devices. This is comprised of a network of cameras, network video recorders (NVRs), and a system manager (SM) to maintain the network and provide information to service requesters so that they may directly connect to devices.

At the next level we have our Acquisition Module. This module takes in user input to specify which data to retrieve from the surveillance network. This could, for example, specify (a) from which camera we want data, (b) the pan tilt zoom (PTZ) of the desired camera, or (c) whether we want live or past-recorded data. These specifications are parsed and interpreted in our data binding and web service scheduler functions. Lastly, the desired web services are passed to the surveillance network via the Pelco API Library. Video data is then streamed and stacked to a buffer, awaiting processing by the Analysis Module. This discussion on Acquisition is continued in Chapter 2 on page 14.

The Analysis Module will then process the video queue. The previous and next

module that feed data to and from this module, respectively, are designed to be independent of the analysis module. The reason being we desire to be able to switch out this Analysis Module with others and future analytic methods to come. In fact, the analytical results reported in the Results chapter are poor under accuracy criterion. The major contribution of this work is not a novel computational model. But rather, a more robust and abundant, yet common, experimental environment.

Looking inside our Analysis Module, data is first converted to grayscale and resized from 1280x720 to 320x180 and cropped to 320x160 (to remove a timestamp bar). Secondly, a pre-trained DTM model is loaded to segment the video into classes. The segment features are then learned and used by the predictor to estimate the counts of each class, i.e. the *Results*. To perform the prediction, we load a pre-trained count model. For a typical measurement we have 200 frames, each with a count prediction times number of classes (typically two), i.e. 400 integer predictions. These values are averaged and saved sequentially in a results database. This discussion is continued in Chapter 3 on page 34.

The results database is the foundation of our Visualization Module. The database is an SQL like web service database called Google Fusion Tables. Fusion Tables are an accessible storage solution for the geospatial-based results via a subset of SQL commands. These tables allow for quick filtered querying of results and can be integrated with the Google Maps API and Google Charts API. This discussion is continued in Chapter 4 on page 47.

For clarification, crowd dynamics are stationary from second-to-second. Therefore, a crowd-counting system need not be truly real-time, i.e. counting number of individuals in each frame. On the contrary, it is sufficient that crowd measurements occur minute-to-minute and to allow intermediate crowd-count predictions to be interpolated. Therefore, the real-time requirement can be relaxed and this system is allowed to work in quasi-real-time, i.e. discrete measurements are made minute to minute and reported with significant lag. If this is seen as a shortcoming in the future, we provide a discussion outlining overcoming this constraint in Future Work, Chapter 5 on page 56.

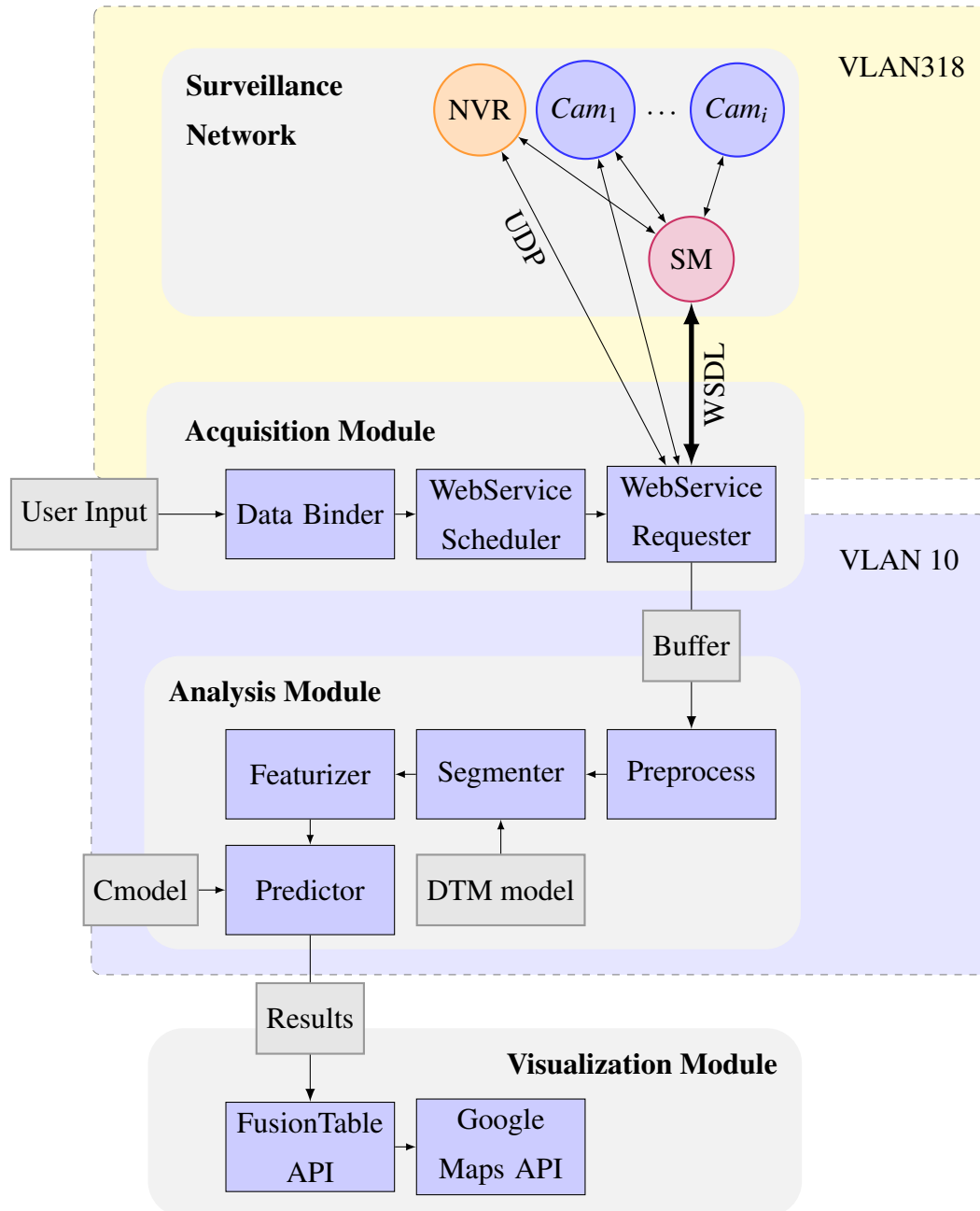


Figure 1.4: System Diagram. Each background box with a dotted outline represents a VLAN while each gray rounded-rectangle represents an application module. The surveillance network resides in the VLAN 318, the top box. The Analysis Module resides in our lab's VLAN 10, bottom box. Straddling both networks is our Acquisition Module. Outside of these networks lies the world wide web where the Visualization Module resides.

2 Acquisition Methods

In Chapter 1, we have already seen a full system layout of the network hardware and application modules in Figure 1.4: System Diagram. Here in Chapter 2 we will focus on the Surveillance Network and Acquisition Module, compare Figure 2.1 on page 16. In this section we provide a more detailed overview of the network hardware and the architecture used so that we may ultimately formulate an automated video acquisition platform to feed our automated Analysis Module. A brief outline of components and data-flow will be illustrated. For application source code, please see Appendix ??.

Working with the crowd counting system in Section 1.1, we knew that the ideal camera setup includes a birds eye view of a fairly dense pathway of pedestrians. Aware of the most popular and salient pathways on the UCSD campus, we first sought to install a surveillance camera above the entrance to UCSD's iconic Geisel Library. There exist considerable interest in creating a visual analytic analogue of turnstile so that richer building entrance statistics and early detection of emergency events could be made possible.

Upon investigation, however, the area proved to be highly political. The area happens to be a crossroads in surveillance jurisdiction between UCSD's Police Department and Geisel Library's own security team, whose primary tasks include privacy-preservation of the library's special collections and of library patrons. Although the library is adamant in disallowing filming in the library's interior, they were quite open to the idea of external entrance monitoring, even if it meant we would be using the Police Department's network infrastructure. This and the promise of automated crowd counting was incentive enough for the Police Department to support my efforts, which ultimately opened up SVCL's access to their rich surveillance network.

Since then, however, any installation of electronics along Geisel Library's con-

crete exterior walls is next to impossible without severe and permanent remodeling due to the building's architecture and age. A task neither the Library's Facility Manager nor SVCL would be willing to undertake for an exploratory project.

Abandoning this effort, we focused on our plethora of options around Price Center. Many cameras are planted high above public pathways and are operational 24/7. This rich database contains many crowded scenes ideal for preliminary development as well as many options to challenge analytics in the future. However, as the surveillance system is a shared resource, we are presented with an interesting challenge to minimally interfere with this dynamic multi-agent control system. Furthermore, we would need a method for automatically scheduling video streams that take into account the variety of communication protocols with which each camera works. As we will see, the surveillance system manufacturer (Pelco by Snieder) has recently started providing solutions and support to deal with these challenges. Although in its infancy during the lifetime of this project and slow to respond, the API support team was helpful in working with their products and working around their software bugs. In the next sections, we will discuss the Surveillance Network's hardware infrastructure and its Software Development Kit so that we may better understand how our Acquisition Module must operate.

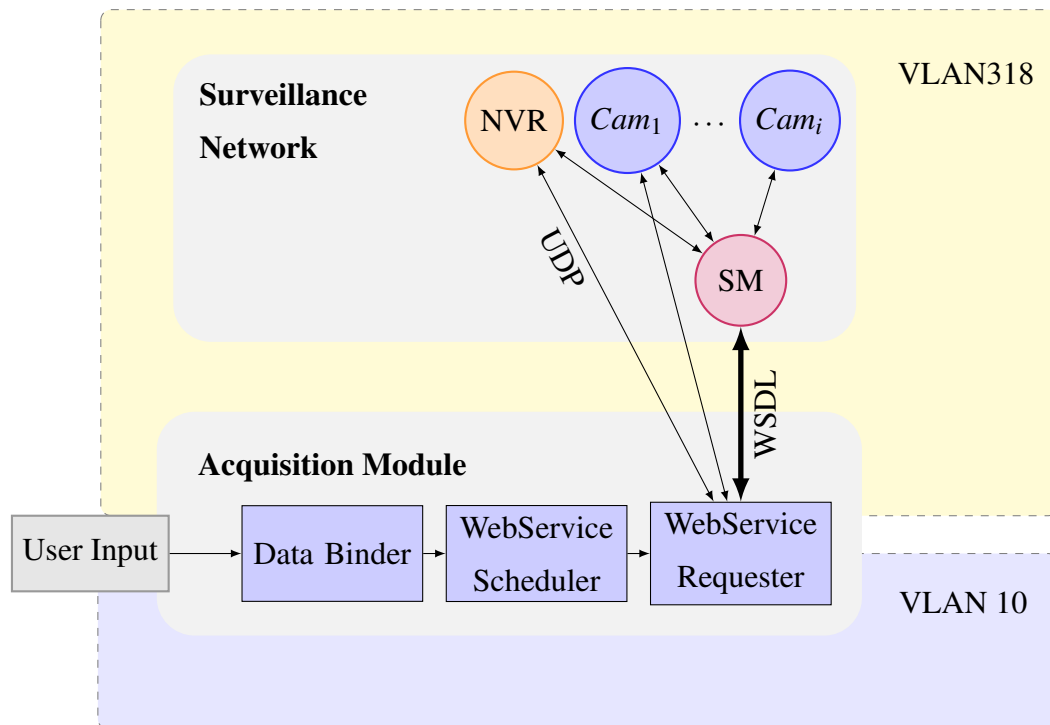


Figure 2.1: Acquisition and Surveillance System Diagram. Shown above is the same Acquisition Module and Surveillance Network previous shown in Figure 1.4: System Diagram on page 13.

2.1 Surveillance Network Hardware

This video network is owned and operated by the UC San Diego Police Department and the UC San Diego Bookstore's Crime and Theft Prevention Team. The network, itself, was developed by Pelco, Inc. by Schneider Electric. Pelco is a leading manufacturer in video and security systems. There are a few different device types relevant to our project and which we will discuss in the next few subsections: Cameras, Network Video Recorders (NVR), and System Managers (SM).

2.1.1 Cameras

Pelco cameras are equipped with industry standard features, see Table 2.1. Our current system uses two basic types of cameras: Static and Pan-Tilt-Zoom (PTZ). The only practical difference is that PTZ cameras are housed in a robotic dome, giving them their PTZ features.

Table 2.1: Pelco Camera Features

| <i>Pelco Camera Features</i> |
|--|
| Up to 1.2 Megapixel (MP) Resolution (1280 x 960) |
| Up to 30 Images per Second (ips) at 1280 x 960 |
| Auto Back Focus |
| H.264, MPEG-4 and MJPEG Compression Capability |
| Day/Night Models with Mechanical IR Cut Filter |
| Wide Dynamic Range with Anti-Bloom Technology |
| Power over Ethernet (IEEE 802.3af) or 24 VAC |
| Up to 2 Simultaneous Video Streams |
| Built-In Analytics |
| Local Storage (Mini SD) for Alarm Capture |
| Open IP Standards |
| Pan Tilt Zoom Dome Enclosures |
| Motion Detection |

Camera Locations

Plan-view and satellite view showing camera locations are provided to contextualize the network as well as demonstrate its expanse. Google does not offer print quality copies of their maps. The following are, unfortunately, screen captures. Disclaimer: Google Permissions[10] states screen captures, as seen here, may be published and reproduced with out their permission.

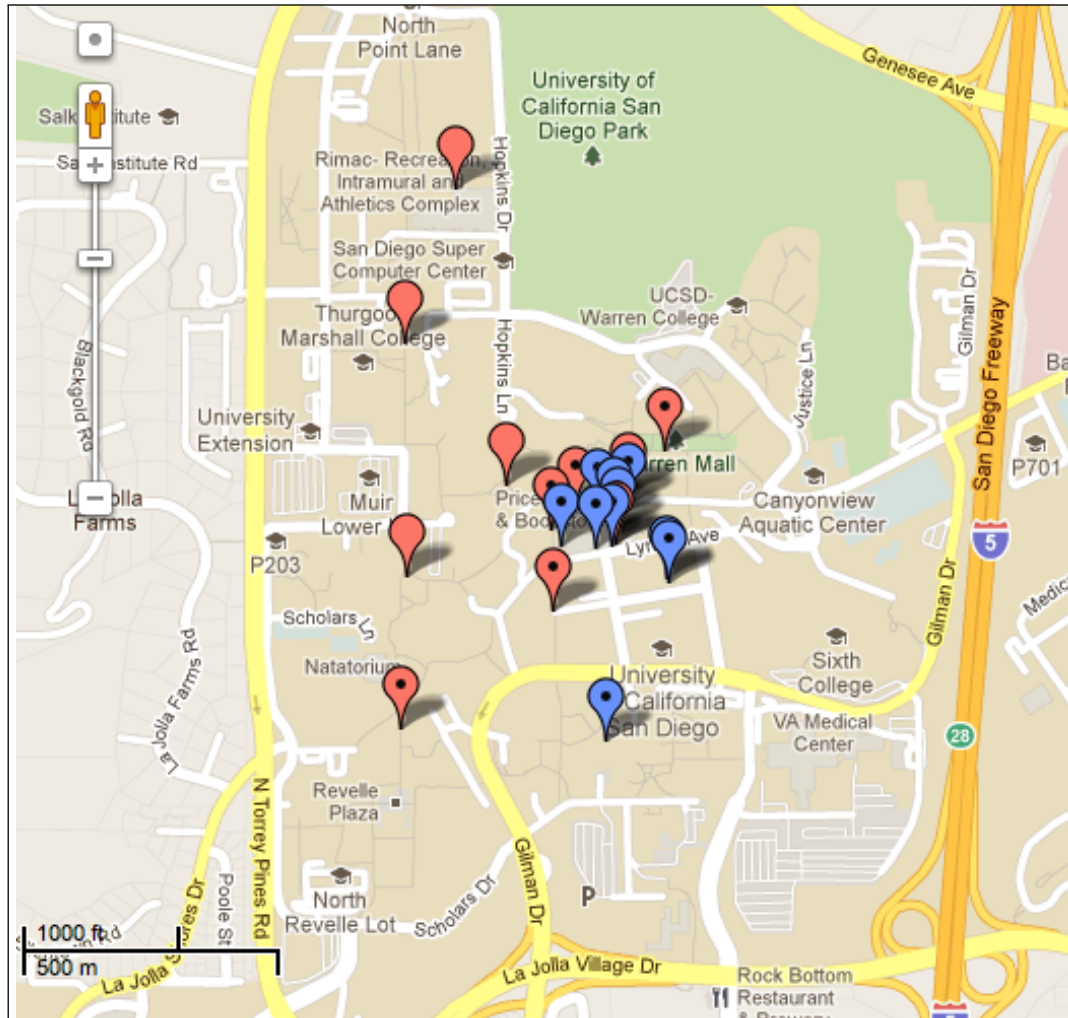


Figure 2.2: Locations of salient cameras on the main UCSD Campus. The approximate geospatial location (latitude, longitude) of the center of this map is (32.8800, -117.2371). At the center of the map is a cluster of cameras. This cluster coincides with the center of campus, Price Center (Detail in Figure 2.3). Blue Markers mark static cameras. Red markers mark PTZ cameras.[11]

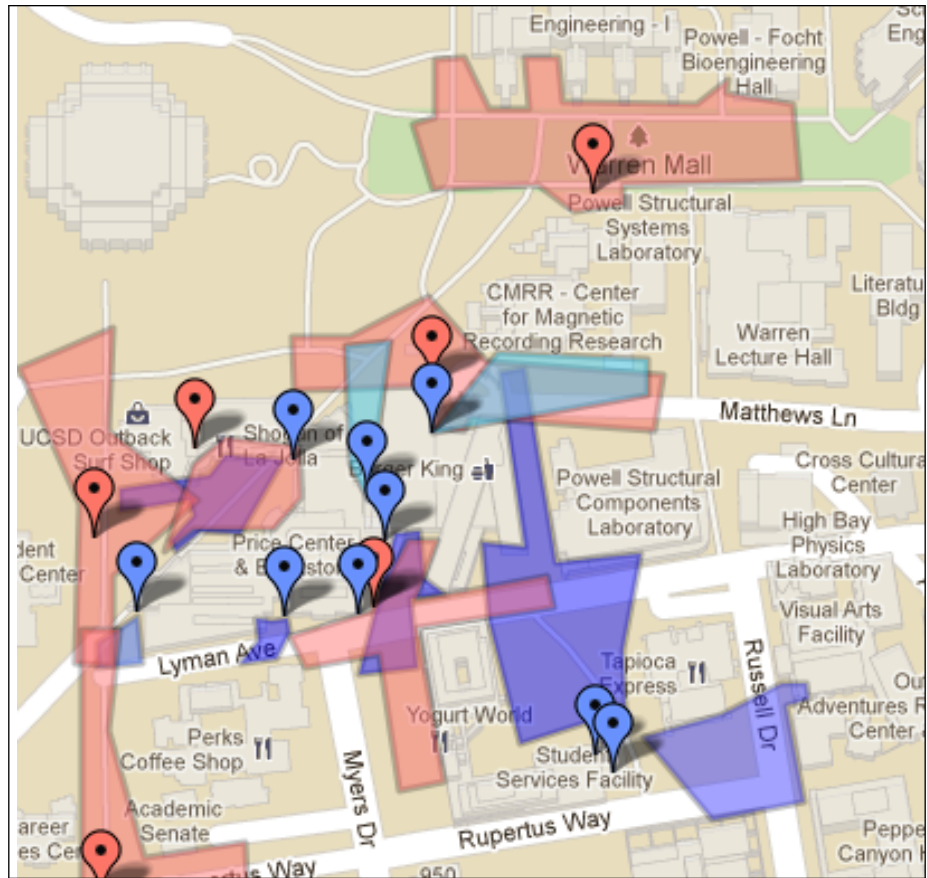


Figure 2.3: Camera Locations and Field-of-Views - Price Center Map. We gave greatest attention to the scene located just above *Student Services Facility*, the scene middistance from *Yogurt World* and *Burger Kind*, and the long pathway just below *UCSD Outback Surf Shop*. The Warren Mall area of the top of the map is another interesting isolated scene. Also, the intersecting fields-of-view at the left end of Lyman Ave is an interesting multi viewpoint scene.

2.1.2 System Manager

The SM of this networked system can be thought of as the central administrator. The SM not only serves as a switch board to connect devices, it ultimately keeps every device clock synchronized and serves as an indexing of attributes.

2.1.3 Network Video Recorders

NVRs are the primary storage centers for the large quantities of data required by recorded video. However, NVRs also serve as encoders and indexing agents; they actively re-encode, compress, index, and backup data per SM requests. These devices are key to the exportation process occurring in our acquisition application, especially for PTZ cameras.

Since cameras are shared devices, PTZ camera orientation may be, and often is, changed at any given time by officers currently viewing the same multi-cast stream. Since our analysis module currently only deals with static-views, it is essential we ensure consistent views when acquiring data. Furthermore, since acquisition is a periodic process, e.g. we record 20 seconds every 10 minutes, we require a method of ensuring this consistency while sharing the cameras. To overcome this problem, we adjust camera orientation prior to each acquisition and export the clip only after ensuring parameter constraints exist for the length of the record. Therefore, it is essential to communicate not only with cameras during this process, but also the the NVRs as well.

2.2 Surveillance Network Software

In this section we provide an overview of the software and communications side of the surveillance network. Namely, an overview of standard communications protocols, such as SOAP and Web Services, and how they link applications together. This discussion will not be thorough nor highly technical. It will provide function descriptions so that the reader may follow the processing flow.

2.2.1 Pelco SDK Libraries

The Pelco Software Development Kit (SDK) is a Pelco distributed software bundle comprised of core resource libraries that provide developers with the ability of interfacing C++ applications with Pelco Surveillance Systems. See Table 2.2 on page 22 for a list of the SDK libraries and their description. This project primarily employs the exporter library for data acquisition. However, the use of the PTZ Control Wrapper library was also necessary to control PTZ camera settings and the GSOAP library to control the video stream quality.

Web Services

Web services is an enterprise standard when applications need to communicate with one another. Simply put, Web Services is a sum of XML and HTTP in that data content is structured in the XML format and transferred over HTTP.

Microsoft Visual C++ 2008 Express Edition

An integrated development environment required by Pelco's Software Development Kit, this software application provided the C++ source code editor, linker, compiler and debugger necessary for developing our acquisition software discussed in Section 2.3.

Table 2.2: Pelco SDK Libraries

| <i>Pelco SDK Libraries</i> | |
|----------------------------|--|
| System Manager Wrapper | Provides Device and Service Discovery web services library. |
| API Viewer | Library for controlling and viewing streams from cameras and NVRs. |
| PTZ Control Wrapper | Library that enables PTZ (Pan, Tilt and Zoom) camera action calls on Endura network (which includes returning the absolute position of Pelco camera domes), as well as encompassing Preset and Pattern action calls. |
| Meta-data Parser | Library for parsing timestamps and built-in motion analytics. |
| Exporter | Library providing capture functionality of live and playback video. Can export queried video in AVI, MP4, 3GP, or PEF formats. |
| GSOAP | PelcoGSoap library provides C++ interface for SOAP clients to make SOAP calls to Endura devices. |

2.2.2 Relevant Functions

Here we list the functions central to our exporter module to better illustrate the exporter's functionality.. We include their Namespace, Class, and argument data types.[12] Function inputs are generated by our timing schedule and user input discussed previously.

Export Initialization Function

This function sets up the exporter module. The sm ip address, and the ip address of the interface to use for incoming streams are passed.

Usage:

```
PELCO_API_EXPORT void PelcoAPI::EnduraExporter::Setup
(const char * sPluginDir, const char * sSmAddress, const char *
sLocalAddress, const char * userName = NULL, int nStartPort =
8000, int nEndPort = - 1, bool bQAppCreated = false)
```

Parameters:

| | |
|---------------|---|
| sPluginDir | The API Plugin Directory |
| sSmAddress | The SystemManager IP Address |
| sLocalAddress | The IP Address of the interface to receive incoming streams |
| nStartPort | The start port number |
| nEndPort | The end port number |
| bQAppCreated | The QApplication is created outside the exporter or not |

Returns: void

Export Start Function

Start a new export using the setup data and the information passed in the parameters.

Usage:

```
PELCO_API_EXPORT bool PelcoAPI::EnduraExporter::StartExport (
const char * sFileInfo, const char * sDeviceID, VideoCodecType
OutputCodec, const char * sStartTime, const char * sEndTime,
bool bVideoOnly = true)
```

Example:

```
bool bSuccess = exporter.StartExport (
    "D:\\Video\\test123.avi",
    "uuid:691fd745-006c-4fc9-9262-23c13e977ce4",
    PelcoAPI::CODEC_ID_MPEG4, "2010-01-11T22:10:35", "2010-01-11T22:11:15",
    false, "uuid:691fd745-006c-4fc9-\\9822-23c13e977ce4");
```

Parameters:

| | |
|-------------|---|
| sFileInfo | Output file path |
| OutputCodec | Output video codec |
| sDeviceID | Camera device UUID |
| sStartTime | Start time for recording - NOW for live or UTC time (e.g. 2009-05-28T16:30:00) for playback |
| sEndTime | End time for recording |
| bVideoOnly | Boolean value for export video only or export with audio if it exists |

Returns: success or failure

PTZ Absolute Move

Sets the IP camera's viewing position. If successful this will return true; it will return false otherwise. You can tilt and pan at the same time, which will move the camera view diagonally.

```
PELCO_API_EXPORT bool PelcoAPI::PTZControlWrapper::AbsoluteMove
(int positionX, int positionY)
```

Parameters:

| | |
|-----------|---|
| positionX | A negative integer value will pan the camera left. A positive integer will pan the camera right. Valid possible values are -360,000 micro-degrees to 360,000 micro-degrees. |
| positionY | A negative integer value will tilt the camera down. A positive integer will tilt the camera up. Valid values approximately range from -90,000 to 90,000 micro-degrees. |

Returns: bool If successful this will return true; it will return false otherwise.

PTZ Get Absolute Position

Returns the IP camera's current viewing position.

Usage:

```
PELCO_API_EXPORT bool PelcoAPI::PTZControlWrapper::GetPosition
(int positionX, int positionY)
```

Parameters:

| | |
|-----------|--|
| positionX | This is the camera's current position on the rotational X plane. |
| positionY | This is the camera's current position on the rotational Y plane. |

Returns: bool If successful this will return true; it will return false otherwise.

PTZ Absolute Zoom

Sets the IP camera's zoom level.

Usage:

```
PELCO_API_EXPORT bool PelcoAPI::PTZControlWrapper::AbsoluteZoom
(int zoom)
```

Parameters:

| | |
|------|---|
| zoom | Integer. The desired magnification value. |
|------|---|

Returns: bool If successful this will return true; it will return false otherwise.

PTZ Get Absolute Zoom

Returns the IP camera's current zoom level.

Usage:

```
PELCO_API_EXPORT bool PelcoAPI::PTZControlWrapper::GetAbsoluteZoom
(int & zoom)
```

Parameters:

| | |
|------|--------------------------|
| zoom | A pointer to the result. |
|------|--------------------------|

Returns: bool If successful this will return true; it will return false otherwise.

PTZ Goto Preset

Brings the device to a specific preset state, given the name of the desired preset to execute.

Usage:

```
PELCO_API_EXPORT bool PelcoAPI::PTZControlWrapper::GotoPreset
(const char * presetName)
```

Parameters:

| | |
|------------|--|
| presetName | The name of the preset to execute. This is the format for the preset-Name: PRESETx, where x is the actual name of your preset. e.g. A preset is named 100. Consequently a valid parameter for presetName would be PRESET100. |
|------------|--|

Returns: bool If successful this will return true; it will return false otherwise.

PTZ Set Preset

Sets a preset or pattern. Depending on whether they already exist or not, it will either create a new preset or pattern or modify an existing one.

Usage:

```
PELCO_API_EXPORT bool PelcoAPI::PTZControlWrapper::SetPreset
(const char * presetName)
```

Parameters:

| | |
|------------|---|
| presetName | The name of the preset or pattern to be either created or modified. |
|------------|---|

Returns: bool If successful this will return true; it will return false otherwise.

isPTZCamera

Check if it is a PTZ camera. If it is a PTZ camera this will return true; it will return false otherwise.

Usage:

```
PELCO_API_EXPORT bool PelcoAPI::PTZControlWrapper::IsPTZCamera ()
```


Parameters:

| | |
|------------|--|
| presetName | The name of the preset to execute. This is the format for the preset-Name: PRESETx, where x is the actual name of your preset. e.g. A preset is named 100. Consequently a valid parameter for presetName would be PRESET100. |
|------------|--|

Returns:

bool If successful this will return true; it will return false otherwise.

Poll Status

The combined usage of these functions with appropriate parameters will pull data from any camera online during the desired schedule. Additionally, using the following function to poll the status of the data stream may be used:

Usage:

```
PELCO_API_EXPORT int PelcoAPI::EnduraExporter::PollStatus
(int timeout = 60)
```

Returns: the status of the export or -1 on error

2.3 Acquisition Module

Module responsible for interfacing our Surveillance Network and Computational Network. This module uses the API libraries listed in Table 2.2 on page 22 to enable command-line functionality, making automated and controlled exportation of video streams possible. This automated platform serves as a rich and common experimental environment. The openness of our database is attributed to the real-time streams from our live system; in the physical system sense, content flows into our database *openly*. The dynamics of the database are attributed to dynamic devices such as our PTZ cameras.

2.3.1 Exporter Application

This application accepts user input to specify which data to retrieve from the surveillance network. This could, for example, specify (i) from which camera we want

data, (ii) the pan tilt zoom (PTZ) of the desired camera, or (iii) whether we want live or past-recorded data. These specifications are parsed and interpreted in our data binding and web-service scheduler functions. For example, if we specify a data stream from a live and PTZ camera, our web-services scheduler will load the PTZ control library using a rule-based algorithm. Lastly, our scheduler passes the correct arguments to the API Library Export function. The library interprets the input and communicates with the specified devices using the necessary web-services. Video data is finally streamed and stacked to a buffer, awaiting processing by the Analysis Module.

This application was compiled using Microsoft Visual C++ 2008 on a windows machine running Windows XP Service Pack 3. After specifying the timing and camera parameters previously discussed, this application can be run on any machine running windows xp and connected to the Surveillance Network. Although it is not required, saving video streams to the Analysis Network requires a secondary network interface card be installed on the machine and a Analysis Network file system be mounted on our local file system. Under our current configuration, we have mounted a public network drive that is also mounted on our SVCL Analysis Network. Under this configuration, our video stream is buffered to a network file system common to both networks.

2.3.2 Timing Schema

User Input Schema is divided into two sections: Timing Parameters and Scene Configuration. User input is specified in a text file. Timing Parameters specify the timing schedule illustrated, right. Scene Configuration specifies camera settings (i.e. which camera, PTZ coordinates, codec compression, resolution, etc.). For example: if we request a 20-second clip every 10 minutes and also specify two scenes, we will receive two 20-second clips every 10 minutes; one for each scene. The illustration to in Figure 2.4 shows the timing schedule containing two clips, though in practice many more would likely be scheduled, and not necessarily from the same scene. Their start times, respectively, are $t=k$ and $t=k+1$. Not all cameras will require LAG1, e.g. static cameras. This application was designed to cycle through multiple scenes/cameras to gather data from multiple areas of interest. Although Pelco software supports multiple streams, this application cycles through scenes and exports single clips at a time until

the lifetime of the experiment expires.

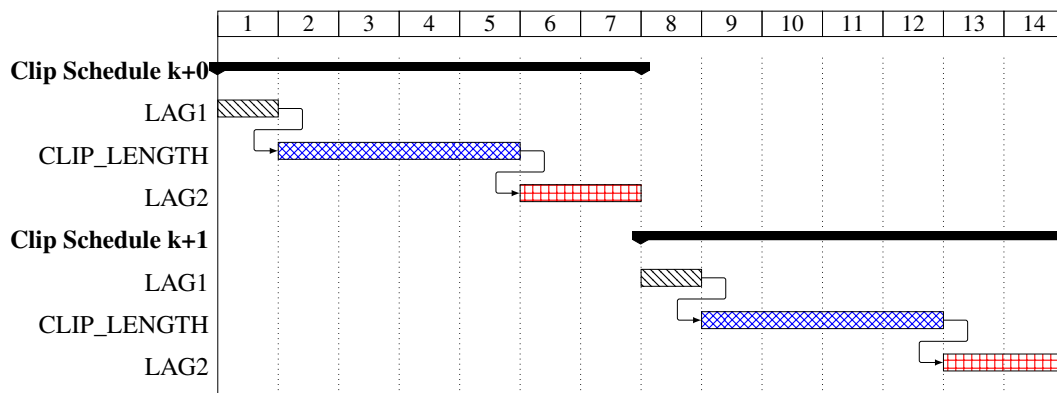


Figure 2.4: Export Schedule

2.3.3 Video Stream Parameterization

In acquiring data, we specify data parameters and run the Exporter C++ application on our windows workstation (Acquisition Module). Parameters are specified in a structured configuration text file with extension *.cfg*, a similar yet simplified and more compact form of an xml schema, which is bound using the open-source shared C++ library *libconfig*. [13] Below we include the configuration schema and an example configuration file. In Table 2.3, we list the schema datatypes seen in the schema and example below.

Table 2.3: Configuration File Datatypes

| | | |
|----------|-------------------------------|--|
| Settings | < name = value > | Values may be a scalar value, an array, a group, or a list. |
| Groups | < { setting, setting, ... } > | Groups may contain any number of settings. |
| Lists | < (value, value, ...) > | We use lists to represent multiple scenes and each value is a group of scene settings. |

The format of the configuration file is as follows. At the root is a setting with the name *configuration*. Its value is a group. This group contains two settings, named *params* (short for parameters) and *scenes*. (i.e. *configuration = {params; scenes};*)

Table 2.4: Configuration File Format

| <i>params</i> Setting Format | <i>sceneX</i> Group Format |
|------------------------------|--|
| params=({ | {<sceneX>} = { |
| CLIP_LENGTH = <seconds>; | friendlyName = <sceneName>; |
| FREQ= <seconds>; | CAMERA_IP_ADDRESS= <XXX.yyy.XXX.yyy>; |
| LIFETIME= <seconds>; | PORT_NUMBER = 80; |
| UTC= <seconds>; | CAMERA_NUMBER = 1; |
| LAG1= <seconds>; | CAMERA_udn = uuid:<uuid>; |
| LAG2= <seconds>; | EXPORT_FOLDER = <directory>; |
| *STARTINGoffset= <seconds> | PREFIX = “vid”; |
| }); | isPTZ=<bool>; |
| | *PRESET = <PRESETXX>; |
| | *intPRESET = <PRESETYY>; |
| | } |

The *params* setting’s value is a list containing a group specifying global data settings such as clip length, frequency, and the lifetime (how many clips to export). In other words, the *params* structure defines the *when*. (i.e. *params*=({ <setting>; <setting>; ... }).)

The *scenes* setting’s value is a list of scenes, each containing a group of settings. Each group of settings define a scene. These scene parameters include the name and location of a camera, the camera’s settings, and the destination of the video stream (scene buffer or export folder). In other words, the *scenes* structure defines the *where*. (i.e. *scenes*=({<scene1>}, {<scene2>}, ... }); .)

This format is shown in Table 2.4 (an asterisk signifies the setting is optional):

Table 2.5 outlines each of the settings contained in the *params* setting and its expected input type. Table 2.6 outlines the settings contained in each *sceneX* group listed in *scenes*.

Table 2.5: Acquisition Module Export Parameters

| <i>params</i> | | |
|-----------------|--------------|---|
| Name | Units | Description |
| CLIP_LENGTH | seconds | Length of each video clip. Usually 20 seconds. |
| FREQ | seconds | Defines the minimum elapsed time required before exporting new data. |
| LIFETIME | seconds | Maximum elapsed time allowed from start of export. |
| UTC | seconds | Current distance, in time, workstation clock is from Universal Time |
| LAG1 | seconds | Time needed by system to ensure PTZ settings take effect before recording. |
| LAG2 | seconds | Time needed by system to ensure recorded clip is ready for export. |
| *STARTINGoffset | seconds | Relative distance from present to past. Used to export non-live video recorded in past. |

Table 2.6: Acquisition Module Scene Settings

| <i>sceneX</i> | | |
|-------------------|--------------|--|
| Name | Units | Description |
| CAMERA_IP_ADDRESS | IPv4 | Used for camera control, i.e. PanTilt-Zoom (PTZ). |
| PORT_NUMBER | int | Typically 80. |
| CAMERA_NUMBER | int | Typically 1. |
| CAMERA_udn | uuid | Universally Unique Identifier of camera for exporting recorded data. |
| EXPORT_FOLDER | directory | Scene-specific storage location of all exported data. |
| PREFIX | string | Arbitrary Video File Prefix. |
| isPTZ | true/false | Flag indicating whether a camera's scene can be adjusted. If true, exporter expects the following PRESET settings to be specified. |
| *PRESET | PRESETXX | Specifies which scene is desired from a camera. XX should be replaced with an integer. |
| *intPRESET | PRESETYY | Dummy name used to save initial camera settings so they may be returned after export. |

Here is an example of a *.cfg file*:

```
1  # Example Configuration File
2  configuration =
3  {
4      params = ( {
5          CLIP_LENGTH      = 20;
6          FREQ             = 30;
7          LIFETIME         = 3600;
8          UTC              = 28800;
9          LAG1             = 2; // Wait for PTZ.
10         LAG2             = 20; // Wait for NVR
11         //STARTINGoffset = 17200;
12     });
13     //END params
14
15     cameras = (
16         //SCENE 1
17         {
18             friendlyName  = "scen1";
19             CAMERA_IP_ADDRESS = "172.31.200.9";
20             PORT_NUMBER    = 80;
21             CAMERA_NUMBER  = 1;
22             CAMERA_udn     = "uuid:31990bfb-c356-75a3-2cb8-114277a23904";
23             EXPORT_FOLDER  = "Z:\\Mulloy\\Incoming\\scen1\\pef\\";
24             PREFIX         = "vid";
25             isPTZ          = true;
26             PRESET         = "PRESET56";
27             intPRESET      = "PRESET57";
28         },
29         //SCENE 2
30         //
31     );
32     //END scene list
33 };
```

3 Analysis Methods

This chapter will discuss the Analysis Module, which is an automated people counting system. The foundation of this counting system was built by SVCL Alumnus Professor Antoni Chan. Upon this foundation we have built an automated system that uses the segmentation, featurizing, and prediction computer vision and machine learning methods discussed in [2, 3, 4, 9, 14]. However, we additionally implement a variant technique to allow for more complex motion modeling (discussed in Section 3.2.1 on page 36).

This chapter is divided into three sections: The first is a brief overview of our automated counting system without technical details; The second covers the computational tools necessary to build a people counting model in technical detail; And the third we discuss in technical detail the usage of these tools in our automated system and its implementation.

3.1 Crowd-Counting Overview

Implemented on our SVCL Linux Cluster in C code, our automated crowd-counting module is illustrated in Figure 3.1. We focus on scenes in which there are many pedestrians with a collectively similar trajectory and/or a collectively opposing trajectory. For example, 15 persons may be moving northernly and while 7 persons move southernly on a shared pathway. Persons naturally will form clusters to benefit their unobstructed motion. These crowds can be characterized by a collective action and holistic motion. Since Dynamic Textures (DT) are good models for holistic motion, these scenes lend themselves to motion segmentation based on DT models (DTM).

Once we have trained our models on these contrasting holistic motions, features

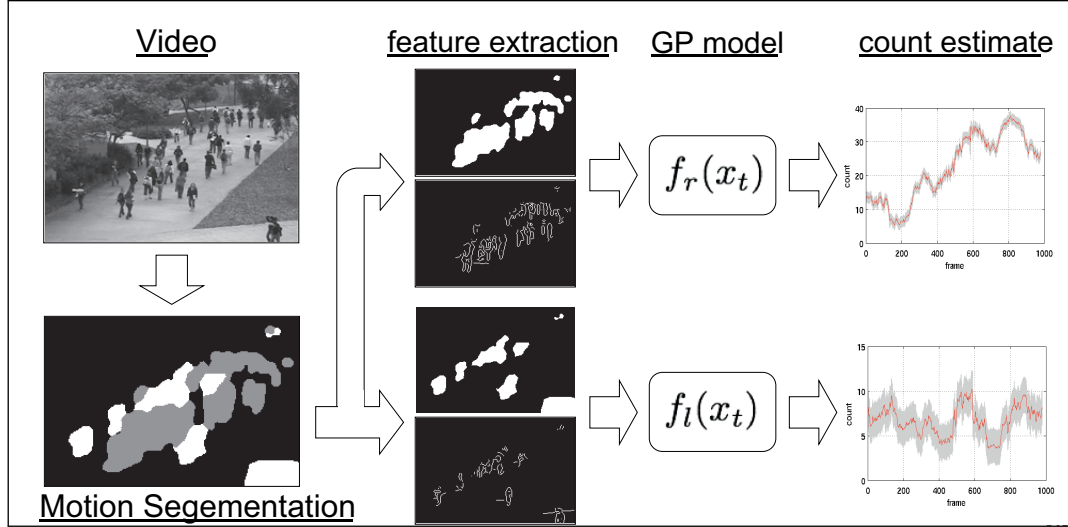


Figure 3.1: Crowd-Count System Flowchart [3]. First, we perform motion segmentation based on DTMs. Next, separate segments by class and perform feature extraction. After training the GP model via maximizing marginal likelihood on training data, we may obtain count estimates of segments based on their features via regression.

like segmentation-blob area may be extracted. Since it is our goal to build a people count model, we learn a Gaussian Process Regression that may relate the extracted features to integer count estimates. Our computational tools are borrowed from the SVCL C and Matlab repositories. Some functions, especially the Matlab, are not optimized for computational efficiency. Rather than re-write these functions in our C library, we have incorporated multi-threading techniques in our wrapper applications where appropriate to reduce computational time.

3.2 Computational Tools

This section covers the technical details of the elements in Figure 3.1. These include: (a) Dynamic Texture Model, (b) Motion Segmentation, (c) Features, and (d) Regression.

3.2.1 Dynamic Texture Model

A Dynamic Texture is a generative probabilistic motion model suitable for holistic representation of crowds. Under this model, video frames and content motion are treated as a sampling of a latent motion model, respectively. In Figure 3.2, observations (video frames) are the sampling of a Linear Dynamic System (LDS),

$$\begin{cases} y_t = Cx_t + w_t \\ x_t = Ax_{t-1} + v_t \end{cases}$$

where,

| | |
|--|--|
| $\mathbf{x}_t \in \mathbb{R}^n$ | hidden state encoding the dynamics of the video over time. |
| $\mathbf{y}_t \in \mathbb{R}^m$ | observation vector encoding video frame at time t . |
| $\mathbf{A} \in \mathbb{R}^{n \times n}$ | transition matrix that controls hidden layer evolution and motion dynamics. |
| $\mathbf{H} \in \mathbb{R}^{m \times n}$ | observation matrix responsible for projecting from our hidden state space to our sample space. |
| $\mathbf{v}_t \sim \mathcal{N}(0, Q)$ | Normally distributed additive noise term. $Q \in \mathbb{R}^{n \times n}$ |
| $\mathbf{w}_t \sim \mathcal{N}(0, R)$ | Normally distributed additive noise term. $R \in rI_m$ |

This technique [15] is useful for learning holistic models of a crowded scene, given the crowd dynamics are stationary. However, what of scenes composed of multiple crowds and/or non-stationary dynamics? For this, we take this technique a step further by modeling each holistic motion class separately with a component of our mixture model, as in [4] and again in our previous work [9].

When working with well-behaved crowds on narrow pathways, holistic motion can be divided into 2 or 3 classes. There are people walking left or right, up or down, northwest or southeast, and so on. Motion of anomalies, such as bicyclists, runners, physically disabled, and golf carts, is constrained to lie closer to a dynamics median. However in open spaces, the motion of anomalies is much more pronounced. Cyclists and runners have more room to navigate and retain their dynamics, created substantially separate classes of motion. Conceptually, imagine two motion histograms characterizing these two cases, narrow spaces and open spaces. Similarly, both histograms will

possess large peaks associated with zero motion and with small background motion. Furthermore, both with contain a wide peak reflecting normal crowd motion mixed with anomalies. However, in the latter open space we expect to also see additional distinct peaks reflecting anomalies of greater liberties, i.e. cyclists, runner and golf carts.

In our previous work [9], crowds were well behaved actors. Our data allowed for a rather eloquent solution: a single Gaussian Mixture to model all motion of interest well. However, two problems arise when employing this same technique on crowds that are not well behaved nor on narrow pathways. First, collective and holistic motion become more difficult to define. For example, if it is our goal to discriminate between crowds moving in opposing directions, how do we classify perpendicular crowd movement? Second, training data becomes sparse. Training a single mixture model, where each mixture component represents a class of motion, presumes all classes are continuously present in our training data. However, in open spaces crowds are more dispersed as well as more numerous. Therefore, the likelihood that all crowd classes are present at any given time quickly diminishes along with our technique's efficacy and ease of training.

In an effort to salvage this technique and capture the complexity of motion, we attempted to model each opposing crowd with its own mixture model. For example, to discriminate northbound and southbound crowds we learn two separate mixture models for each of these two crowds, rather than a single mixture model with two components. Whereas before we attempted to capture all northbound crowds (comprised of sub-classes: walkers, runners, cyclists, and golf-carts) in a single mixture component, we now define all these motions as a complex collective motion and model each subclass with a single component. All preceding and following techniques may remain intact.

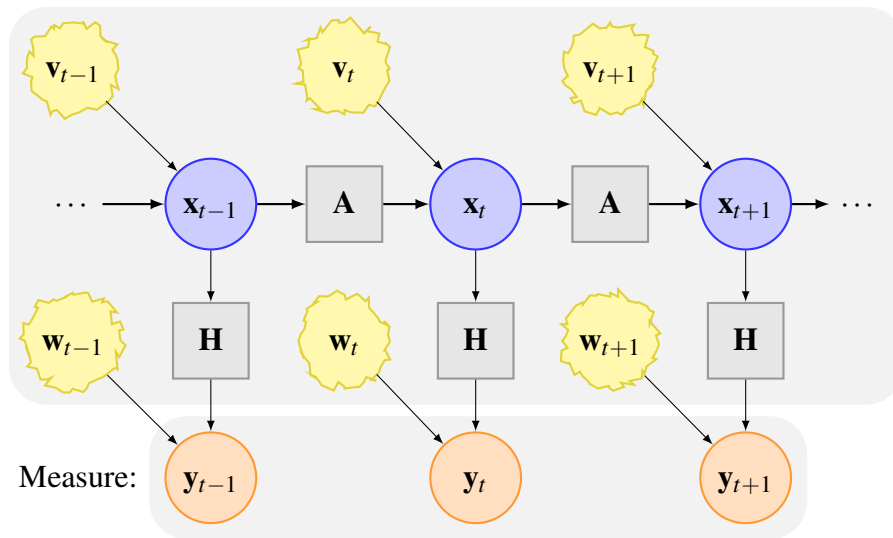


Figure 3.2: Dynamic Texture Model (DTM) Diagram. A DTM has two layers. In the first layer, \mathbf{x}_t is our hidden state and \mathbf{A} is our transition matrix. In the measurement layer, \mathbf{y}_t is our observations (video frames) and \mathbf{H} is our observation matrix. \mathbf{v}_t and \mathbf{w}_t are our normally distributed additive noise terms.[15] Figure adapted from [16].

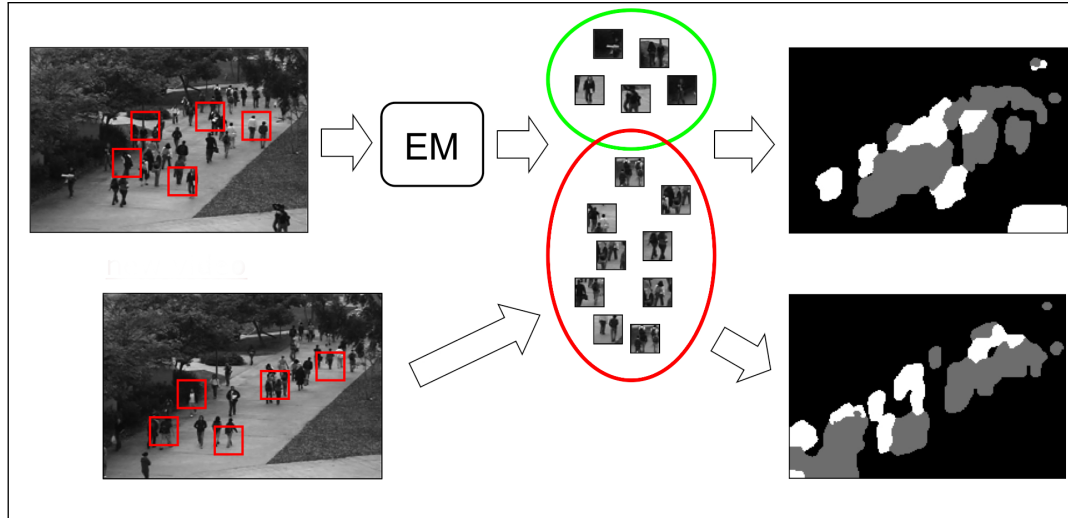


Figure 3.3: Motion Segmentation Flowchart [4]. First (left) video is cut into video cubes, typically $7 \times 7 \times 20$. Next (middle) we learn a mixture of dynamic textures and cluster the cubes to form classes. Lastly (right), we perform motion segmentation by assigning pixels to clusters.

3.2.2 Motion Segmentation

As the previous section elucidated, we may model multiple motions simultaneously via learning a gaussian *mixture* model using the ubiquitous Expectation Maximization (EM) algorithm. Trained models can then be loaded on demand for the purpose of classifying video patches, sequentially. However, when using previous methods (where opposing motions are modeled by mixture components) patches may be readily assigned a class and clustered based on mixture components of largest posterior probability, declaring the patch location as belonging to a segmentation region associated with this component. [4] However, in our new technique we modify our modeling so that multiple mixture models are trained, one for each opposing complex motion. Therefore, this modification requires assigning patch classes based on mixture model of largest likelihood. Figure 3.3 illustrates both methods.

Table 3.1: Segmentation Features Table

| | |
|---|--|
| Segment Features These features capture segment shape and size. | |
| area | number of pixels in segment. |
| perimeter | number of pixels on segment perimeter |
| edge orientation | orientation histogram of perimeter |
| perimeter-area ratio | measures complexity of segment shape |
| blob count | number of connected components with more than 10 pixels in the segment |
| Internal edge Features The edge is extracted using a Canny edge detector, which is highly indicative of number of persons [17] [18]. | |
| Total edge pixels | number of pixel contained in segment |
| Edge orientation | 6-bin histogram of the edge orientation in the segment. |
| Minkowski dimension | fractal dimension of the internal edges, which estimates degree of space-filling of edges.[19] |
| Texture Features computed using GLCM. Each computed for $\theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$ | |
| Homogeneity | measures smoothnes of texture |
| Energy | measures the total sum-squared energy |
| Entropy | measure randomness of texture distribution |

3.2.3 Features

In Table 3.1, we list the features used to compose of feature space. These features characterize the crowd segments from the previous section based on size and shape as well as texture and entropy. These features can be used to regress onto a count estimate of each segment. This is accomplished by concatenating our features into a vector and modeling the regression using a Gaussian Process (GP). The feature vector characterizes a distribution over functions.

3.2.4 Regression

If we take a look at some of the features such as segment area and shape, we can see how they are good linear indicators of how many people are contained in a crowd. However, how can we account for non-linearities? Computationally, these features can be used to regress linearly and non-linearly onto a count estimate of each segment. For this regression we use the Gaussian Process (GP) Regression and create a GP regression model based on our feature vector. The feature vector is composed of a concatenation of all our features.

A Gaussian Process is a distribution over functions, say $f(x)$, defined by a covariance function, let's call it $k(x, x')$. Taking the inner product of f using the *kernel trick*, we retrieve an expression for k such that we can train our GP model. We select a model that easily takes into account our mostly linear regression (after perspective is accounted for) while taking into account local non-linearities that may be attributed to person occlusion, intra-crowd spacing, and segmentation errors (perhaps due to lighting variation and shadows). Capturing both the linear and non-linear components, we select the following kernel/covariance function with hyperparameters $\alpha = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$:

$$k(x_p, x_q) = \alpha_1(x_p^T x_q + 1) + \alpha_2 \exp \frac{-\|x_p - x_q\|^2}{\alpha_3} + \alpha_4 \delta(p, q)$$

Our first term captures the linear components of our people count model. The second term of the kernel is the RBF component, which captures the local non-linearities. The third term models observation noise. Since this is a special case of Bayesian Inference where our prior is a GP, the hyperparameters may be learned via maximizing the

marginal likelihood of the training data. After training the model, we may obtain count estimates of segments based on their features via our GP regression model.[3] [20]

3.3 Automation Implementation

Now that we have the computational tools for a people count model, we may discuss how to integrate these methods into an automated system. We will discuss the core functional blocks seen in

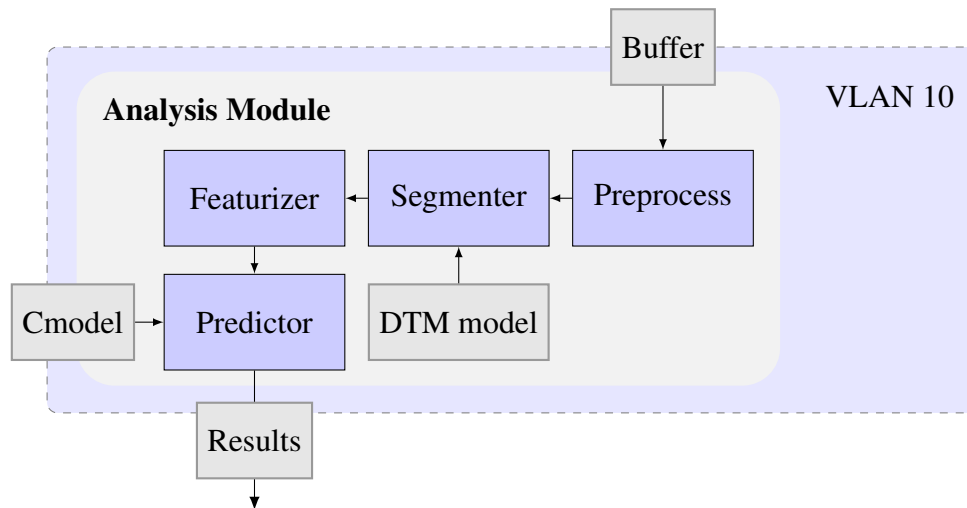


Figure 3.4: Analysis Module Diagram. Depicted above is our Analysis Module. The Analysis Module resides in SVCLs VLAN 10.

3.3.1 Scene Setup

Scene Definition

Since our crowd counting analytic is not viewpoint invariant and counts people without people models, i.e. has no method for contending with camera motion, we must constrain the definition of *scene* to any fixed camera field-of-view. Static cameras support one field of view. However, their high-definition single-field-of-view image may be cropped, yielding many sub-scenes for the crowd counting analytic. Pan-Tilt-Zoom (PTZ) cameras support a larger variety of scenes. Panning a camera a few degrees

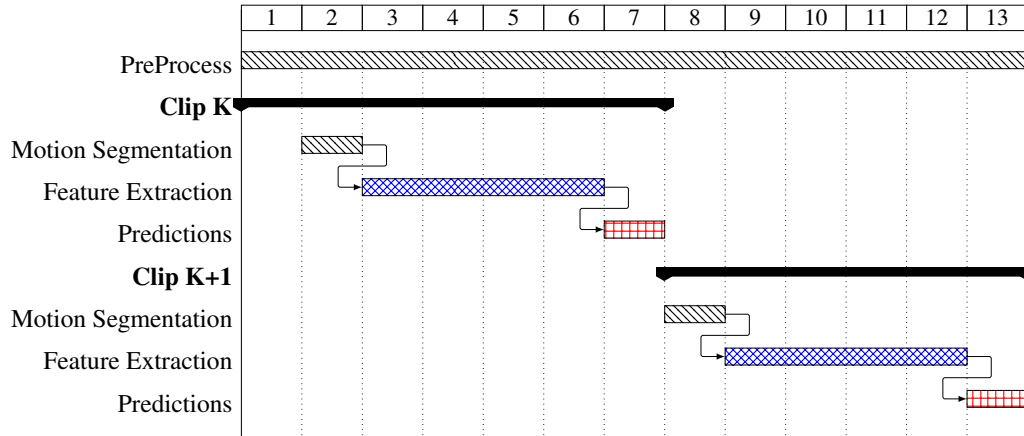


Figure 3.5: Analysis Schedule - The preprocess is a background process. The initial Clip K must wait for the preprocessing of the first clip. However, at the end of processing Clip K, Clip K+1 immediately begins.

may introduce enough change in the holistic motion of classes that which a new DTM model is necessary. Since cameras are shared and may be controlled by the UCSD Police, we must have a method for holding viewpoints constant when streaming data. Conveniently, static cameras inherently possess this functionality and PTZ cameras may be programmed to store necessary settings as *presets*. (see section 2.2.2 for GotoPreset class reference on page 26.)

3.3.2 Count Functions

The main functions our automated counting wrapper integrates, including itself, are listed in Table 3.3.

3.3.3 Experiment Setup

Before we have actually turn on the automated system, our computational tools require some supervised learning for each scene. This has been outlined in [21] and [22] located in the SVCL Matlab code repository. However, for completeness we provide a list of the required files in Table 3.3.

Training the people counting model requires the following files in Table 3.3.

Table 3.2: Core Analysis Automation Functions

| Compiled C Library functions in SVCL CrowdAnalytics repository. | |
|--|--|
| count_scene.c | Multi-threaded People Counting Wrapper Application that integrates all of the following functions and applications. |
| pef2y.c | A modification of the perl script <i>mp2dat</i> . Used to convert videos from <i>pef</i> format (MPEG4-PS codec) to linux <i>dat</i> format. Converts from 3-channel RGB to grayscale. Rescales from 1028x720 to 320x180 and cropped to 320x160 to remove timestamp. |
| Shared C Library functions in SVCL dytex repository. | |
| segm_dytexmix_learn | Used to learn a DTM <i>and</i> segment clip based on model. |
| dytex_mix_save | Used to save a DTM model for usage on future clips. Shared C Library. |
| dytex_mix_load | Used to load a previously learned DTM. |
| segm_dytexmix | Used to segment a clip based on a previously learned DTM. |
| Compiled Matlab functions in Antoni Chan’s <i>peoplecnt matlab</i> library. | |
| peoplecnt_cmd | Computes features (see Table 3.1 on page 40 listed in options structure (see Appendix A.2). And regresses features to count predictions using our trained GP regression model. |

Table 3.3: Required Files for Counting Model Training

| |
|--|
| an options structure (see Appendix A) |
| videos. |
| segmentations of the videos into different crowds (e.g. by direction of motion). |
| a segmentation map, which labels each segment. |
| a region-of-interest (ROI) in the video. |
| the ground-truth counts for each segment in the ROI. |
| a perspective (normalization) map. |
| an evaluation set, which specifies the training and test sets. |

Place all required files in a folder named `sceneX` corresponding to the name of our scene. After we have thoroughly defined our scene parameters, our exporter should be running and placing the desired clips in our shared buffer. Our re-coding application `pef2y.c`, or some modification of it, can be used to produce the desired video clips. If we modify `pef2y.c`, `count_sceneX.c` should inherit these changes before compilation. After training is complete, compile `count_sceneX.c` with the modified `pef2y.c` (replacing `X` with the scene name).

For multiple scenes, there should be multiple compiled versions of the `count_sceneX.c` wrapper application. This application should be compiled using the `imake` method.[23] Once our application has compiled, running it will process scene clips on the fly. The application can run `pef2y.c` as a multithreaded subprocess thus readying a video for count prediction as soon as the previous result estimates are calculated.

As discussed previously, see Figure 3.4, our wrapper: (1) loads a pre-trained DTM model, (2) performs motion segmentation based on DTM model, (3) our pre-trained count model is loaded into the prediction function, (4) features are then fed into our prediction model where count estimates are formed, (5) results are saved a matlab `.m` file, and finally (6) `AppendTable.py` parses and sends the results to our visualization module.

3.4 Known Issues and Notes

1. As mentioned previously, the core agents were developed in matlab, converted to a shared C library, and linked to our main application. While other functions in this relay race take seconds to run, these functions are attributed to over 80% of computational time. It is difficult to say how much time could be saved. However, rewriting these functions in C would make bring the lag down from 10 minutes to, perhaps, 3 or 4 minutes. The incentive to do so will increase as this distributed analytics system is scaled up to handle more scenes.
2. Many fields of view are overlapping in the center of campus where the density of outdoor cameras is greatest. A method for integrating knowledge from each viewpoint, i.e. consolidating predictions, is an interesting next step to take. De-

veloping a viewpoint invariant analytic could perhaps be tethered to efforts of bootstrapping training of models of similar scenes.

3. Overlapping with the previous note: anytime camera maintenance or system updates are performed on the surveillance system, DTM models must be re-trained as we cannot ensure presets and camera orientations are consistent.
4. Many cameras are relatively low to the ground and are sub-optimal for our analytic. This results in complex crowd motion that can be difficult to segment into a couple classes. Therefore, in some cases it may be necessary to train multiple classes and combine their models after training to ensure the desired class segregation is achieved. (look up function)

4 Visualization Methods and Results

The results database is the foundation of our Visualization Module. The database is an SQL like web service database called Google Fusion Tables. Fusion Tables are an accessible storage solution for the geospatial-based results via a subset of SQL commands. These tables allow for quick filtered querying of results and can be integrated with the Google Maps API and Google Charts API.

4.1 Core Python Scripts

The following functions were written in python and intended to be command-line linux functions that also can be, and are, called from our automated counting wrapper. The source code for each function can be found in Appendix ?? on page ??.

4.1.1 ShowTables

Our ShowTables python function can be called by simply typing “python ./ShowTables.py”. This will return a list of Fusion Tables and their unique table id, which are associated with the default Google Account “svclmulloy@gmail.com” that was created explicitly for this project.

4.1.2 CreateTable

When creating a new Fusion Table, we use the CreateTable.py python function. This function accepts a tablename and the directory path of two metadata files. After authenticating, we save the authentication token, table name, and table id as a *pickle file* so that our other functions may quickly access the same table.

```
##### e.g. $ python CreateTable.py -n 'TestTable'
##### -n <tablename>
##### -s <header_names>.csv, default: "header_names.csv"
##### -t <header_types>.csv, default: "header_types.csv"
```

These helper files contain our standard table format and define the column and column type space of our tables. The contents are shown below.

```
1 DATETIME, STRING, LOCATION, NUMBER, NUMBER, NUMBER, NUMBER, NUMBER, NUMBER, \\
2 NUMBER, NUMBER, STRING, STRING, LOCATION
```

```
1 time, locationName, location, count_class1, variance_class1, \\
2 direction_class1, count_class2, variance_class2, direction_class2, \\
3 averageCount, averageVariance, previewURL, Analytic, geometry
```

4.1.3 AppendTable

This function will upload our automated count results from local storage to Fusion Table storage. To do so we need to tell the function: - Which scenes to update. - Some metadata that remains constant measurement to measurement - And where to upload the results

We achieve this, we pass the following arguments:

```
##### e.g. $ python AppendTable.py -n 'TestTable' -s 'scene1.csv'
##### -n 'TableName' - Name of pickle file for existing table.
##### -f 'header_names.csv' - Table Header Names
##### -s 'scene1.csv' - Static Data for each scene
```

TableName is used to retrieve the table authentication token from a pickle file of the same name.

Since each scene has unique metadata (e.g. location, area, preview image), the -s option is used to pass the filename containing this metadata. (e.g. *scene1*, *scene2*, and *scene3*). These names correspond to (pre-written by user) csv files which contain metadata describing visualization related metadata. Below is the csv file structure and

content name.

| | |
|-------------------------|----------------------|
| Segment Name 1 | Segment Name 2 |
| Direction 1 | Direction 2 |
| Scene Prefix | Location Coordinates |
| Scene Preview Image URL | |
| Analytic | |
| Polygon Map | |

For example, here are three csv files with the scene-specific metadata.

scene1:

```

1 scenel_d, scenel_u
2 90, -90
3 scenel, "32.879276 -117.235893"
4 http://www.svcl.ucsd.edu/~mulloy/Preview/01PTZ.jpg
5 CrowdCount
6 "<Polygon><outerBoundaryIs><LinearRing><coordinates>-117.235771, \\
7 32.879353,0 -117.235756,32.879269,0 -117.236084,32.879219,0 \\
8 -117.236107,32.879345,0 -117.235771,32.879353,0</coordinates>\\
9 </LinearRing></outerBoundaryIs></Polygon>"

```

scene2:

```

1 scene2_l, scene2_r
2 135, 315
3 scene2, "32.879033 -117.235231"
4 http://www.svcl.ucsd.edu/~mulloy/Preview/02QN.jpg
5 CrowdCount
6 "<Polygon><outerBoundaryIs><LinearRing><coordinates>-117.235458, \\
7 32.879223,0 -117.23538,32.878948,0 -117.235092,32.879063,0 \\
8 -117.235222,32.87928,0 -117.235458,32.879223,0</coordinates>\\
9 </LinearRing></outerBoundaryIs></Polygon>"

```

and scene3:

```

1 scene3_l, scene3_r
2 0, 180
3 scene3, "32.87974 -117.237554"
4 http://www.svcl.ucsd.edu/~mulloy/Preview/PTZLibraryWalkNorth.jpg

```

```

5 CrowdCount
6 "<Polygon><outerBoundaryIs><LinearRing><coordinates>-117.237625,\\
7 32.879807,0 -117.237633,32.879551,0 -117.23744,32.879551,0 \\
8 -117.23748,32.879841,0 -117.237625,32.879807,0</coordinates>\\
9 </LinearRing></outerBoundaryIs></Polygon>"

```

It is important the Segment Name suffixes (e.g. `_r`, `_l`, `_d`, or `_u`) on the first line end correctly and correspond to the class names during the People Counting System setup in Section 3.3.3 on page 43.

4.1.4 Table2XML

This function extracts the most recent measurements of each scene from our Fusion Table and creates an XML document for the purpose of webpage visualization referencing via javascript.

```

#### e.g. $ python Table2XML.py
#### note: scenes to include in xml are specified in SCENES.csv

```

The following csv file (`SCENES.csv`) lists the scenes whose results we wish to visualize.

```

1 scene1, scene2, scene3

```

4.2 Visual Demos

Using the above python functions, count results are easily queried. The following demos rely on the functionality of the previous section.

4.2.1 Count Vectors

This is the first demonstration Prototype. Each scene is represented by a marker on a Google Map. Projecting out from each marker are two vectors. The vector directions are determined by the class directions while the count of each class are mapped to

their respective vector's magnitude. In the electronic version, you can see the vectors change in magnitude in realtime. Please refer to Figure 4.1.

4.2.2 Realtime Map and Trends

Please refer to Figures 4.2.

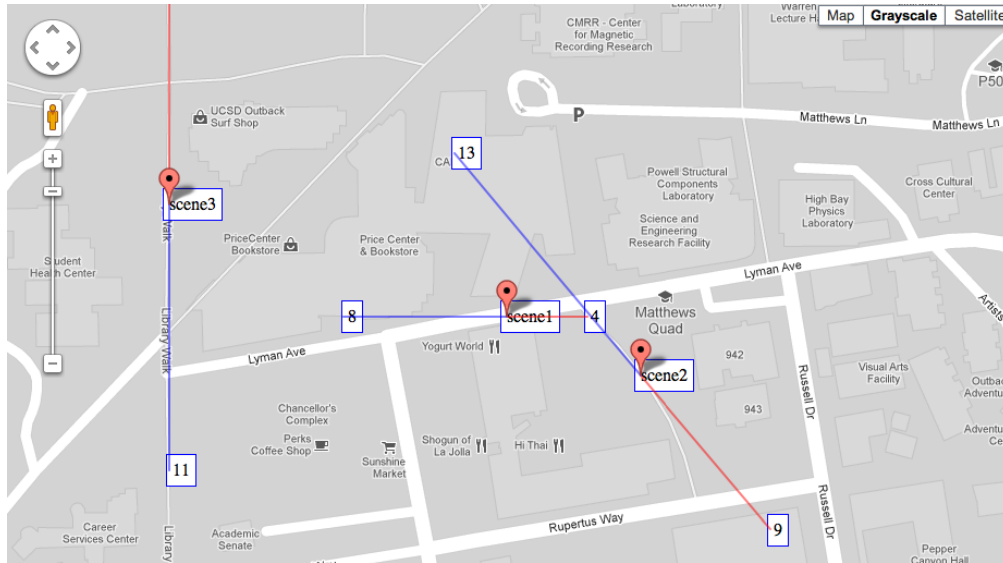
In our second demo prototype, we replace dynamic vectors for a heatmap so that we may have a more intuitive comparison of scenes and perhaps causal relationships. Another contrasting difference seen is the presence of a chart. This chart displays historical trends of specific scenes. Scenes may be selected using a dropdown menu (See Figure 4.2). Also present is an Analytic selector. In the future, a visualization, such as this one, will obtain more than only *count data* from each of these scenes (i.e. anomaly detection, event recognition, or even comparison of counting methods.).

4.2.3 Zoomable Line Charts

Please refer to Figure 4.3.

In Demo 3 we implemented a zoomable line chart using Google's Chart API. These charts were generated using a URL submission query such as the following for *Scene 1*:

```
https://www.google.com/fusiontables/embedviz?
gco_displayAnnotations=true&gco_wmode=opaque&containerId=gviz_canvas
&rmax=250&q=select+col10%2C+col9+from+2416508+where+col6+contains+
'scene1'+and+col10+%3E+'02%2F20%2F2012'&qrs=+and+col10+%3E%3D+
&qre=+and+col10+%3C%3D+&qe=+order+by+col10+asc&viz=Gviz&t=TIMELINE
&width=750&height=300
```



(a) Demo 1 Grayscale Map



(b) Demo 1 Satellite

Figure 4.1: Visual Demo 1 - Count Vectors

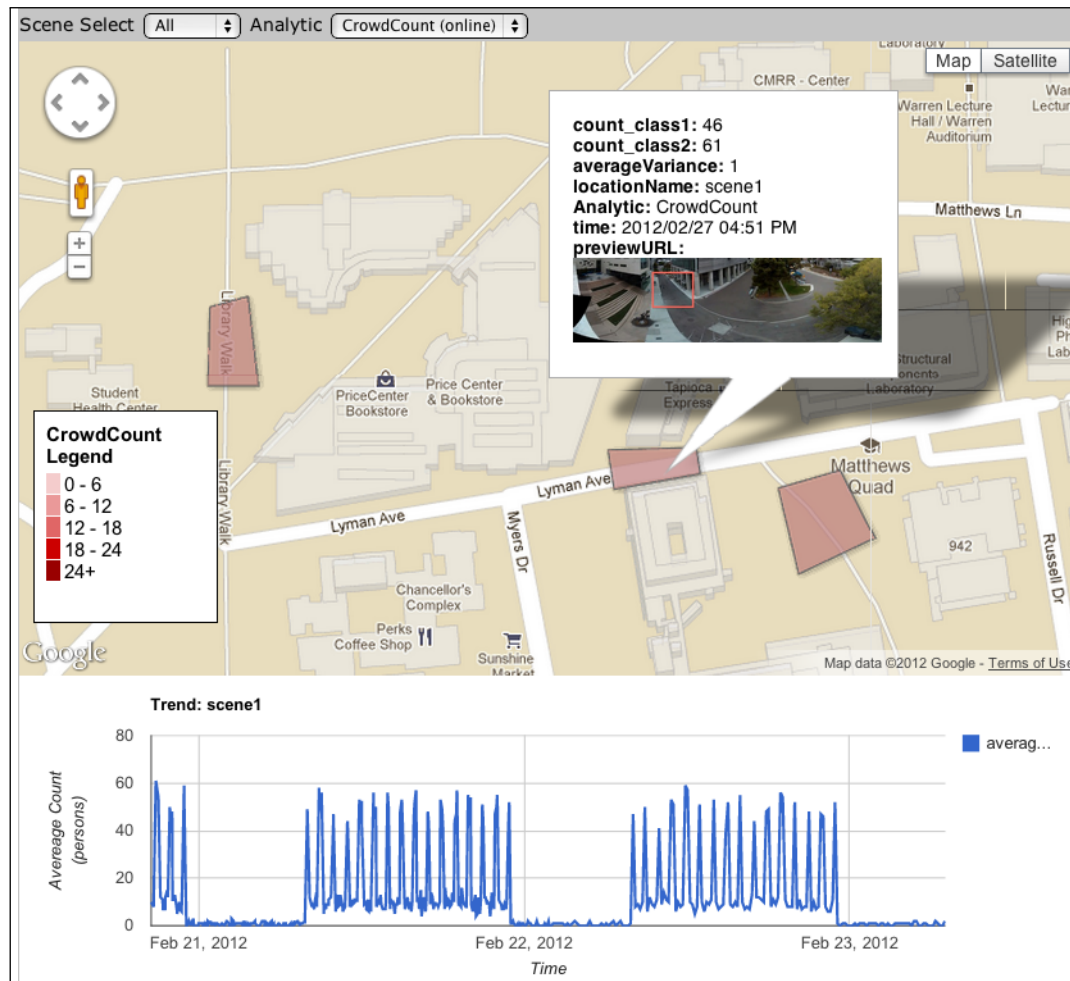
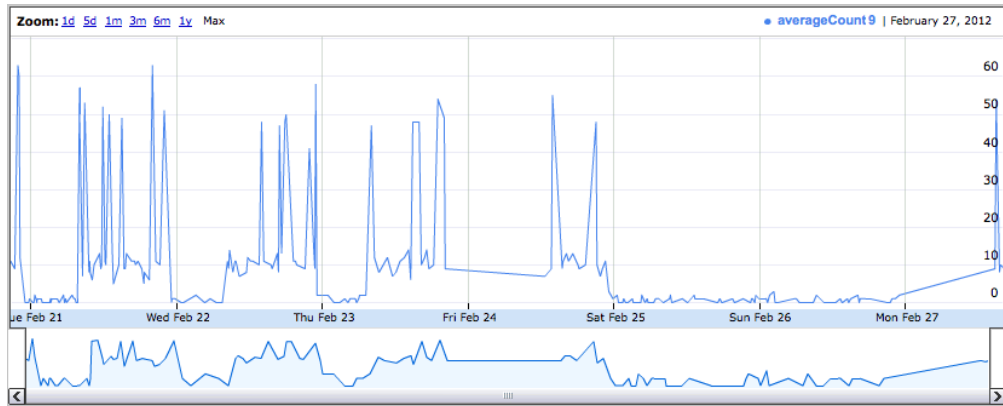
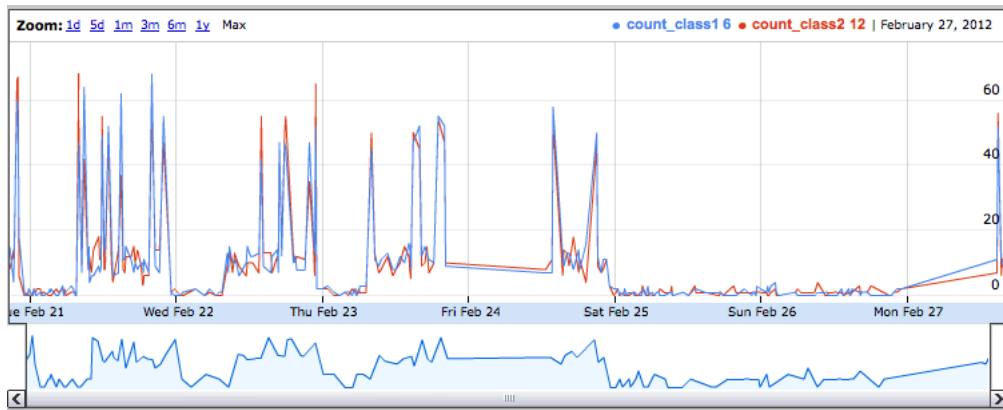


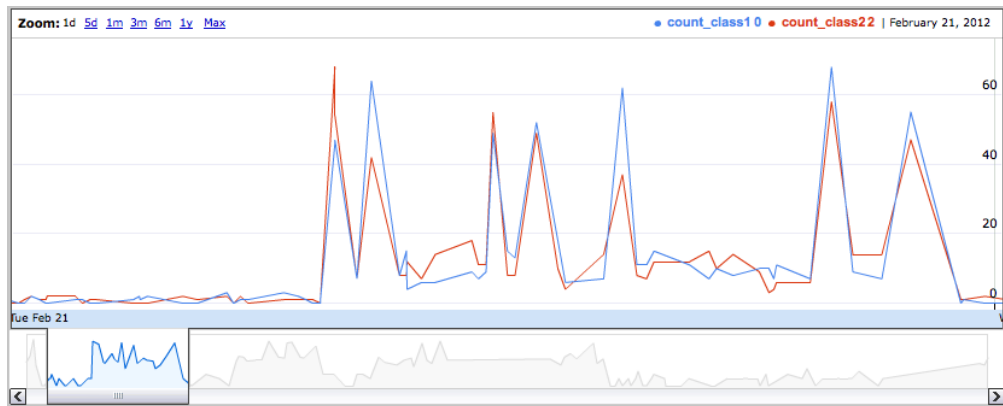
Figure 4.2: Realtime Map and Timeline (All Scenes) with Scene Selection and popup window. *Top*, a realtime heatmap showing the spatial and count relationships of scene areas. Popup window displays more information, such as a preview camera view of scene. *Bottom*, a timeline with average pedestrian count versus time. URL: <http://www.svcl.ucsd.edu/projects/crowdanalytics/demos.html>



(a) 7-day Timeline



(b) 7-day Timeline



(c) 1-day Timeline

Figure 4.3: Visual Demo 3 - Zoomable Timelines

4.2.4 Known Issues

It is possible that at times the police department may need to perform system maintenance or replace cameras. Since DTMs are not viewpoint invariant, any changes to viewpoint presets or camera calibration requires models to be retrained. Furthermore, in compliance with Police department regulations, video is not kept in storage at SVCL but rather discarded after analysis. Therefore, re-training can be a significant cost to operating this system.

Furthermore, Fusion Table queries must be made at a frequency less than 5 times a second, otherwise the server may fail to process queries. Additionally, Fusion Table API limits the rows of Fusion Tables to 500 in the free version. However, in our simulations we have had little trouble exceeding this limit. However, there is a HTTP 500 error that will crash our simulations occasionally. This may be related to this limit. If not, then the Google servers have a significant failure rate. Visualizations could be performed on SVCL servers. However, Google's Fusion Tables and Charts API Teams are very active and provide unmatched functionality for free.

5 Future Work

5.1 Code Optimization

In our Analysis Module, there exist two core functions from the SVCL Matlab repository that are compiled as stand-alone libraries. These functions are the Feature Extractor and the Count Predictor. Although other Matlab functions and GUIs are used in the training of our count and DTM models, these two functions are used in the real-time processing flow and contribute the greatest time cost to our system. Re-writing these two functions so they have no redundant calculations and are optimized for speed would help meet a real-time processing demand with no lag time.

5.2 Other Counting Methods

Many fields of view are limited to open paths where pedestrian trajectories are not confined to a space. In these quite frequently encountered situations, more traditional person-tracking algorithms would be better equipped to count pedestrians that pass, say, an arbitrary line. Furthermore, for the sake of comparison all scenes should utilize multiple counting methods.

Furthermore, we began work on creating a semi-supervised training method for our mixture of dynamic textures. In this method, we employ not one mixture model for all classes. But rather, we train an entire mixture model for each class, where each component models sub-class motion. In this case, motion segmentation can be performed similarly as before. However, rather than assigning pixels to the mixture component of greatest MAP, we assign pixels to a model of greatest Likelihood.

5.3 Suppressing Training Costs

The motion segmentation modeling we employed in this project assumes all classes of motion exist in each frame of our training and test clips. However, this assumption is often not satisfied. Furthermore, there are many scenarios in which the human-defined opposing motions are too similar or even mixed from our motion-model's perspective. For instance, modeling northward and southward pedestrian motions may be difficult with the prevalent motion of eastern/western motions or, more subtly, southwestern/northeastern (and vice versa) motions.

Dr. Chan's current motion segmentation method is insufficient for scenarios in which a class of motion may be composed of multiple or even several sub-class motions; not all of which possess sufficient dissimilarity to discriminate from other sub-classes. For example, suppose our motion classes are (1) north motion and (2) south motion. Sub-classes may include (a) pedestrians, (b) cyclists, (c) runners, (d) golf carts, (e) a skewed direction such as northwestern rather than north.

To overcome these obstacles, we develop an experimental method of training the subclasses independently (assuming their covariance is zero) and perform our class assignment based on maximum posterior probability of the query likelihood on all trained classes. Although this method works under the more frequently satisfied assumption that sub-class motion occurs without the presence of others in the field of view for a significant duration, the implementation of this method is unfinished.

5.4 Suppressing Count Variance

Count estimates are calculated under the assumption that all objects in our field of view are pedestrians. However, anomalies such as bicyclists, golf carts, and pedestrians with large objects do exist. Although these anomalies are sporadic and may be acceptable, other sources of variance are more prevalent. For example, there are many variations in how pedestrians are lighted, giving rise to shadows. Shadows do not affect motion-based segmentation. However, they introduce error into many of our area-based features, which may bias the count predictions upwards. To compensate for this prevalent bias, features may be weighted to suppress this shadow bias. These weights could be

based on a shadow suppression model as in [25]. Alternatively, there exist many statues on campus, whose shadows may support a realtime and more precise shadow suppression model. 3D models could be composed or roughly learned (via holography) for the purpose of inferring lighting and cast shadow area. However, this latter suggestion is only speculative, at present.

5.5 Control Theory

Our system, currently, is a single agent system. If we ignore the LSE in our line plots, the only intelligent layer resides in the Analysis Module's People Counting Model. There are significant research efforts developing an additional agent at the Acquisition level, whose goal it is to improve the allocation of limited resources and utility viewpoints. This can be achieved by adding an intelligent control layer that may be in charge of (1) prioritizing scenes, (2) creating new salient scenes, (3) combining salient information from multiple viewpoints, and/or (4) search and track objects independent of viewpoint and across a distributed camera network.

We direct the interested reader to two Associate Professors: (1) Amit K. Roy-Chowdhury's (UCR, Electrical Engineering) whose projects include an intelligent camera network;[24] (2) Faisal Z. Qureshi (UOIT, Computer Science) whose work includes various methods of Camera Control, Multi-Tasking, and Scheduling. Both gentlemen co-organized the CVPR Workshop on Camera Networks and Wide Area Scene Analysis. (Workshop URL: <http://faculty.uoit.ca/qureshi/conferences/wcnwasa11>)

5.6 Viewpoint Invariance

Our Dynamic Texture approach to motion segmentation is robust to variations in lighting and partial occlusions inherent of crowds. However, the assumption that our viewpoints are constant is a heavy cost to a surveillance system that is shared, runs 24 hours a day 7 days a week and possesses PTZ cameras. System maintenance must be performed. Although this is rare, when it happens this perturbs our viewpoints significantly. Furthermore, keeping PTZ cameras stationary produces an excessive necessity

for more cameras while additionally throwing away their capabilities. For these reasons, viewpoint invariant motion models that cope with minor camera movement and/or interpolate adaptive model parameters for PTZ motion are system features highly valuable to an intelligent camera network. As seen in a controlled and distributed camera network, there is greater utility in PTZ cameras.[24]

References

- [1] Svcl website. [Online]. Available: www.svcl.ucsd.edu
- [2] A. Chan and N. Vasconcelos, "Probabilistic kernels for the classification of autoregressive visual processes," *IEEE CVPR*, 2005.
- [3] A. Chan, Z. S. J. Liang, and N. Vasconcelos, "Privacy preserving crowd monitoring: Counting people without people models or tracking," *IEEE CVPR*, 2008.
- [4] A. Chan and N. Vasconcelos, "Modeling, clustering, and segmenting video with mixtures of dynamic textures," *IEEE TPAMI*, 2008.
- [5] I. C. Society. (2009) Proceedings from the eleventh iee international workshop on performance evaluation of tracking and surveillance. [Online]. Available: http://www.cvg.rdg.ac.uk/PETS2009/PETS09_PROCEEDINGS.pdf
- [6] U. of California Office of the President Department of Information Resources and Communications. Fall 2010: University of california statistical summary of students and staff. [Online]. Available: <http://www.ucop.edu/ucophome/uwnews/stat/statsum/fall2010/statsumm2010.pdf>
- [7] M. Morrow. (2008) Motion database. [Online]. Available: <http://svcl.ucsd.edu/projects/motiondb/index.html>
- [8] J. Ferryman. (2009) Performance evaluation of tracking and surveillance. [Online]. Available: <http://www.cvg.rdg.ac.uk/PETS2009/>
- [9] A. Chan and M. Morrow, "Analysis of crowded scenes using holistic properties," *IEEE PETS*, 2009.
- [10] Google permissions. [Online]. Available: <http://www.google.com/permissions/>
- [11] M. Morrow. Surveillance camera locations. [Online]. Available: <http://g.co/maps/qhhpb>
- [12] Pelco. Pelco developer network. [Online]. Available: pdn.pelco.com

- [13] M. Lindner. libconfig – c/c++ configuration file library. [Online]. Available: <http://www.hyperrealm.com/libconfig/>
- [14] A. Chan. Dynamic texture models. [Online]. Available: <http://www.svcl.ucsd.edu/projects/dytex/>
- [15] G. Doretto, “Dynamic textures,” *International Journal of Computer Vision*, vol. 51.2, p. 91, 2003.
- [16] B. Lingner. Example: Kalman filter system model. [Online]. Available: <http://www.texample.net/tikz/examples/kalman-filter/>
- [17] A. C. Davies, J. H. Yin, and S. A. Velastin, “Crowd monitoring using image processing,” *Electron and Communications Engineering Journal*, 1995.
- [18] H. D. Kong, D. Gray, “Counting pedestrians in crowds using viewpoint invariant training,” *British Machine Vision Conf.*, 2005.
- [19] A. N. Marana, L. F. Costa, R. A. Lotufo, and S. A. Velastin, “Estimating crowd density with minkoski fractal dimension,” *IEEE Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, 1999.
- [20] C. E. Rasmussen and C. K. I. Williams, “Gaussian processes for machine learning,” *MIT Press*, 2006.
- [21] A. Chan. People count library in svcl repository.
- [22] ———. People count tutorial - command-line programs.
- [23] [Online]. Available: <http://www.xfree86.org/current/imate.1.html>
- [24] [Online]. Available: <http://www.ee.ucr.edu/~amitrc/CameraNetworks.php>
- [25] A. Doshi and M. Trivedi, “Satellite imagery based robust, adaptive background models and shadow suppression.” *Signal, Image, and Video Processing Journal*, vol. June, no. 1(2), pp. 119–132, 2007.

A Appendix - People Counting System

A.1 Compiled Matlab Programs

Our functions for setting up the counting system, feature extraction, and count predictions require compilation in before being used in our automated system. Future versions of these functions will be optimized in native C source code. [21]

| | |
|-----------------------------------|--|
| peoplecnt_cmd [mode] [optionfile] | a command-line program that runs the counting system. The command takes two arguments: 1) the operation, 2) the name of the options file. [mode] may be: 'feature', 'train', 'predict', or 'eval'. |
| peoplecntgui | A GUI interface for setting up a counting system. |

A.2 Options File for Training People Counting Model

The following options structure is to be used in conjunction with the Matlab People Counting stand-alone command-line program *peoplecnt_cmd*

| | |
|------------------|--|
| opt.mode | the stage of the system to run: 'feature', 'train', 'predict', or 'eval' |
| opt.basedir | the base directory of the data. All filenames given in the configuration file are relative to this directory. |
| opt.recompute | Set to 1 to force the system to ignore the cache, i.e. recompute everything. e.g. opt.recompute = 0; |
| opt.dirs | Names of the directions of interest. These are one or two-letter values, which should match the segmentation map file and ground-truth file. e.g. opt.dirs = 'r', 'l'; Here the directions are named 'r' and 'l' for "right" and "left". Note that 't' is a special direction which corresponds to all foreground regions (i.e. all directions). |
| opt.update_flags | flags to force an update on each stage: [feature, train, predict, eval]. e.g. opt.update_flags = [0 1 0 0]; will force retraining the system. In general, this should be set to opt.update_flags = [0 0 0 0]; |
| opt.flag_usemat | if 1, assume imgs and segms are .mat files. if 0 (the default), they are dat files. |
| opt.flag_nogz | if 1, do not gzip matlab file outputs. if 0 (the default), gzip output). Set this to 1 on Windows if gzip is not installed. |
| opt.debug | 1 = enable debugging mode (default = 0) |

The following are set automatically in source code: opt.mode, opt.debug=0, opt.flag_usemat=1, and opt.flag_nogz=0. We set opt.flag_usemat equal to zero in the source code prior to compiling so that our videos may be in .dat format rather than in the Matlab .mat format.[22]

Example options file:

```

ProjOptions
1 % Project file automatically generated by % peoplecntgui v1.01
2
3 % base directory for the experiment
4 opt.basedir =
5 '/data4/home_new/mulloy/PeopleCnt/test';
6
7 % set to 1, to force recomputing everything

```

```
8 % (i.e. do not use cached values)
9 opt.recompute = 0;
10
11 % the region of interest file
12 opt.roiname = 'MetaData/vid_roi.mat';
13
14 % the perspective map file
15 opt.pmapname = 'MetaData/vid_dmap.mat';
16
17 % the segmentation map file
18 opt.segmmname = 'MetaData/vid_segmmat.mat';
19
20 % the directions
21 opt.dirs = {'d', 'u'};
22
23 % the video files
24 opt.imgnames{1} = 'Data/vid0002.mat';
25
26 % the segmentation files
27 opt.segmmnames{1} = 'Data/vid0002_segmmat.mat';
28
29 % file prefix for saving features
30 opt.featspref{1} = 'features/vid0001_feat';
31 opt.featsuf = 'feat';
32
33 % the ground truth files
34 opt.truthname{1} = '
35 MetaData/vid0002_count_roi.mat';
36
37 % evaluation set file
38 opt.evalname = 'MetaData/EvalSet1.mat';
39
40 % the index of the evaluation set to use
41 opt.evalnum = 1;
42
43 % file prefix for saving models
44 opt.modelpref = 'models/modell';
45
```

```
46 % file prefix for saving predictions
47 opt.predpref = {};
48 opt.predsuf   = 'pred1';
49
50 % file prefix for saving model evaluation
51 opt.evalpref   = 'results/eval1';
52
53 % the feature set
54 opt.feature{1}.name = 'norm_area';
55 opt.feature{1}.opt = [];
56 opt.feature{1}.sub = [];
57 opt.feature{2}.name = 'norm_perimeter';
58 opt.feature{2}.opt = [];
59 opt.feature{2}.sub = [];
60 opt.feature{3}.name = 'norm_perimeterorient';
61 opt.feature{3}.opt = 6;
62 opt.feature{3}.sub = [];
63 opt.feature{4}.name = 'norm_paratio';
64 opt.feature{4}.opt = [];
65 opt.feature{4}.sub = [];
66 opt.feature{5}.name = 'norm_edge';
67 opt.feature{5}.opt = [];
68 opt.feature{5}.sub = [];
69 opt.feature{6}.name = 'norm_edgeorient';
70 opt.feature{6}.opt = 6;
71 opt.feature{6}.sub = [];
72 opt.feature{7}.name = 'norm_edgeminkowski';
73 opt.feature{7}.opt = [];
74 opt.feature{7}.sub = [];
75 opt.feature{8}.name = 'norm_glcm';
76 opt.feature{8}.opt = 'f';
77 opt.feature{8}.sub = [3 4 5 8 9 10 13 14 15 18 19 20];
78
79 opt.model.name           = 'gp';
80 opt.model.namesuf       = 'gp-lin-rbf';
81 opt.model.norm          = 'X';
82 opt.model.param.covfunc =
83 {'covSum', {'covSEiso_fast', 'covLINone', 'covNoise'}};
```

```
84 opt.model.param.normmode = 'y';  
85 opt.model.param.numtrials = 1;
```