Improved Robustness and Efficiency for Automatic Visual Site Monitoring

by

Gerald Edwin Dalley

B.S. Electrical and Computer Engineering,
The Ohio State University (2000)
M.S. Electrical Engineering,
The Ohio State University (2002)

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author ....
Department of Electrical Engineering and Computer Science
June 27, 2009

Certified by...
W. Eric L. Grimson
Bernard Gordon Professor of Medical Engineering
Thesis Supervisor

Accepted by ....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

Improved Robustness and Efficiency for Automatic Visual Site Monitoring

by

Gerald Edwin Dalley

Submitted to the Department of Electrical Engineering and Computer Science on June 27, 2009, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Abstract

Knowing who people are, where they are, what they are doing, and how they interact with other people and things is valuable from commercial, security, and space utilization perspectives. Video sensors backed by computer vision algorithms are a natural way to gather this data.

Unfortunately, key technical issues persist in extracting features and models that are simultaneously efficient to compute and robust to issues such as adverse lighting conditions, distracting background motions, appearance changes over time, and occlusions. In this thesis, we present a set of techniques and model enhancements to better handle these problems, focusing on contributions in four areas.

First, we improve background subtraction so it can better handle temporally irregular dynamic textures. This allows us to achieve a 5.5% drop in false positive rate on the Wallflower waving trees video.

Secondly, we adapt the Dalal and Triggs Histogram of Oriented Gradients pedestrian detector to work on large-scale scenes with dense crowds and harsh lighting conditions: challenges which prevent us from easily using a background subtraction solution. These scenes contain hundreds of simultaneously visible people. To make using the algorithm computationally feasible, we have produced a novel implementation that runs on commodity graphics hardware and is up to 76$\times$ faster than our CPU-only implementation. We demonstrate the utility of this detector by modeling scene-level activities with a Hierarchical Dirichlet Process.

Third, we show how one can improve the quality of pedestrian silhouettes for recognizing individual people. We combine general appearance information from a large population of pedestrians with semi-periodic shape information from individual silhouette sequences.

Finally, we show how one can combine a variety of detection and tracking techniques to robustly handle a variety of event detection scenarios such as theft and left-luggage detection. We present the only complete set of results on a standardized collection of very challenging videos.

Thesis Supervisor: W. Eric L. Grimson
Title: Bernard Gordon Professor of Medical Engineering

Acknowledgments

I would like to share my gratitude to all those who have contributed to the research encapsulated in this thesis.

Eric Grimson has been my advisor while at MIT, providing feedback on research as well as readily granting insight on topics ranging from educational methodology to career paths to public policy. He also provided me with the opportunity to teach the venerable MIT 6.001 introductory computer science course, once as a teaching assistant and once as an instructor.

I thank Bill Freeman and Trevor Darrell for being on my thesis committee as readers. Bill has also acted as my academic advisor and as a co-mentor during an internship at Mitsubishi Electric Research Labs (MERL). Each of them has helped complement what I have learned from Eric and other professors through courses they have taught and interactions we've had in coordinating talks for a departmental colloquium.

Complementing the academic work I have done at MIT were a collection of internships graciously provided by Microsoft, MERL, BAE Systems, and D.E. Shaw. These internships have provided opportunities to work on complementary research problems, to experience new application areas, and to see real-world applications of my algorithms and analysis.

I would like to thank those organizations that helped fund this research, including the following sources.

- MIT Presidential Graduate Fellowship
- Defense Advanced Research Projects Agency (DARPA)
- Singapore National Government

Several fellow student collaborators, listed alphabetically, are as follows.

- Biswajit Bose has been a great friend throughout graduate school, providing alternative perspectives and expertise on many problems.

- Lily Lee provided the opportunity for my first collaboration as a Ph.D. student.
- Josh Migdal and I enjoyed energetic discussions exploring the software engineering side of research. I learned from his great coding and design skills while together on joint codebases.
- Kinh Tieu's rational mind whose minimalism produces elegance serves as a great example.
- Xiaogang Wang and I shared endless discussions on activity modeling and HDPs along with a thrilling month working jointly on the PETS 2007 dataset [23].

I also appreciate the contributions of Jim Sukha, Krista Ehinger, and Geza Kovacs as we learned how to make a more efficient pedestrian detector.

There are too many other friends and fellow students here at MIT to list individually who have contributed to making my time at MIT a rich experience.

Finally, I would like to acknowledge the contribution of my family. My parents raised me to approach life with curiosity, logic, and determination. Without that upbringing, I could not have completed this work. I thank my wife for her support, selflessness, and alternative perspectives.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Knowing who people are, where they are, what they are doing, and how they interact with other people and things is valuable from commercial, security, and space utilization perspectives. In commercial settings, retail outlets are interested in monitoring shopper traffic patterns to understand the effectiveness of in-store marketing campaigns and aisle layout choices. Where are people going? Are they stopping at the sales booth? What are the traffic bottlenecks? How can we better prevent revenue loss due to customer and employee theft [32]? These are all questions the owners need answered. Police and security officials are increasingly faced with threats that can be mitigated through surveillance of key physical assets such as seaports, airports, transportation hubs, and government buildings. By automating portions of a surveillance network, users may broaden the range of events and activities they can track and detect.

In setting up a data-gathering network for site monitoring an attractive sensing modality is video. Video cameras are small, safe, inexpensive, and can be used for many different tasks, unlike specialized sensors such as pressure plates or laser tripwires. Users can directly view and interpret the raw data, or with appropriate algorithms, monitoring can be done automatically by computers. An automatic system does not experience fatigue like human operators do, and it can be deployed in settings that would be too dangerous for people.

In order to maximize the value of a site monitoring system, many technological pieces must be integrated. The combined system must achieve acceptably good performance so that the value it provides is not overly diminished by the need to have humans correct its errors. It must solve a large enough part of the user's problems well enough that its cost and complexity are justified.

An ideal automatic site monitoring system should be able to detect all people and meaningful objects in the monitored area, track them over time, and infer all the relationships between them. It should be able to associate observations of an individual from videos taken days, months, or years apart. One should be able to easily detect individual activities like running, excessive loitering, or entering unauthorized regions. It should also be able to detect activities involving multiple actors and/or objects such as theft, violence, surreptitious coordination, or chasing. Further, it should be able to characterize and detect larger scale events like crowd formation, a panic, or shifting traffic patterns.

A system should be adaptable to situations in which there are dense camera networks, sparse networks, active cameras, infrared or hyperspectral videos, and when combined with other sensor modalities such as audio or tripwires.

In handling these tasks, the system must be able to clearly communicate the results back to the user in a timely fashion. For systems used in forensic analysis, it is desirable to have a flexible query system that can easily and intuitively help the user sift through the data with minimal training, either for the human or algorithm. For realtime systems, they must stream results out continuously and in a timely manner, ruling out typical batch algorithms.

Finally, the ideal system would be cheap, robust, and fully automatic. It would require no training, and it would adjust itself automatically to changing conditions in the world such as weather, lighting, and camera placement. When installed, it would automatically calibrate itself. It would make no mistakes, preserve privacy, and cost $1.

Unfortunately, there are significant gaps between current vision capabilities and these ambitious long-term goals. In this thesis, we will present advancements in key computer vision technologies and demonstrate how they help us narrow gaps in accuracy, cost-effectiveness, and/or adaptivity for four application areas. Our work primarily focuses on developing more robust and efficient low-level techniques that enable more effective visual surveillance systems.

1.2 Traditional Visual Surveillance Systems

Many traditional visual surveillance systems [13, 33, 100] utilize a processing pipeline consisting of the following stages:

1. *Data acquisition:* Video data is acquired from one or more cameras and transferred to a computational device.

2. *Detection:* From each video frame, objects of interest are extracted.

3. *Tracking:* Associations between objects in one video frame are made to objects in other frames, forming "tracks." When applicable, tracks from different cameras are also associated with each other.

4. *High-level Analysis:* Using trajectory and/or appearance information, tracks are analyzed to look for motion patterns or specific objects of interest.

In this section, we provide an overview of common existing methods for addressing steps 2–4. With literally thousands of papers written on different portions of this pipeline spanning decades of research, we will focus on those methods that provide the context for our contributions, which we will highlight in §1.6.

1.3 Detection

After acquiring a stream of video data, it is common to use low-level computer vision techniques to find the objects of interest in each frame.

One can model just the background (§1.3.1). Any pixels with unexpected colors are assumed to correspond to foreground objects. This is a computationally-efficient and generic technique that requires little or no training to be able to detect any type of moving object, but it has shortcomings under adverse lighting conditions, with non-stationary backgrounds, and in crowded scenes.

Efforts have been made to retain the advantages of a weak model (robustness and generality) while explicitly modeling the shape and appearance of both the foreground and background as independent 2D layers (§1.3.2).

The other extreme from background modeling uses a strong model for every class of object one wishes to detect (§1.3.3). These models typically have better error rates than a pure background subtraction approach but require much more computation, require substantial training effort, and have trouble scaling well to large numbers of object classes.

Below we outline each of these approaches and touch on some of the contributions we have made in background modeling and strong object models.

1.3.1 Adaptive Background Subtraction

When the video camera is stationary, a typical modeling assumption is that the visual world consists of (a) non-moving objects like roads, trees, signs, buildings, and furniture and (b) moving objects like people, cars, boats, or animals. In site monitoring applications, we typically care most about tracking the moving objects. Knowing whether a given pixel is observing an object of the background class (a) versus the foreground class (b) assists us.

We note that a camera pixel that is observing a non-moving object will tend to see similar color values frame after frame: it's looking at the same part of the real world. On the other hand, when a moving object travels through the scene and blocks the light coming from the background object, the pixel's observed color value changes to that of the moving object.

If we know the per-pixel color distribution for the static objects of the scene, we can estimate the posterior probability that the color value seen at a particular pixel location in a particular frame was the result of no moving object being present. Using Bayes' Rule [37], we can formally evaluate this posterior:

$$p\big(L = 0 | C = \boldsymbol{c}^{(t)}\big) = \frac{p\big(C = \boldsymbol{c}^{(t)} | L = 0\big)\, p(L = 0)}{p(C = \boldsymbol{c}^{(t)} | L = 0) + p(C = \boldsymbol{c}^{(t)} | L = 1)} \tag{1.1}$$

$$\propto p\big(C = \boldsymbol{c}^{(t)} | L = 0\big)\, p(L = 0) \tag{1.2}$$

where $C = \boldsymbol{c}^{(t)}$ is the observed pixel color at time t, $L = 0$ means the color was generated by the non-moving background, and $L = 1$ means it was generated by a moving object.

An important question is how we estimate the background color likelihood distribution, $p\big(C = \boldsymbol{c}^{(t)} | L = 0\big)$. If we could tell the system when there are no moving objects present, we could build a statistical model of observed colors using whatever estimation technique was most desirable. Unfortunately, several practical matters complicate this scenario, such as:

- *Labeling Expense:* A user must manually identify time periods when there are no moving objects.

- *Labeling Difficulty:* In scenes with busy traffic and wide fields of view, there often do not exist any camera frames with no foreground objects present. A solution could be to label, for every single pixel, when a collection of video frames contains no foreground objects. Unfortunately, this would be even more expensive and time-consuming than per-frame labels.

- *Environment Changes:* The observed color distribution tends to change over time due to changes in lighting conditions, so the likelihood actually needs to be frequently updated. Training based on manually labeled pixels taken from a single time slice is insufficient.

- *Camera Drift:* Except in the most tightly controlled circumstances, camera mounts tend to drift slightly over time. Even when this motion is small, the effects are noticeable when considering what part of the outside world a single camera pixel is observing.

For these reasons, it is beneficial to have an automatically learned and adapting model of background color distribution.

Arguably the most commonly-used model is the Mixture of Gaussians (MoG) proposed by Stauffer and Grimson [101]. This model makes a few fundamental assumptions about every pixel location in the image:

- *Modality:* the colors produced by the static portions of the scene are drawn from a small number of Gaussian-distributed modes,
- *Independent pixels:* the color of each pixel is independent of all others,
- *Quasi-stationarity:* the Gaussian modes change slowly over time (*e.g.* due to environment changes and camera drift), and
- *Foreground rarity:* moving objects appear infrequently.

The Model: Modality and Independent Pixels

The appearance model for a pixel at location p is defined as a mixture of Gaussian modes,

$$p\big(C_p = \boldsymbol{c}_p^{(t)} | L_p = 0\big) = \sum_{k=1}^{K} \omega_{pk}^{(t)} \mathcal{N}\left(\boldsymbol{c}_p^{(t)}; \boldsymbol{\mu}_{pk}^{(t)}, \Sigma_{pk}^{(t)}\right), \tag{1.3}$$

where K is the number of Gaussian mixture components, $\omega_k^{(t)}$, $\boldsymbol{\mu}_k^{(t)}$, and $\Sigma_k^{(t)}$ are the respective mixing weight, mean, and color covariance of pixel p's component k at time t, and $\mathcal{N}(\cdot;\cdot,\cdot)$ is the N-dimensional multivariate normal distribution,

$$\mathcal{N}(\boldsymbol{X}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{N/2} \left\|\Sigma\right\|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{X} - \boldsymbol{\mu})^\top \Sigma^{-1} (\boldsymbol{X} - \boldsymbol{\mu})\right). \tag{1.4}$$

Because of the independent pixels assumption, we can abbreviate the notation in

Eqn. 1.3 as

$$p\big(C = \boldsymbol{c}^{(t)} | L = 0\big) = \sum_{k=1}^{K} \omega_k^{(t)} \mathcal{N}\Big(\boldsymbol{c}^{(t)}; \boldsymbol{\mu}_k^{(t)}, \Sigma_k^{(t)}\Big), \tag{1.5}$$

with the understanding that there is an implicit p subscript where applicable.

Updates: Quasi-stationarity

When a new pixel is observed, the MoG algorithm finds which Gaussian mixture component, $\hat{k}^{(t)}$, was most likely to have generated it. If none are likely, *i.e.* if $-\log \mathcal{N}\Big(\boldsymbol{c}^{(t)}; \boldsymbol{\mu}_k^{(t-1)}, \Sigma_k^{(t-1)}\Big) > \tau_{match}$ for all k and for some threshold τ_{match}, we create a new mixture component.

Assuming an existing component was matched, we downweight past observations and update the model with the new data:

$$\omega_k^{(t)} \leftarrow \begin{cases} (1-\alpha)\omega_k^{(t-1)} + \alpha & \text{if } k = \hat{k}^{(t)}, \\ \omega_k^{(t-1)} & \text{otherwise,} \end{cases} \tag{1.6}$$

$$\boldsymbol{\mu}_k^{(t)} \leftarrow \begin{cases} (1-\rho)\boldsymbol{\mu}_k^{(t-1)} + \rho\boldsymbol{c}^{(t)} & \text{if } k = \hat{k}^{(t)}, \\ \boldsymbol{\mu}_k^{(t-1)} & \text{otherwise,} \end{cases} \tag{1.7}$$

$$\Sigma_k^{(t)} \leftarrow \begin{cases} (1-\rho)\Sigma_k^{(t-1)} + \rho(\boldsymbol{c}^{(t)} - \boldsymbol{\mu}_k^{(t-1)})^\top(\boldsymbol{c}^{(t)} - \boldsymbol{\mu}_k^{(t-1)}) & \text{if } k = \hat{k}^{(t)}, \\ \Sigma_k^{(t-1)} & \text{otherwise,} \end{cases} \tag{1.8}$$

where α is the weight learning rate and ρ is the color learning rate. α and ρ are user-tunable exponential-forgetting parameters and may take on values within the $(0, 1)$ range. The color learning rate ρ should be chosen based on how quickly one expects environmental changes to affect the color distributions. The weight learning rate α controls how quickly a newly-observed color mode is considered to be part of the background distribution. Larger values encourage faster adaptation to new data.

We note that it is common practice to normalize the weights of all mixture components at each time step after applying the updates of Eqn. 1.6. In this thesis, the equations will be agnostic to normalization. In practice, normalizing the weights when their sum exceeds 1 is wise for reasons of numerical stability.

Background versus Foreground Modes: Rarity

Over time, Gaussian mixture components that are commonly matched will acquire large relative mixing weights, ω. Under the assumption that moving objects are rare, the first $b_p^{(t)}$ components are interpreted as belonging to the static background objects, where

$$b_p^{(t)} = \operatorname*{arg\,min}_{b' \in (1,\ldots,K)} \left(\sum_{k=1}^{b'} \omega_{pk}^{(t)} > \tau_{bgfrac} \sum_{k'=1}^{K} \omega_{pk'}^{(t)} \right), \tag{1.9}$$

and τ_{bgfrac} is the fraction of the mixing weight assumed to belong to static objects. An observed color that is unlikely to be emitted from any of the top $b_p^{(t)}$ components will tend to be labeled as foreground. $0 \leq \tau_{bgfrac} \leq 1$ can be interpreted as the expected frequency of observing the background.

Background/Foreground Labeling

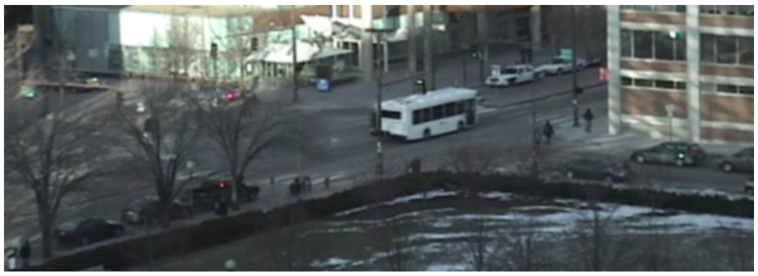
We have just explained how the MoG model is represented and learned. In background subtraction applications, its purpose is to produce a per-pixel map of values representing the likelihood that the observed colors were generated by the static parts of the scene. As we shall briefly see, this likelihood will be used to classify each pixel as background (if its color was most likely caused by static objects in the scene) or foreground (if moving objects most likely generated the color).

Instead of explicitly computing the full posterior of Eqn. 1.2 for each pixel, it is often convenient to use an approximation of the negative log likelihood (Eqn. 1.3),

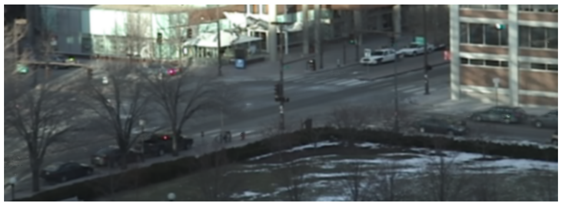
$$\mathrm{d}^2(\boldsymbol{c}; \boldsymbol{\mu}_k, \Sigma_k) = (\boldsymbol{c} - \boldsymbol{\mu}_k)^\top {\Sigma_k}^{-1} (\boldsymbol{c} - \boldsymbol{\mu}_k). \tag{1.10}$$

This approximation is called the squared Mahalanobis distance and it is used because

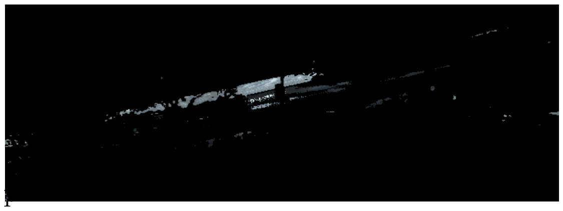
- *Speed:* it avoids evaluation of the computationally-expensive transcendental



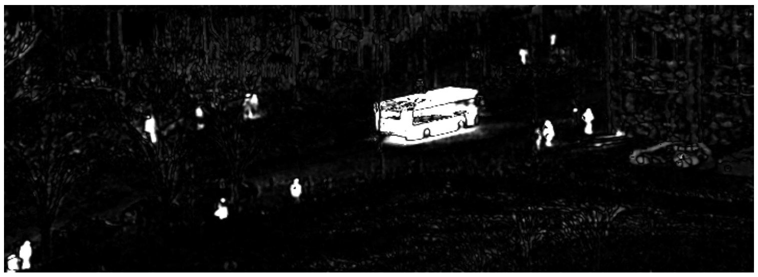
(a) input frame ($\boldsymbol{c}$ for each pixel)



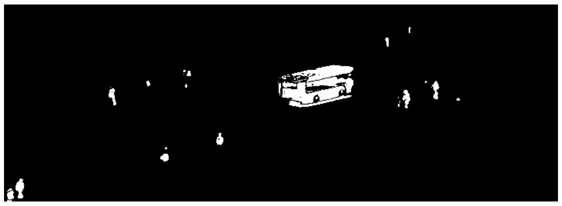
(b) first mixture component (μ_1 for each pixel)



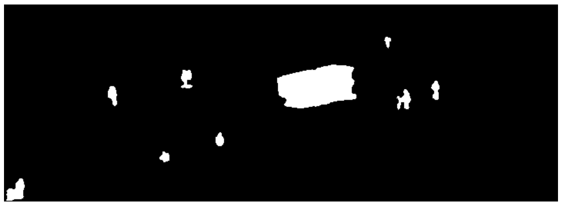
(c) μ_2 for each pixel



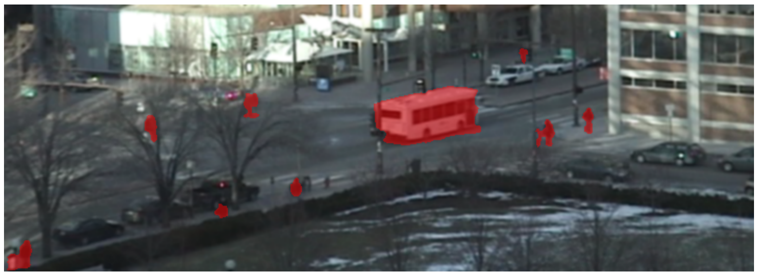
(d) best squared Mahalanobis distance for each pixel



(e) thresholded foreground classification



(f) MRF foreground classification



(g) overlaid MRF foreground

Figure 1-1: *Background Subtraction Illustration:* In (b) and (c) we show the mean color values for the top two Gaussian mixture components at each pixel. For this video clip, most pixels are well modeled with a single mixture component. In cases where no second component exists, a black pixel is used. See the text for additional explanation.

exponent function of Eqn. 1.4,

- *Implicit regularization:* it implicitly acts as a regularization against large covariance matrices[1], and
- *Convenience:* later processing steps can be conveniently cast in terms of negative log likelihoods.

In Fig. 1-1, we illustrate the background subtraction process. A Mixture of Gaussians (MoG) model (Fig. 1-1(b) and Fig. 1-1(c)) is responsible for taking an input image (Fig. 1-1(a)) and producing a per-pixel map communicating the foreground likelihood (Fig. 1-1(e)). A foreground/background classifier such as simple thresholding can be used to produce a map of foreground/background labels, $\{l_p\}_p$, independently at each pixel,

$$l_p = \begin{cases} 0 & \text{if } \left(\min_{k \in (1,\ldots,b_p)} \mathrm{d}^2(\boldsymbol{c}_p; \boldsymbol{\mu}_{pk}, \Sigma_{pk}) \;\; \leq \;\; \tau_{match} \right) \\ 1 & \text{otherwise,} \end{cases} \tag{1.11}$$

for some threshold τ_{match}.

In Fig. 1-1(e) we see that the foreground/background classification results are good, but not perfect. Parts of the bus passing through the intersection look enough like the road that they are improperly classified as background. Although most of the pedestrians have some pixels detected as foreground, they are often broken up into multiple blobs. In a few other places, most notably just to the right of the rightmost pedestrian, there are some isolated false positive foreground detections.

These errors can be addressed if we provide a mechanism to encourage spatial smoothness of the label field. Because moving objects in these scenes are spatially

[1]Consider an observed pixel value that is equally likely under two mixture components (*i.e.* $\mathcal{N}(\boldsymbol{c}; \boldsymbol{\mu}_1, \Sigma_1) = \mathcal{N}(\boldsymbol{c}; \boldsymbol{\mu}_2, \Sigma_2)$). Combining Eqn. 1.4 and Eqn. 1.10, we see that $\mathrm{d}^2(\boldsymbol{c}; \boldsymbol{\mu}_1, \Sigma_1) - \frac{1}{2}\log\|\Sigma_1\| = \mathrm{d}^2(\boldsymbol{c}; \boldsymbol{\mu}_2, \Sigma_2) - \frac{1}{2}\log\|\Sigma_2\|$. Suppose that $\|\Sigma_1\| \gg \|\Sigma_2\|$. Then $\mathrm{d}^2(\boldsymbol{c}; \boldsymbol{\mu}_1, \Sigma_1) > \mathrm{d}^2(\boldsymbol{c}; \boldsymbol{\mu}_2, \Sigma_2)$ and the Mahalanobis distance for component 1 is penalized because it has such a broad and non-discriminating covariance matrix. This is a desirable regularization property because the outlined online learning process can easily fall into a trap where a few outliers match one Gaussian best, expanding its covariance matrix so that it becomes more likely to appear to be a good match for an arbitrary pixel value. By penalizing large covariance matrices, we favor mixture components that compactly represent a cluster of color values.

coherent and are much larger than a single pixel, if one pixel was generated by a moving object, its neighbors should probably be labeled as foreground too. Furthermore, there is temporal coherence in video. If a moving object generated a pixel's color in one frame, part of that object will probably generate the pixel's color in the next frame as well. Similarly, if one pixel is labeled as background, its temporal and spatial neighbors are probably also background.

These time- and space-biases can be formalized in a model called a Markov Random Field (MRF). One good MRF formulation for background subtraction, proposed by Migdal and Grimson [70, 71], minimizes the following objective function:

$$E(\boldsymbol{l}) = \sum_{\{p,q\}\in\mathbb{N}} \mathrm{V}_{p,q}(l_p, l_q) + \sum_{p\in\mathbb{P}} \mathrm{T}_p(l_p) + \sum_{p\in\mathbb{P}} \mathrm{D}_p(l_p) \tag{1.12}$$

where $\boldsymbol{l} = (l_1, ..., l_{\|\mathbb{P}\|})$ is the field of foreground-background labels, $\mathbb{P}$ is the set of pixel sites, $\mathbb{N}$ is the 8-neighborhood graph, $\mathrm{V}_{p,q}(l_p, l_q)$ is a graph edge weight that encourages spatially adjacent pixels to have the same label, $\mathrm{T}_p(l_p)$ encourages pixels to have the same foreground/background label that they had in the previous frame, and $\mathrm{D}_p(l_p)$ encourages pixels to be labeled as foreground when they do not match the background model well. These energy terms have the form

$$\mathrm{V}_{p,q}(l_p, l_q) = t_N \delta(l_p, l_q) \tag{1.13}$$

$$\mathrm{T}_p(l_p) = t_T \delta\big(l_p, l'_p\big) \tag{1.14}$$

$$\mathrm{D}_p(l_p) = \begin{cases} t_F, & \text{if } \Big(l_p = 1\Big) \vee \Big(t_F < \min\limits_{k=1}^{b_p} \mathrm{d}^2(\boldsymbol{c}_p; \boldsymbol{\mu}_{pk}, \Sigma_{pk})\Big); \\ \min\limits_{k=1}^{b_p} \mathrm{d}^2(\boldsymbol{c}_p; \boldsymbol{\mu}_{pk}, \Sigma_{pk}) & \text{otherwise,} \end{cases} \tag{1.15}$$

where $\delta(\cdot,\cdot)$ is the Kronecker delta function, t_N is the user-selected spatial mismatch potential, t_T is the user-selected temporal mismatch potential, l'_p is the label assigned to pixel p in the previous frame, and t_F is the user-selected foreground label potential[2].

[2]If $t_N = t_T = 0$, then the MRF becomes degenerate and does independent thresholding on each pixel. In that case, t_F in Eqn. 1.15 is equivalent to τ_{match} in Eqn. 1.11.

The MRF energy function in Eqn. 1.12 has the form of a 2-label Potts model[3]. We can approximately find the label set $\boldsymbol{l}$ that minimizes it through a technique called Gibbs sampling [70] in which one iteratively samples each label in turn, given the labels of its neighbors. Alternatively, Greig *et al.* [43] showed that it can be solved exactly in polynomial time using a max-flow/min-cut algorithm [16]. Expanding on that work, Boykov *et al.* [10, 9] showed that minimizing the N-label Potts model is NP-hard but they provide a collection of approximate techniques that work by iteratively solving related 2-label problems exactly.

Having experimented with both Gibbs sampling and min-cut algorithms, we have found that for the background/foreground segmentation problem, an optimized Gibbs sampler is easier to implement, especially when one wants to take advantage of multicore processor architectures. For multithreaded implementations, one can simply partition the set of pixels and have each execution thread be responsible for resampling labels for one partition. We have found that boundary effects along the partition edges are negligible even when doing in-place updates. The Gibbs sampler typically converges within a few iterations (*i.e.* often much fewer than 5–10 iterations) and its solutions are nearly identical to those of an exact min-cut algorithm. Recent work by Anderson and Setubal [1], Wilson and Boykov [26], and Vineet and Narayanan [109] on parallel min-cut algorithms is promising and may be the better choice in the future as greater parallelism becomes available.

In Fig. 1-1(f), we see the results of applying this MRF with typical parameter choices and some minor post-processing[4]. In Fig. 1-1(g) we show the MRF results overlaid on the original image.

After labeling all pixels as foreground or background, foreground blobs can be extracted using connected components analysis. An efficient $O(\|\mathbb{P}\|)$ algorithm is well-known (*i.e.* Shapiro and Stockman provide pseudo-code in their textbook [92](pages 69–73)). Each distinct blob is considered to be a single moving object, ending the

[3]If one simplifies this MRF such that $\mathbb{N}$ is a 4-neighborhood graph, then the Onsager solution to the 2 dimensional Ising model can be used to find an exact solution [77].

[4]We have removed all blobs with fewer than 50 pixels. If we had done this same operation with the thresholded results seen in Fig. 1-1(e), we would have lost important parts of the bus and many of the pedestrians.

object detection stage in a typical background subtraction pipeline.

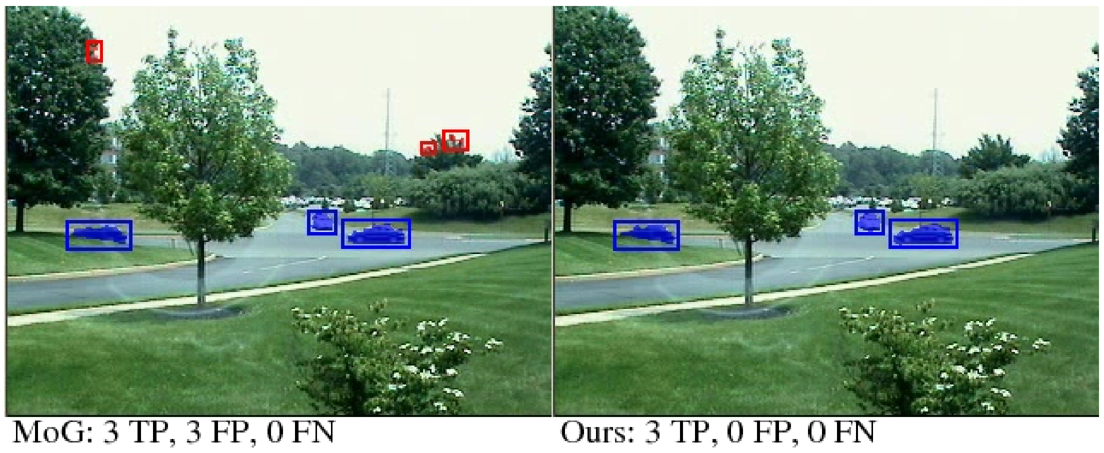
Alternatives and Some Extensions

The pipeline just described uses the modeling techniques of Stauffer and Grimson [101] and the foreground/background classification of Migdal and Grimson [71]. Contemporaneous to Stauffer's work, Toyama *et al.* [107] showed that in real-world scenarios, object detectors that use background subtraction are well suited to detecting times when the model breaks down. For example, when lights are turned on or off, the assumption of gradual illumination changes is violated. Upon detecting this model violation, the system can reset itself and discount the old color models in favor of new data. Others such as Wang and Suter [112, 113] have presented a number of additional heuristic modifications to the procedure. Most notably, they use more robust color spaces. To deal with dynamic textures like rippling water or waving trees, some such as Mittal and Paragios [72] and Sheikh and Shah [93] use fixed-window Gaussian kernel density estimators to model the background color distribution instead of a Gaussian mixture model.

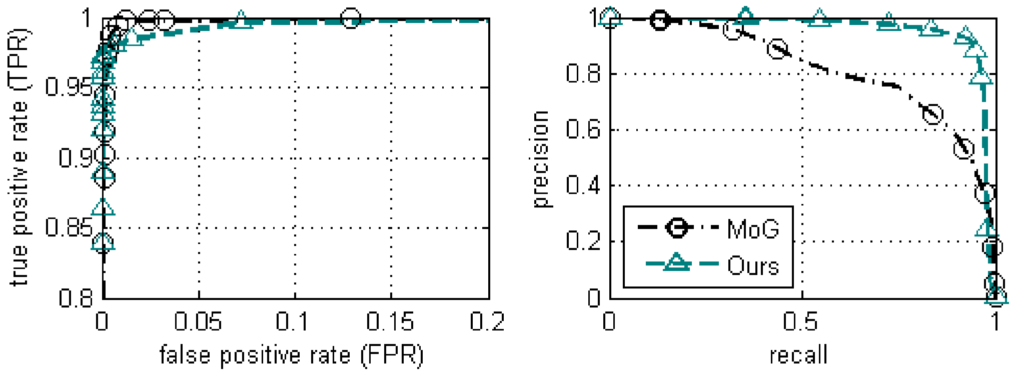
In this thesis, we will describe contributions we have made on reducing false positive foreground detections in the presence of dynamic textures (see Fig. 1-2 here or read more in §2), using foreground blobs for recognizing individual people (§5), and in combining background subtraction with stronger foreground models to overcome lighting and crowding challenges (§6). Before describing in §1.4 how tracking is done on the detections, we will discuss a few alternative approaches to detecting the objects of interest.

1.3.2 Layered Models

With background subtraction, we explicitly model the color distribution for each pixel. Observed colors that are unlikely under the model tend to result in corresponding foreground labels. In Eqn. 1.11, there is an implicit and naïve assumption that the color distribution for foreground objects is uniform. This begs the question, "Can we



(a) Typical False Positive Reduction



(b) Whole-Clip Performance

Figure 1-2: *Typical Improvements for Dynamic Textures:* Here we preview typical improvements seen by applying the model of §2 to background subtraction. In (a) true positives are given in blue and blob-level false positives are highlighted in red. Our model (right) is able to suppress false positives from trees waving in the wind (left). In (b), we show the achievable performance tradeoffs between the standard model and ours as Receiver Operator Characteristic (ROC) curves (left), and precision recall curves (right).

explicitly model the foreground too?" A helpful framework for discussing this topic is the layered model used by Darrell and Pentland [24] and expanded upon by Wang and Adelson [114].

In their framework, the world is a collection of independently-moving 2D planar layers. Each layer is parallel to the image plane and nearer layers occlude more distant ones. A layer is composed of a mask indicating where it is visible versus where more distant layers can be seen through it. It also has an appearance: a color distribution for every non-masked point.

Background subtraction as previously described is a degenerate layered system. The background layer is the farthest one and its color distribution is modeled as a point-wise mixture of Gaussians. There is a second layer that represents the foreground. The foreground color model is uniform at every location and the mask for the foreground layer is inferred in each frame using thresholding or an MRF. The only temporal constraint on the mask is the set of $\mathrm{T}_p(l_p)$ terms in Eqn. 1.12.

Wang and Adelson use their model to approach the problem differently. Given a pair of subsequent video frames, they explicitly search for a segmentation of the scene that is consistent with a collection of layers, each undergoing independent affine motion. They first compute dense optical flow using a method such as Horn and Schunck's [47](pages 280–293). They then use a form of agglomerative clustering to group pixels. Each pixel cluster's optical flow must be consistent with the flow that would be produced by a plane undergoing a single affine motion.

A few important details make it difficult to use Wang and Adelson's approach in practice. By using optical flow as a primary input feature, it becomes difficult to model thin objects like tree branches well. Most optical flow algorithms do not produce accurate flow fields in those situations. Furthermore, most optical flow algorithms perform best when there is a high degree of texture throughout the image. Unfortunately, their approach does not deal with the uncertainties that arise in optical flow estimation given untextured regions. Finally, the agglomerative clustering can be computationally burdensome.

In 2001, Jojic and Frey expanded on the work by Wang and Adelson by using a variational expectation maximization algorithm to solve the layer segmentation problem. Their method assumes there are a fixed and known number of layers and that the layers undergo translation only, not general affine motion. With a series of modeling approximations and extensive optimizations, the authors were able to achieve performance of 1 frame per second for 320×240 images and two layers.

Winn and Blake [120] extended the work of Jojic and Frey to handle general affine motions and tracking of appearance, but only for two layers. As part of that work, Winn developed an inference technique called variational message passing [121].

Zhou and Tao [129] used the layer model of Jojic and Frey but find the layers by bootstrapping with background subtraction. The layered model is sequentially optimized using a number of heuristics, allowing for automatic discovery of the number of layers over time. As the number of layers increases, their exhaustive search of layer ordering is likely to cause performance problems because the search is $O(L!)$, where L is the number of layers.

Kumar, Torr, and Zisserman [58] made significant improvements that automatically find the number of layers, handle rotation, translation, anisotropic scaling, motion blur, and brightness and contrast changes. Initially, layers are found using a method that resembles Wang and Adelson's, but coarse optical flow is computed and more sophisticated inference algorithms are used for the segmentation. Then they sequentially refine the layer boundaries, their appearances, and the lighting and transform parameters. Their algorithm takes several minutes per pair of frames on a 2.4GHz Pentium IV using C++ code.

The layered representation of video just discussed is appealing in that it is an intuitive abstraction. Like background subtraction, it is a weak model in the sense that no special training is needed to detect particular types of foreground objects, beyond some parameter tuning. It is also more powerful than background subtraction because non-stationary cameras can be handled within the layered framework without requiring a separate motion compensation mechanism.

Unfortunately, state-of-the-art layered motion algorithms suffer from one or more of the following shortcomings: (a) high computational complexity, (b) the number of layers must be fixed before inference, (c) only a very small number of layers can be managed without a combinatorial explosion and/or loss of model fidelity, and (d) inference must be done in batch. Furthermore, as Sand and Teller demonstrated [87], the real world is not actually layered. Consider the case where a camera is constantly pointed at the base of a tree and the camera's base revolves around the tree at a constant radius. While logically, one might assume that we could have a ground layer and a tree layer, it turns out that a world model with 2D layers parallel to the image plane cannot generate the observed motion.

Great progress has been made in layered modeling in recent years, particularly that of Kumar, Torr, and Zisserman [58]. Unfortunately, we believe that significant additional improvements are needed before it will be practical for usage in real-world settings with continuous video footage and complex motions. We will not be using layered models in this thesis.

1.3.3 Strong Object Models

In attempting to detect objects of interest, one can use a very weak model such as background subtraction. That model assumes that a pixel with an unexpected color observation is caused by a moving object. Because most interesting objects are spatially coherent[5], clusters of neighboring pixels with unlikely colors generally correspond to the same moving object. As described above, background subtraction makes several important assumptions: (a) the camera does not move (or that its movement can be recovered through some external mechanism), (b) moving objects are distant from each other in the image plane (or else their blobs will be merged), and (c) we only care about detecting objects that are moving (objects that do not move for time periods $\gg \frac{1}{\alpha}$ will tend to get incorporated into the background model).

One approach to addressing these concerns with background subtraction is layered modeling. One searches for the motion corresponding to the background layer just as one searches for motion in other layers, so moving cameras can be handled within

[5]A stretched-out net made of rope is one example of an object with limited spatial coherence: it is mostly composed of holes in image space.

the framework. As long as different objects have sufficiently distinct motion and appearance, they can be segmented from each other, even if they overlap in the image plane. Unfortunately, an object that does not move with respect to another overlapping one cannot be segmented from it. For example, a stationary person cannot be segmented from the background under a variety of camera motions[6].

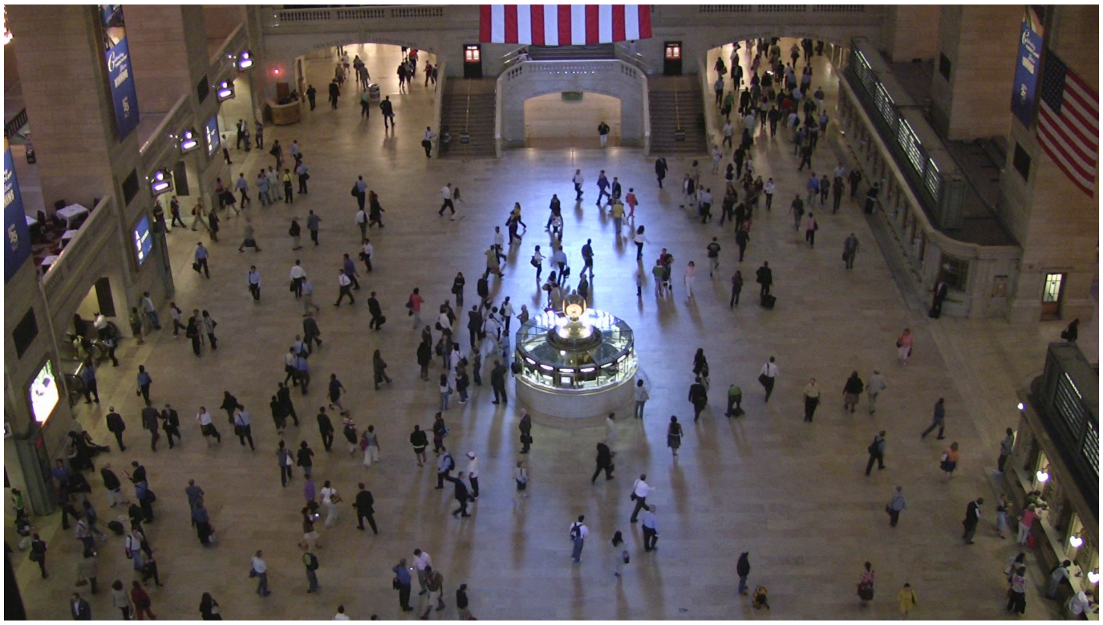
An alternative to these bottom-up approaches is a top-down one, which we motivate here. Consider the scene found in Fig. 1-3. For activity modeling purposes, there are essentially three types of objects in this scene: people, their possessions, and context. Contextual objects include:

- the information booth with its clock (in the center of the scene),
- a ticketing booth on the right,
- the floor,
- escalators to the left,
- staircases at the top of the image,
- support structures, and
- various exits to train platforms, subways, and connecting tunnels.

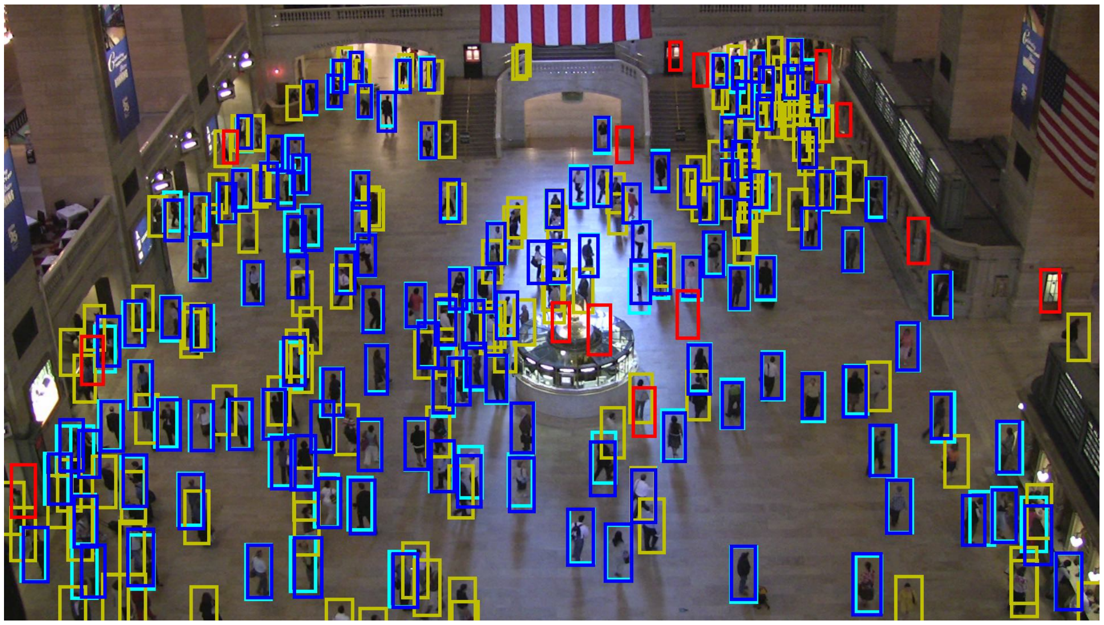
Pedestrians tend to travel from one tunnel or exit to another, gather at various locations on the floor, wait to purchase tickets, or loiter near the information booth, against walls, or in lower-traffic areas of the floor.

For a scene like this with fixed context, the most interesting computer vision problems relate to finding the pedestrians and tracking their movements relative to each other and the contextual objects. In this scene, we specifically care about finding pedestrians, and we do not care about finding objects of other classes like pencils, airplanes, chairs, or horses. While background subtraction and layered models are

[6]Due to parallax and foreshortening effects, sufficiently large camera translations can allow for separation, but pure rotations or no camera motion at all produce motion vectors that are identical with and without, regardless of the presence of stationary occluding objects.



(a) Large Image with Many People



🟦 true positives 🟦 matched ground truth 🟥 false positives 🟨 false negatives

(b) HOG-based Pedestrian Detections

Figure 1-3: *Scene with Dense Pedestrian Traffic:* (a) This is a still frame from an eastern-facing video observing the Great Hall in Grand Central Terminal in New York City. Bright spots on the floor are due to the morning sun coming through large windows on the east wall (not pictured) and producing glare on the polished marble floor and strong cast shadows from pedestrians. Bright spots on the wall are due to internal lighting and signage. (b) Using a pedestrian detector based on Dalal and Triggs' HOG features [19], good detection rates are possible despite the crowdedness of the scene.

largely agnostic to the type of foreground object, we know that we only care about one very particular kind of object.

Given this restricted problem domain, it makes sense to consider approaches that involve detectors specifically tuned for finding pedestrians. By doing so, we can hope to overcome the deficiencies of background subtraction and layered models. Unlike layered modeling, we can handle massive numbers of simultaneously visible objects. Unlike either layered modeling or background subtraction, we can hope to separate out individual people from groups of people that occlude each other and jointly move with coherent motion. What we lose by having a tuned object class detector is generality. We expect poor performance if there are significant changes in viewpoint or dramatically different poses. If an activity modeling system using one context were deployed in a new context with, say cars or farm animals instead of pedestrians, a new low-level detector would need to be created.

Pedestrian Detection Models

Soon after summarizing [35] the state-of-the-art in pedestrian detection, Gavrila [36] created a pedestrian detector for automotive applications. A novel image window is classified as a pedestrian by comparing it with a training corpus of edge images taken from registered and cropped photographs of pedestrians. He uses the Hausdorff distance: the average truncated distance of an edge pixel in the probe image to the nearest edge pixel in the training image. If the Hausdorff distance is small, the image window is classified as containing a pedestrian. For this to work well, the training corpus should cover the space of expected poses. To avoid having to compare each test window with the whole training corpus, he builds a template hierarchy by clustering training images which are similar, in a Hausdorff sense. An additional verification step is used to improve the results. For scenes with isolated individuals but sampled from an extensive space of human poses, Shakhnarovich *et al.* [91] demonstrated an efficient alternative mechanism for template matching.

Leibe, Seemann, and Schiele [63] created a pedestrian detection system for usage in crowded scenes. They learn local features that represent parts of pedestrians.

Hough voting is used to find possible pedestrian centroids, then various validation and cleanup steps are taken to produce the final result. They concentrate on eye-level views and achieved 65% recall at 80% precision or 45% recall at 90% precision on their test data.

Dalal and Triggs [19] developed a high-quality detector for upright pedestrians in general outdoor scenes. They scan the whole image looking for pedestrians. For each window, they extract a local feature descriptor on a regular grid. These descriptors are collections of histograms that count the number of image gradients in a particular direction, weighted by the gradient magnitude. Their Histogram of Oriented Gradients (HOG) representation is similar to the descriptors used in Lowe's Scale Invariant Feature Transform (SIFT) [66]. The local features are concatenated to form a single feature vector for the detector window. That feature vector is then classified as pedestrian or non-pedestrian using a linear Support Vector Machine (SVM). They tested their algorithm on a varied collection of manually registered positive samples and a set of random windows from images with no pedestrians. When adjusted to miss fewer than 5% of pedestrians, they achieve a false positive rate of less than 0.2%[7]. In §3 of this thesis, we will discuss their algorithm in detail.

The Dalal and Triggs detector has been extended with flow features to produce better results in tracking situations [20]. Zhu *et al.* [130] saw speedups between 16× and 70× relative to Dalal and Triggs and similar error rates by using a boosted cascade inspired by the Viola and Jones face detector [110]. Wojek *et al.* [122] achieved a 30× speedup on small images by re-implementing key portions of the algorithm to run on highly parallel graphics card (GPU) hardware. In §3, we describe a GPU port we have created that achieves a $58 - 76\times$ speedup on very large input frames.

We note that all of the strong-model pedestrian detectors described above require significant amounts of hand-labeled training data. For example, Dalal and Triggs' detector is trained with 2,416 manually cropped and scaled pedestrian images and 1,218 full images with no pedestrians. The latter set is relatively cheap to obtain: the

[7]We cite results for their "Lin. R-HOG" classifier. Although their "Lin. R2-HOG" and "Ker. R-HOG" classifiers perform better, they are prohibitively expensive in terms of speed.

annotator need only verify that no pedestrians exist in the whole image. The positive training set is however quite expensive. We found that just labeling the crown of each person's head with a single dot in images like Fig. 1-3 took 10–15 minutes per frame. In uncalibrated images like Dalal and Triggs', the whole bounding box must be selected, which is much more time-consuming. In §3.4, we will explore the effect of annotation quality. We will show that by providing more consistent registration in the training and/or test data we could achieve an extra 1% to 10% recall at 80% precision for various classification experiments.

A long-standing critique of strong models relates to this issue of training data. If the test data contain different articulations, viewing angles, or types of background clutter, a high-capacity learned model is likely to see serious performance degradation. One solution is to augment the training set for each new situation. In §3.4, we will show that in the Grand Central scene, we see an additive 15% improvement in recall at 80% precision in full-fledged detection experiments when we provide independent training data that mimics the viewpoint, background clutter, and occlusion characteristics of the test data. An alternative approach is to use bootstrapping: generate "training data" by using only the most confident detections from a more robust general detector. We will briefly explore an application of this idea in §2 by using background subtraction with a very weak object model to obtain data for a per-individual meanshift tracker.

General Object Recognition: Constellation Models

It is worth pointing out that the last decade or so has seen tremendous progress in general object recognition. Early work by Fischler and Elschlager [31] represented images as "pictorial structures." First, one builds robust detectors for parts of an object. For example, for face detection we might have specialized detectors for each eye, each ear, the nose, and the mouth. Then the composite face detector looks for constellations of low-level detections of the right types that are in the right spatial positions with respect to each other.

Leung, Burl, and Perona [64] used this model for face detection. Weber and

Welling [117] provided an unsupervised mechanism for automatically building classifiers for many different object classes given a large corpus of training images. Fergus, Perona, and Zisserman [28] proposed a method for efficient joint learning of appearance (the low-level detectors) and geometry (the spatial constraints on low-level detections).

General Object Recognition: Bag-of-Words Models

A collection of methods with even weaker geometric constraints derive from natural language processing research. It turns out that for natural language documents, one can often determine the collection of topics being discussed by having a simple word frequency model for each topic. Sports articles might commonly contain words like “run,” “win,” and “score.” Computer science articles might commonly contain words like “CPU,” “run,” and “algorithm.” Algorithms such as latent semantic analysis [59], probabilistic latent semantic analysis (pLSA) [45], and latent Dirichlet allocation (LDA) [105] can be used to automatically find topics: clusters of words that commonly co-occur in documents. Given a new document, one can determine the most likely topic or collection of topics to have generated it. These models are typically called “bag of words” models because they consider documents to be unordered collections of words.

In a bag of words object detection model, a training set of low-level features like Lowe’s SIFT features is generated from image data. The features are clustered into a codebook, resulting in a large but finite set of “visual words.” An image window is thought of as a document that contains these visual words, but their spatial locations within the window are ignored. It is represented as a histogram of the visual words. Sivic *et al.* [96] use pLSA to recognize new image windows. Grauman and Darrell [42] use an efficient multiset matching technique for recognition and other tasks called the “pyramid match kernel.” The bag of words models tend to be much faster than constellation models and have competitive performance despite encoding geometry only implicitly by having features taken from overlapping image patches.

Although general-purpose object detection for a large number of classes has seen important progress, specialized detectors such as Dalal and Triggs' often outperform them. As the field continues to progress, the ability of unsupervised and semi-supervised object model learning may very well allow general systems to dominate in the future.

1.4 Tracking

If we wish to do higher level analysis based on knowing the location history of a given object, we must associate object detections in one video frame with detections in other frames. If the scene is sparse, the detector is nearly perfect, and objects move slowly relative to the video frame rate, one can assume that if an object is at location p at time t, then at time $t+1$, the detection nearest to location p corresponds to the same underlying object. When the scene is not sparse, ambiguities can arise. One solution is to solve the "linear assignment problem." Given a set of detections $\mathbb{D}^{(t)} = \{d_i^{(t)}\}_i$ at time t, another set $\mathbb{D}^{(t+1)}$ at time $t+1$, and an assignment cost function $\mathrm{C}(\cdot,\cdot)$, one wishes to find an assignment $\hat{\mathrm{T}}(\cdot,\cdot)$ of every detection in $\mathbb{D}^{(t)}$ to a single detection in $\mathbb{D}^{(t+1)}$ that minimizes the total cost:

$$\hat{\mathrm{T}}(i,j) = \operatorname*{arg\,min}_{\mathrm{T}(\cdot,\cdot)} \sum_{i=1}^{\left\|\mathbb{D}^{(t)}\right\|} \sum_{j=1}^{\left\|\mathbb{D}^{(t+1)}\right\|} \mathrm{C}\left(d_i^{(t)}, d_j^{(t+1)}\right) \tag{1.16}$$

subject to the constraints

$$\mathrm{T}(i,j) \in \{0,1\} \;\; \forall i,j, \qquad \sum_{i=1}^{\left\|\mathbb{D}^{(t)}\right\|} \mathrm{T}(i,j) = 1, \quad \text{and} \quad \sum_{j=1}^{\left\|\mathbb{D}^{(t+1)}\right\|} \mathrm{T}(i,j) = 1. \tag{1.17}$$

The cost function $\mathrm{C}(\cdot,\cdot)$ may be as simple as the Euclidean distance between detection centroids or it may also include penalties for having different appearance characteristics or underlying detection types. Well-known solutions with $O(\left\|\mathbb{D}^{(t)}\right\|^3)$ complexity exist as well as extensions that allow for unmatched detections from each set [16].

If the real-world objects undergo Brownian motion, then it is sensible to have a cost function of the form $\mathrm{C}\left(d_i^{(t)}, d_j^{(t+1)}\right) \propto \mathcal{N}\left(d_i^{(t)}; d_j^{(t+1)}, \Sigma\right)$ for some Σ parameter. A more typical choice is to assume that objects move with a constant velocity or constant acceleration, plus some noise. A Kalman filter is a common way to estimate the location of an object given its prior location history [55]. Because standard Kalman filters use Gaussian distributions for modeling the uncertainty in location, velocity, acceleration, and observed versus actual location, they can be estimated efficiently in closed form. Welch and Bishop [118] provide a thorough discussion of the filter and its usage. Arulampalam *et al.* provide an overview of additional tracking techniques that relax the Gaussian assumptions and/or allow one to maintain several simultaneous data association hypotheses [4].

Zhao, Nevatia, and Wu [125] have built a sophisticated pedestrian tracking system that simultaneously optimizes foreground blob segmentation with tracking. For the background model, they use a single Gaussian per pixel. A color histogram model for each detected pedestrian is used along with the meanshift algorithm to assist with segmenting foreground objects. Additionally, they use a weak 3D pedestrian model composed of three ellipses. Because most of the inference is done in 3D coordinates, explicit occlusion handling can be integrated into the system with little additional difficulty. Their model also allows for easy adaptations to novel viewpoints. They assume pedestrians travel on a flat ground plane and that the camera calibration is known. For temporal estimation, they use Kalman filtering. After computing priors based on the maximum a priori (MAP) estimate of the state in the previous frame and from the current frame's appearance data, they use an Markov Chain Monte-Carlo (MCMC) sampler to perform frame-to-frame data association. Their MCMC optimizer utilizes a number of key efficiency improvements and was chosen instead of particle filtering to avoid a combinatorial explosion with non-trivial state spaces. They achieve an impressive 98% detection rate and 0.27% false alarm rate on their outdoor test dataset. With separate segmentation and tracking steps, the performance drops to 93% detections with 0.18% false alarms. On a 2.8GHz Pentium IV with 384×288 input images, their C++-based system runs at about 2 frames per second (fps). If we optimistically assume that all the algorithms are linear in the number of pixels (*i.e.* they are $O(\|\mathbb{P}\|)$), we estimate this is equivalent to 0.36fps on a 2.66GHz Core i7 with high definition 1920×1080 video.

In an alternative system, Zhao and Nevatia [124] included a texture model with exponential forgetting and a collection of 16 walking and 16 running HMMs for extracting gait phase information. Their emission model uses motion templates of the legs and is similar in spirit to the work we describe in §5.

In this thesis, we use standard constant velocity Kalman filters where tracking is needed.

1.5 High-Level Analysis

After detecting and tracking objects, high-level analysis in a site monitoring application can be performed. Examples include detecting the occurrence of pre-defined events, ad-hoc event querying, identifying individual people, characterizing co- occurring activities in the scene, and finding anomalous tracks.

1.5.1 Event Detection

Much of the research in automatic detection of pre-defined events has focused on using generalizations of language models to interpret the output of a vision system. Here we chronologically outline some of the major contributions to the area in the last decade, then we mention research on some recent standardized datasets.

Ivanov and Bobick [49] use a stochastic context free grammar (sCFG) to express semantics atop low-level detections. Results are demonstrated for gesture recognition and surveillance applications. In surveillance applications, they handle tracking failures in the sCFG layer.

Vu, Bremond, and Thonnat [111] use a logic-based approach to recognize predefined events. Their system works by solving a constraint satisfaction problem involving actors, logical predicates, and temporal relations between subevents. By incrementally building a representation of subevents that do not violate any constraints, they are able to have a realtime system that can manage multiple simultaneous actors.

In 2002, the system could handle 7 simultaneous actors at 10fps.

Taking in a collection of objects, low-level events, and temporal relations, Ghanem *et al.* [40] generate a deterministic Petri Net representation for events. Using a Petri Net inference engine, they detect instances of these events and can do more sophisticated operations like counting the number of cars parked in a parking lot. Inspired by their work, Dalley and Ižo [21] experimented with a system that uses automatically-parsed schematic diagrams as an input then performs fast inference using dynamically-compiled queries on large multi-camera tracking databases.

Hongeng, Nevatia, and Bremond [76] represent activities which are composed of action threads executed by a single actor. Each thread is represented by a stochastic finite automaton over atomic states detected by low-level trajectory and shape analysis. Multi-agent events are composed of action threads that satisfy a collection of temporal constraints. Due to the complexity of their inference methods, the system is most appropriate when there are a small number of actors in the scene that could be participating in the multi-agent event of interest.

François *et al.* [34] created a pair of standardized languages for representing video events and annotating them. Their ontology framework allows designers to specify things such as an entity type hierarchy, primitive properties, and deterministic rules.

Joo and Chellapa [53] use probabilistic attribute grammars with logical predicates to detect events such as casing a vehicle in a parking lot and boarding an airplane sitting on a tarmac. Cuntoor, Yegnanarayana, and Chellappa [18] uses HMMs to model characteristic trajectory paths indicative of events like open doors, exiting a building, or entering a car.

To complement the representational and inference work described above, standardized datasets were created in 2006 [29] and 2007 [30] for the Performance Evaluation of Tracking and Surveillance workshop (PETS). They come with an evaluation methodology and a collection of pre-specified deterministic rules defining events such as loitering, bag theft, and abandoned luggage. The foci are testing the robustness of low-level vision and evaluating whole tracking systems. A successful system must be able to tolerate lighting changes, strong specular reflections of the sun, many distracter objects, and simultaneous activities involving individuals and groups of objects. Here we highlight a few of the approaches used by various authors.

For PETS 2006, Auvinet *et al.* [5] perform background subtraction in each of the four provided camera views, they project silhouettes onto the ground plane, then they retain all projection intersections as detections. Their technique is reminiscent of the Visual Hull algorithm of Laurentini [60]. Tracking and rule satisfaction layers are built on top of the detectors. Their system works well in sparse scenes but is susceptible to a high false-positive person detection rate in dense scenes. Arsić *et al.* [3] expanded on this idea for the PETS 2007 challenge by looking for consistency in silhouettes projected onto planes parallel to the ground plane at different heights.

del-Rincón *et al.* [68] do multi-camera tracking using a variation on the standard Kalman filter. Object height is used to classify objects as pedestrian or not. Only humans who were ever near a dropped object are tracked.

For the 2006 dataset, Lv *et al.* [67] use frame-to-frame data association of blob detections, building up an appearance model. They then automatically switch to a meanshift tracker when the blob tracker fails. Event detection is done via probabilistic rules.

In §6, we describe our work on the more challenging 2007 videos. We combine techniques from Lv *et al.* and del-Rincón *et al.* with additional low-level techniques, allowing us to report the most comprehensive set of results on the dataset. In Fig. 1-4 we show the output of our system for several key frames of one of the test sequences.

1.5.2 Activity Co-occurrence and Anomaly Detection

In addition to detecting pre-specified events of interest, it is valuable to be able to characterize the trajectories that occur in a scene and cluster them such that semantically meaningful activities can be automatically discovered. This information can be directly useful in assisting users in understanding the scene. Such a model can also be used to flag anomalous observations.

Stauffer [99] produced a hierarchical clustering of tracks by vector-quantizing the tracker's state (position, velocity, and size) and computing a co-occurrence matrix

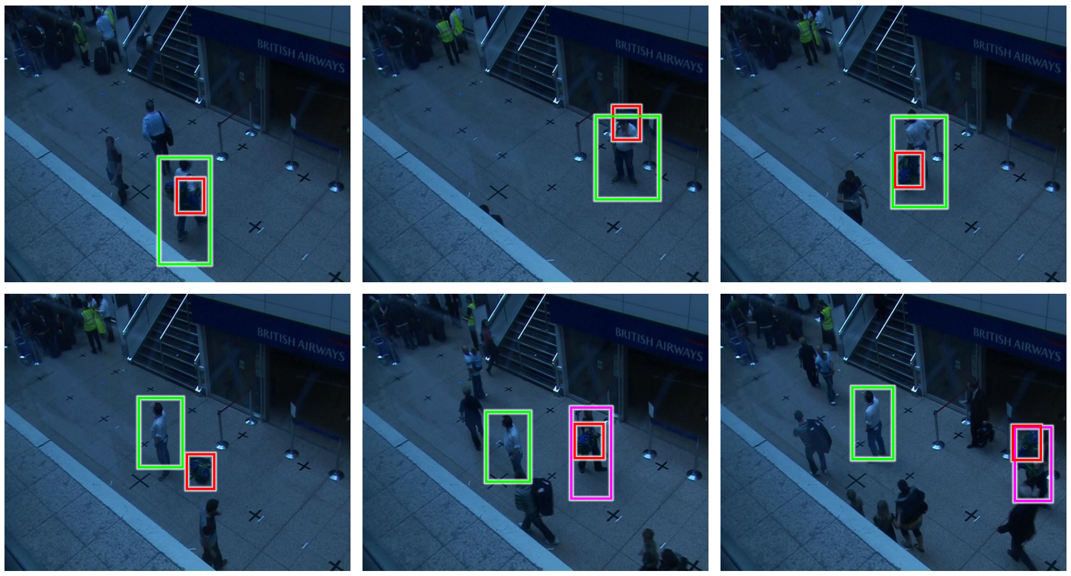


Figure 1-4: *PETS 2007 Event Detection Result:* Here we show tracker bounding boxes for several key frames from our event detection work for PETS 2007 [23]. The original owner's track is visualized with a green bounding box. He sets his bag (red) on the floor, which is stolen by a thief (magenta). Full details are in §6.

that indicates how often one quantized state occurs along with each of the other states within a track. By doing so, he was able to automatically extract different modes of behavior from a given scene such as separating different traffic lanes and distinguishing between pedestrian and vehicular traffic in a parking lot.

Wang, Ma, and Grimson [116] used a Dual-HDP as a more sophisticated and Bayesian co-occurrence model. A video from a traffic scene is partitioned into 10 second "documents" (in a language model sense). Whenever movement is seen within a block of pixels, a "word" is generated corresponding to that location and the quantized direction of movement. For a 720×480 video with 10×10 blocks and 4 moving directions, the vocabulary has about 14,000 distinct words. The model learns clusters of co-occurring places and directions of motion. In a second layer of clustering, it also learns how the low-level clusters co-occur with each other. An infinite mixture model is used which adapts the number of learned clusters at both levels in a data-driven manner and both levels of clustering are jointly optimized with Gibbs sampling. This approach learns semantically meaningful clusters in an unsupervised manner when different video partitions contain distinctive activities. They demonstrate results that show the model learning clusters that correspond to different flow patterns related to different periods in a traffic signal's cycle. Results are also shown for video segmentation, anomaly detection, and ad-hoc querying applications.

Unfortunately, in some scenes, activities last much longer than a few seconds to a few minutes. Consider the Grand Central scene shown in Fig. 1-3. With manual inspection of the video, one can see a number of activities:

- *Through-traffic:* The concourse's main floor has 16 entry/exit points and there are clear flows between the busiest pairings.
    - *Subway $\leftrightarrow$ Midtown:* For example, much of the traffic travels between either of the two large entrances on the top edge of the image to the escalators on the left edge.
    - *Secondary Traffic:* Smaller traffic flows travel between other entry/exit pairs.
- *Loiterers:* The concourse is popular for waiting, meeting, and touring.
    - *Information Booth:* The information booth in the center is a common place for people to meet, ask questions of terminal personnel, and loiter.
    - *Pillars:* Other common loitering spots include the vertical structures on the left side of the image.
    - *Ad-hoc Clusters:* Tour groups and clusters of independent loiterers will tend to form in low-traffic areas. As they form, those traveling between low-traffic entry/exit pairings often need to adjust their routes.
    - *Ticketing:* On the right side of the image is one of two banks of ticketing booths. In the afternoon and early evening hours, queues 5–10 people deep form.

In scenes with timed traffic lights, activity phases tend to last on the order of 5 seconds to a few minutes. In less-regulated environments like Grand Central, the distribution of the aforementioned activities tends to be stable on hour-long timescales.

Because of this, the 10-second Dual-HDP model learned by Wang, Ma, and Grimson's approach becomes degenerate: it either learns a single cluster describing all scene motion, or it creates clusters driven by noise. One solution could be to increase the document timescale and use data collected over extended periods of time. We have been investigating this approach but we note that it poses significant challenges. Learning a simpler standard HDP model on a single hour of video after pruning out 98% of the data takes about 20 hours. Learning time is super-linear[8] in the amount of data, so modeling longer time periods will be more expensive. Care will be needed to reduce computation and memory requirements so the model may be scaled up to handle day- or week-long video. This is an area of future research.

As an alternative approach, Wang *et al.* [115] changed the representation over which the Dual-HDP operates. The language model's vocabulary is still the cross product of position with moving direction, but individual tracks are used as the documents instead of video clips. This allowed them to analyze oceangoing ship behavior using coastal radar data, and pedestrian and automobile behavior in a parking lot scene. In §4, we will model activities at a transportation hub using a similar approach coupled with a novel and efficient Dalal- and Triggs-style pedestrian detector implementation described in §3. The learned activities in §4.3 are more semantically meaningful than those using sparse optical flow to produce low-level features in §4.2. Furthermore, because we use tracks from actual pedestrian detections rather than simpler flow-based methods of acquiring movement information, our system can be more readily extended with other modules such as gait recognition.

1.5.3 Individual Person Recognition

If a person is entering a sensitive area, a site monitoring application can be assisted by knowing who the person is. Interdiction may be needed when an unauthorized person enters or a system may need to maintain a "watch list" of suspects. Similarly,

[8]A single Gibbs sampling round is linear in the number of clusters and the amount of input data. By design, HDP-based models generate more clusters when there is more data. More clusters often also require more Gibbs sampling iterations to reach convergence.

when an event detection system triggers an alert, individual identity information can be helpful to human operators in knowing how to respond.

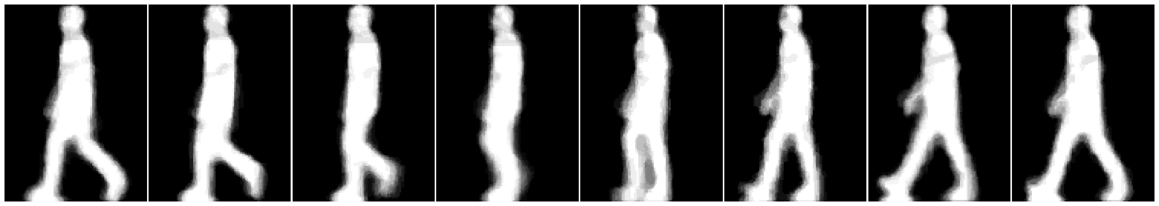
Perhaps the most commonly-considered remote biometric is facial appearance. Sinha *et al.* [95] have studied the biological characteristics of face recognition performed by the human brain, and Zhao *et al.* [126] provide an extensive survey of face recognition techniques available in 2003. Challenges for face recognition include sensitivity to lighting conditions, difficulty with varying facial expressions, low performance for non-frontal viewpoints, and the ability to mask appearance through changes in facial adornment such as beards, mustaches, and glasses.

In search of other biometrics, the US Defense Advanced Research Projects Agency (DARPA) in 2001 began funding an initiative called Human Identification from a Distance (HID). A standardized dataset was created as well as a baseline algorithm to study ways of identifying individual people based on their appearance and gait [82, 88]. The best results during the initiative used a hidden Markov Model (HMM) by Sunderesan, Chowdhury, and Chellapa [103]. They model the appearance of pedestrian silhouettes as a vector of distances to a collection of exemplar silhouettes, conditioned on discrete phases of the walking cycle.

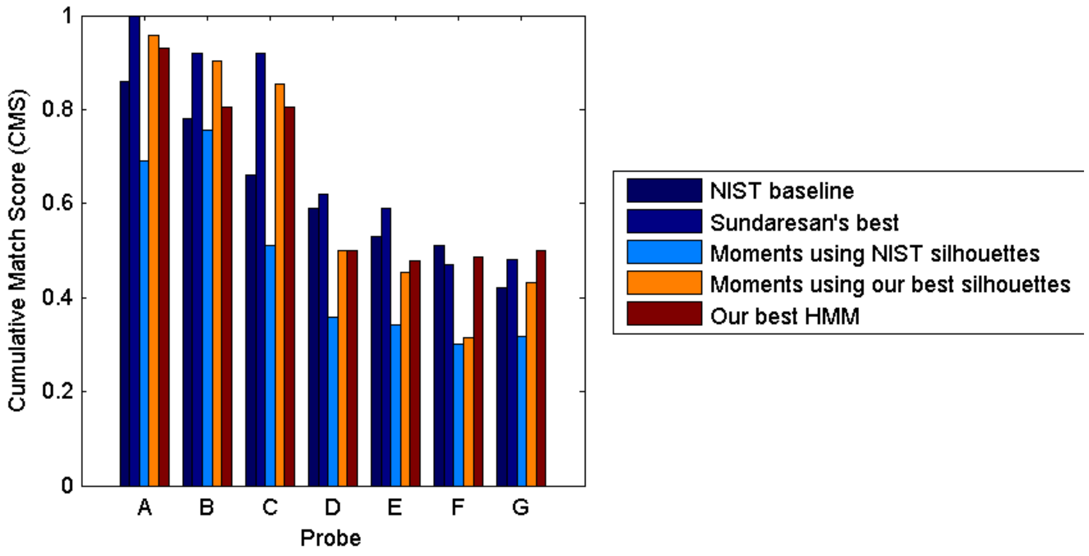
In §5, we describe a similar model that was developed contemporaneously that of Sunderesan *et al.* Our observation model is a field of independent binomial random variables over the silhouettes themselves. A preview of our recognition results is shown in Fig. 1-5.

1.6 Contributions and Thesis Organization

As described in this chapter, a substantial amount of work has been done in creating systems for recognizing individuals and interactions amongst them in site monitoring applications. Now that the computer vision field has made a pass through activity recognition, there are a number of areas that can be enhanced to provide the opportunities for further improvements. In this thesis, we discuss a collection of four systems that are aimed at improving the robustness, utility, and scale of existing solutions.



(a) HMM Observation Model Example



(b) Abbreviated Rank 5 Recognition Results

Figure 1-5: *Pedestrian Recognition Results Preview:* In (a) we show the learned HMM observation model for one person in the USF dataset [81]. Brighter pixels correspond to a higher likelihood of observing foreground at that corresponding position in a silhouette. In (b) we see that using cleaner silhouettes results in better performance when using ellipse moments (compare the light blue and orange bars). Although our focus was on silhouette quality improvements, our methods compare well to the contemporaneously-developed work of Sunderesan *et al.* [103] which focused directly on the recognition task. The probe sets are ordered from easiest to hardest. Full details are given in §5.

In §2, we present a mechanism for improving background subtraction results in the presence of temporally irregular dynamic textures such as leaves fluttering in the wind or rippling water. We achieve state-of-the-art results, demonstrating superior performance simultaneous with less sensitivity to other parameter choices. Our solution can be readily integrated into any Mixture of Gaussian background subtraction system.

In §3, we quantitatively demonstrate the shortcomings of using a weak appearance model such as background subtraction or corner detection in large crowded scenes. By using a strong model, we are able to detect and track individual people and we use these tracks to build a site activity model in §4. Our solution uses the Histogram of Oriented Gradients (HOG) detector of Dalal and Triggs [19], and we provide additional experiments investigating the interplay of annotation registration, camera viewpoint, and test protocols. We have created an open-source HOG implementation that takes advantage of the processing power of modern consumer graphics cards (GPUs) to achieve 1 fps on our full tracking system on 1920×1080 high definition video, despite performing an exhaustive search for pedestrian detections in every frame. The detection pipeline itself is $58\times$ to $76\times$ faster than our optimized CPU-only version. Based on performance extrapolations, we believe our implementation offers double the speedup relative to the fastest prior solution [122]. This extra throughput makes it practical to track pedestrians in large scenes over extended periods of time. We show preliminary activity modeling results based on our tracking output, using a standard HDP model similar to the Dual-HDP Wang *et al.* [115].

In some site monitoring applications, it is important to be able to identify individual people. Given the track of an individual moving fronto-parallel to the image plane, we demonstrate a system for determining the individual's identity in §5. We evaluate robustness to real-world silhouette noise and present HMM-assisted ranking functions that are competitive with the best contemporaneously-developed solution, especially for the most challenging scenarios.

In §6, we describe an event detection system built with constrained development time and a standardized dataset. The system is able to overcome challenges presented by quickly changing lighting conditions, trivial amounts of available training data, and crowdedness in the scene. Despite using weak models in each part of our hybrid system, it produces the most complete set of detection results known to us on the PETS 2007 data. This work reminds us as researchers of problems that need to be addressed by systems that are meant to be deployed in real-world settings.

We summarize our findings and contributions in §7.

As a convenience to the reader, in Appendix A we provide a summary of the notation and quantitative evaluation methodologies used in this thesis. Those unfamiliar with ROC or precision-recall curves will want to read the appendix before proceeding with any of the body chapters.

Chapter 2

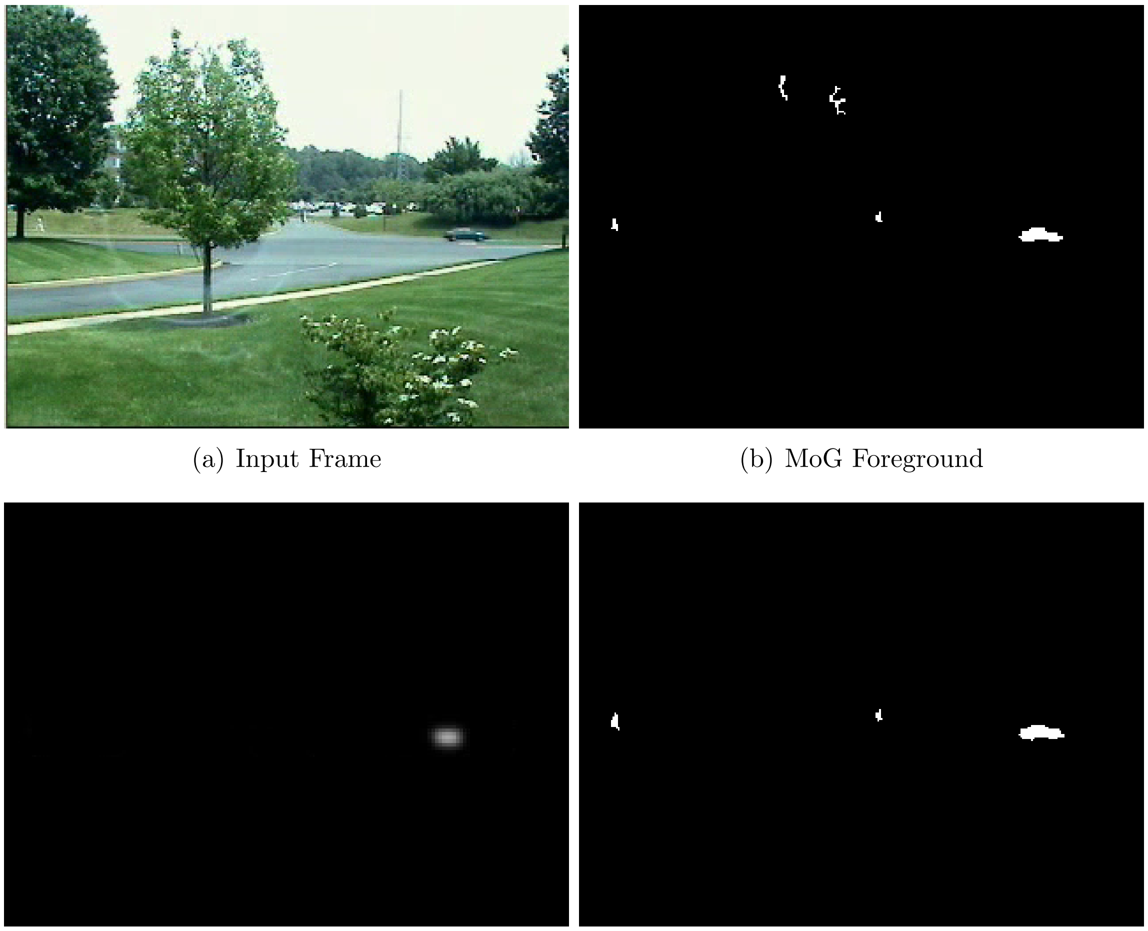
Background Subtraction with Temporally Irregular Dynamic Textures

This chapter contains joint work with Josh Migdal and Eric Grimson [22].

2.1 Introduction

A typical approach to moving object detection in current scene analysis systems is to build an adaptive statistical model of the background image, of the form introduced in §1.3.1. When a new frame is presented, pixels that are unlikely to have been generated by this model are labeled as foreground.

Stauffer and Grimson [101] represent the background as a mixture of Gaussians (MoG). At each pixel, a collection of Gaussians emits values in RGB (red, green, blue) or some other colorspace. When a pixel value is observed in a new frame, it is matched to the Gaussian most likely to emit it. The Gaussian is then updated with this pixel value using an exponential forgetting scheme that approximates an online k-means algorithm. This allows online adaptation to changing imaging conditions such as shifts in lighting or objects that stop moving. Pixel values are labeled as foreground when they are associated with uncommon Gaussians or when they do


(a) Input Frame

(b) MoG Foreground

(c) Mittal & Paragios [72]

(d) Our Foreground

Figure 2-1: *Foreground Classification Comparison:* Our method is able to suppress the two false positive waving tree foreground blobs seen in the MoG results. We are simultaneously able to retain the two very small pedestrians, unlike the previously best results on the video.

not match any Gaussian well. This approach lends itself to realtime implementation and works well when the camera does not move and neither does the "background." However, for most applications, objects such as branches and leaves waving in the wind, and waves in water, should be considered as background even though they involve motion. Because these dynamic textures cause large changes at an individual pixel level, they typically fail to be modeled well under a fully independent pixel model. In Fig. 2-1(b), we see how the MoG foreground mask not only (correctly) includes both pedestrians and the vehicle, but also includes many other pixels due to image noise and moving trees.

More recently, Mittal and Paragios [72] used the most recent T frames to build a non-parametric model of color and optical flow, with care taken to handle measurement uncertainty when estimating kernel density bandwidths. Uncertainty management is especially important here due to the inherent ambiguities in local optical flow estimation. Although their approach still models the image as a collection of independent pixels, they produce impressive results as long as the same motions are observed many times in every block of T frames. In Fig. 2-1(c), we see that their system suppresses the false positives on the tree. Challenges are likely to occur when infrequent motions occur, such as trees rustling from time to time (but not constantly) due to wind gusts. Better classification performance results in a cost linear in T. For a 200-frame window, their highly optimized implementation is one to two orders of magnitude slower than typical MoG implementations.

Sheikh and Shah [93] have also developed a kernel-based model of the background using the most recent T frames. Their kernels are Gaussians over the pixel color and location. By allowing observed pixels to match kernels centered at neighboring pixel locations, they are able to interpret small spatial motions such as trees waving in the wind as being part of the background. Like Mittal and Paragios, they must maintain a long enough kernel history to represent all modes in the local background distribution. Fortunately, for many types of scenes, this history length will be shorter for Sheikh and Shah since information can be "shared" by kernels spawned at nearby pixel locations.

We will show that our approach is able to achieve similar sharing benefits, and we do so by including a small set of easily implemented modifications to any standard MoG system. In Fig. 2-1(d), we see that our method is able to avoid false positives from the shaking tree while still detecting the small pedestrians.

Nam and Han [73] recently published a background subtraction method that uses particle filtering to track the positions of the generative model's pixel processes. They use a constant velocity (plus Gaussian noise) motion model, and they represent the appearance distribution of an individual pixel process as a color histogram. In order to make the problem tractable, they make several simplifying assumptions that allow for independent decisions in the inference and update stages. Limited quantitative results are given.

Zhong and Sclaroff [127] use an autoregressive moving average model for scenes with highly regular dynamic background textures like consistently rippling water or escalators moving at a constant pace. For training clips of 96 frames, they retain 80 eigenimages. Unlike their approach, our method is designed to work well when temporal structure is lacking from the dynamic textures.

Jojic and Frey [52] have taken a radically different approach, extending a model proposed by Wang and Adelson [114]. They consider an image to be generated by a collection of layers, where near layers occlude far ones. Their model assumes that the number of layers and their depth ordering are known and fixed. Each layer is free to translate across the image. Extensions include Winn and Blake's [120] affine motion model. Because finding the optimal solution is intractable, they employ variational approximations to their model. Unlike the other methods mentioned, their approach is batch-mode, so it cannot be used as-is on continuous video feeds.

Our work is most closely related to that of Stauffer and Grimson and of Sheikh and Shah. We combine the usage of a spatial neighborhood in background likelihood estimation with the compactness of a semi-parametric MoG representation. In §2.2, we describe our generative model and how we perform inference on it. In §2.3, we then highlight experiments we have performed on our algorithm. We summarize in §2.4.

2.2 Our model

Our model uses the same intuition expressed in the kernel-based model of Sheikh and Shah [93]: small local shifts of background object locations should not cause them to be flagged as foreground. For example, consider the case where a given pixel normally is observing the tip of a leaf and our model has learned this. If a sudden puff of wind occurs, the leaf tip may temporarily move by a pixel and occlude part of the sky. When observing the green pixel at a normally-blue location, we would like to acknowledge that its appearance is likely due to movement of the leaf tip moving rather than a new foreground object appearing.

To capture this concept of small local motions, we model the image generation process as arising from a mixture of components that have a Gaussian distribution in color and **some spatial distribution**, depicted in Fig. 2-2(c) and formalized here:

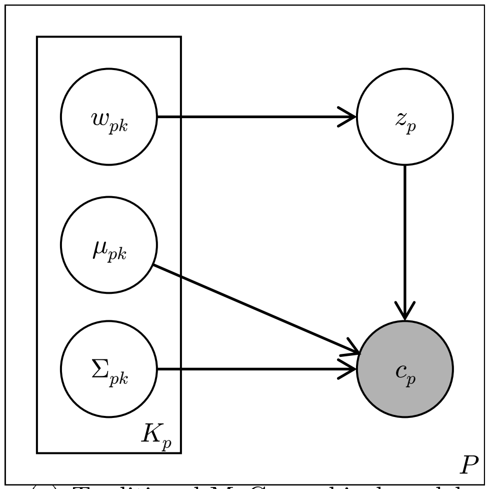
$$p\left(c_p \;\middle|\; \Phi\right) = p\left(c_p \;\middle|\; z_p, \Phi\right) p\left(z_p \;\middle|\; \Phi\right), \tag{2.1}$$

where c_p is the observed color at pixel location p, and

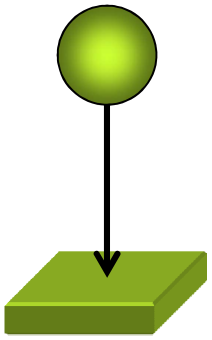
$$\Phi = \left\{\omega_{qk}, \mu_{qk}, \Sigma_{qk} \;\middle|\; q \in \mathbb{N}_p \;\wedge\; k \in \{1, 2, ..., K_q\}\right\} \tag{2.2}$$

is our model where mixture components are indexed by location, q, and an arbitrary local index k; ω_{qk} is the weight of a component, μ_{qk} is its mean color, Σ_{qk} is its color variance, K_q is the number of mixture components at pixel location q, and $\mathbb{N}_p$ is the set of pixel locations **in some local neighborhood about** p. In practice, we typically make $\mathbb{N}_p$ be a 3×3, 5×5, or 7×7 grid of pixels centered around p. A 3×3 grid is depicted in Fig. 2-2(d).

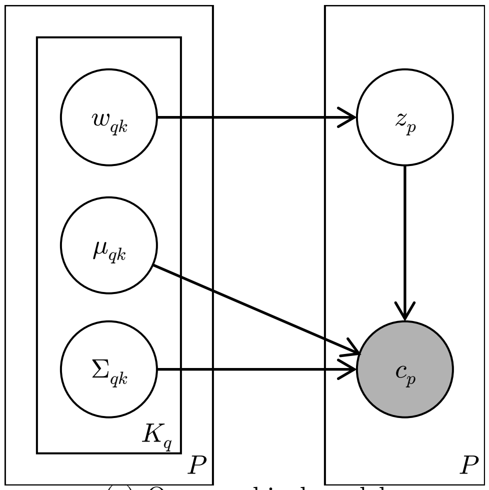
Each c_p is generated from a single independently-chosen mixture component $z_p =$



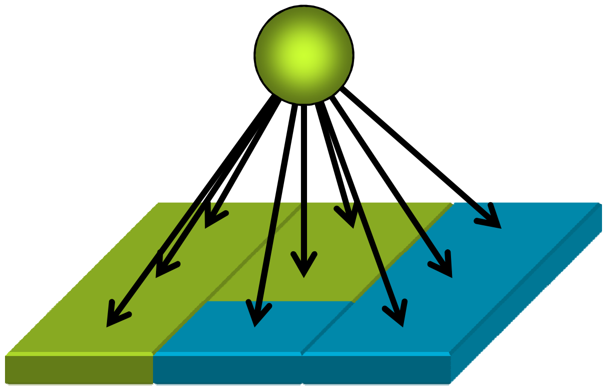
(a) Traditional MoG graphical model



(b) Mixture component area of influence (MoG)



(c) Our graphical model



(d) Mixture component area of influence (ours)

Figure 2-2: *Traditional MoG Model vs. Ours:* In Fig. 2-2(a) and Fig. 2-2(c), we show a graphical model representation of the traditional Mixture of Gaussians (MoG) background subtraction model and our model, respectively. Circles represent random variables, shaded circles represent observed values, and plates indicate replication of random variables which are independent except where there are arrows. In the traditional MoG model, the color c_q observed at pixel q is generated from a randomly-selected mixture component at the same pixel location. A given mixture component can at most influence the observed color for a single pixel (see Fig. 2-2(b)). In our model, c_q is sampled from mixture components at location q as well as components located at neighboring pixels. Each mixture component can influence the observed colors within some local neighborhood (a 3×3 neighborhood is shown in Fig. 2-2(d)). This extra spatial neighborhood allows our model to more gracefully handle backgrounds that undergo small local motions. See the text for precise definitions of the random variables.

(q, k) and

$$p\left(c_p \,\middle|\, z_p, \Phi\right) = \mathcal{N}\left(c_p; \mu_{z_p}, \Sigma_{z_p}\right), \tag{2.3}$$

$$p\left(z_p = (q,k) \,\middle|\, \Phi\right) \propto \begin{cases} \omega_{qk} & \text{if } q \in N_p \wedge k \in \{1, 2, ..., K_q\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

The same independent pixels assumption is made with nearly all non-layered approaches, including ones that have a spatial component (*e.g.* Sheikh and Shah [93]). Our model is equivalent to the traditional MoG model when the neighborhood is degenerate, *i.e.* when $\mathbb{N}_p = \{p\}$. In Fig. 2-2, we show a comparison between our model and the traditional MoG model using a directed graph representation and a per-mixture component area of influence illustration.

Note that we are not restricted to the RGB colorspace for observations. As with other models, we are free to use other colorspaces (such as YCrCb) or build an observation space over more exotic features such as spatio-temporal gradients [83] or optical flow [72]. In our experience we have found that when a proper neighborhood size is chosen, the background-foreground labeling is less sensitive to the choice of colorspace or the inclusion of optical flow features.

2.2.1 Foreground-Background Classification

The primary purpose of most background models is to determine the likelihood that each pixel was generated from the background process. In classic MoG approaches, the model is the same as Eqn. 2.1, with the constraint that the neighborhood function is degenerate and only selects mixture components at the same location where the colors are sampled. A collection of mixture components is maintained, where only those with the highest weights are considered part of the model and used in the likelihood evaluation. Under the assumptions that all Gaussians have similar covariances, all background Gaussians have comparable weights, and that they do not overlap significantly, the squared Mahalanobis distance

$$d_{pqk}^2 = (c_p - \mu_{qk})^\top \Sigma_{qk}^{-1} (c_p - \mu_{qk}) \tag{2.5}$$

serves as a good proxy for the negative log likelihood, and it can be computed much more efficiently than the precise likelihood value (recall the discussion in §1.3.1). For the experiments presented in this thesis, we have followed this tradition and used the squared Mahalanobis feature for foreground-background classification.

After the model returns the pixelwise likelihood estimates, a higher-level procedure is responsible for classifying each pixel as foreground or background. Common choices for the external classifier often include some combination of a simple thresholder, a Markov Random Field (MRF) optimizer to perform uncertainty-aware label smoothing, morphological operations to remove isolated foreground detections and merge disjoint blobs, and higher-level detection, tracking, or explicit foreground modeling to filter the results. The external classifier choice is outside the scope of the model itself. For the results shown in this chapter, we used Migdal and Grimson's [70] MRF that we discussed in §1.3.1.

2.2.2 Model Update

A model consisting of a single Gaussian may be updated online as new observations are obtained in an optimal manner by retaining its sufficient statistics. Mixture distributions add the complexity of needing to know which observations were generated from which mixture components. In this section, we give a more detailed view of the update procedure than we gave in §1.

Stauffer and Grimson [101] use an online approximation of expectation maximization (EM). Given a pixel location p, they find the observation likelihood $\mathcal{N}(c_p; \mu_{qk}, \Sigma_{qk})$ for each mixture component (q, k) (recall that $p = q$ for the traditional model), and then update its sufficient statistics by assuming an evidentiary weight of $(1 - \rho)$ for the old statistics and ρ for the new data point, where $\rho_{pqk} = \alpha \mathcal{N}(c_p; \mu_{qk}, \Sigma_{qk})$ for some exponential learning rate α.

Typical hard EM-like implementations simplify this further by only updating the most likely Gaussian using $1-\alpha$ and α as evidentiary weights. Depending on initialization and the order of online updates, the second approach tends to yield tighter Gaussian distributions that overestimate the covariance less. More recently, Porikli and Thornton [84] used a richer prior model with a greedy update scheme to improve the updates and reduce the effects of the update ordering. All three of these update mechanisms have been used on likelihood models essentially equivalent to Eqn. 2.1, with the neighborhood size restricted to only consider Gaussians and observations at the same pixel location.

In our model, we allow a pixel's color to be generated from Gaussians positioned at nearby pixel locations. This means each Gaussian has the possibility of independently generating multiple observations and thus potentially needs to be updated from multiple simultaneous measurements. One way of accomplishing this is to retain the time-weighted sample sum $s_{qk}^{(t)}$, squared sample sum $corr_{qk}^{(t)}$, and total effective sample size $e_{qk}^{(t)}$ as follows:

$$s_{qk}^{(t)} = (1-\alpha)s_{qk}^{(t-1)} + \sum_{p\in\mathbb{N}_q} \tilde{\rho}_{pqk}^{(t)} c_p^{(t)}, \tag{2.6}$$

$$corr_{qk}^{(t)} = (1-\alpha)\,corr_{qk}^{(t-1)} + \sum_{p\in\mathbb{N}_q} \tilde{\rho}_{pqk}^{(t)} c_p^{(t)} \left(c_p^{(t)}\right)^\top, \text{ and} \tag{2.7}$$

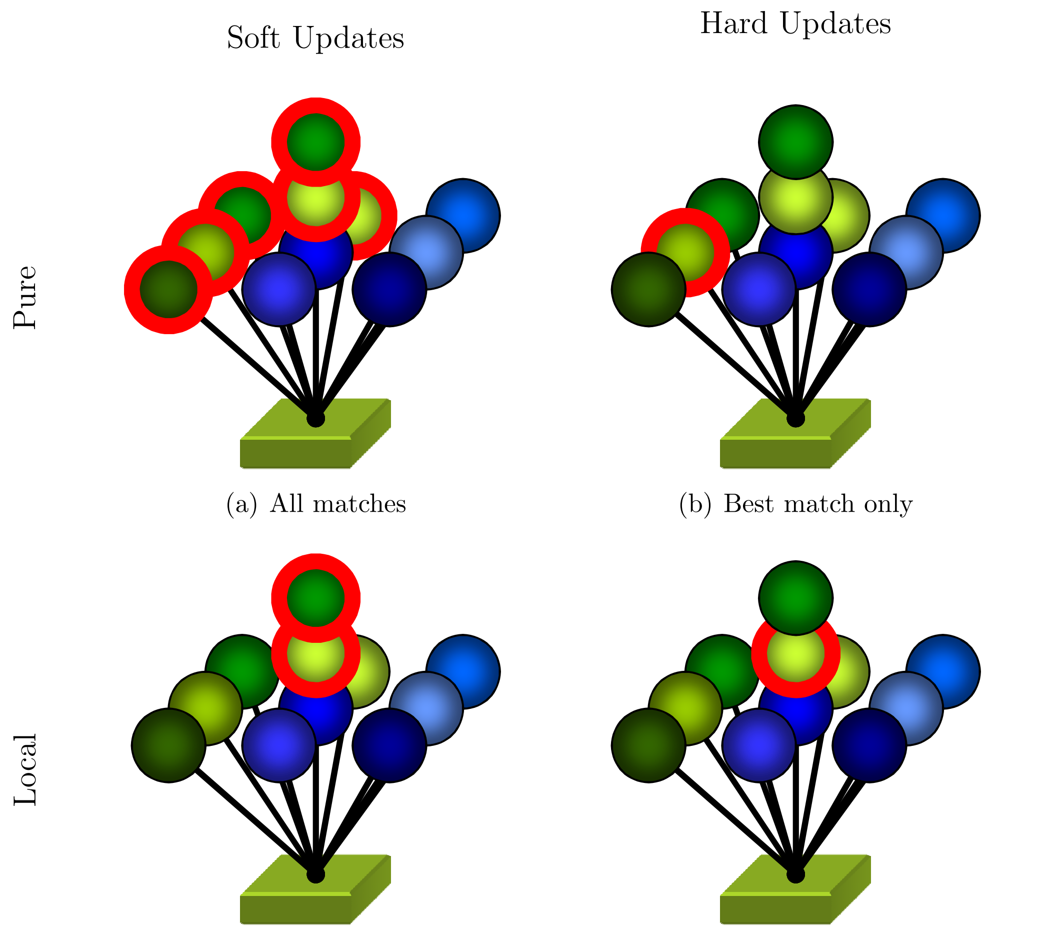
$$e_{qk}^{(t)} = (1-\alpha)e_{qk}^{(t-1)} + \sum_{p\in\mathbb{N}_q} \tilde{\rho}_{pqk}^{(t)}, \tag{2.8}$$

where α recursively downweighs old samples and ρ_{pqk} is the contribution of the observation at pixel p to Gaussian (q,k), and $\tilde{\rho}_{pqk} = \rho_{pqk}/\sum_k \rho_{pqk}$. Our model parameters are

$$\mu_{qk}^{(t)} = s_{qk}^{(t)}/e_{qk}^{(t)}, \tag{2.9}$$

$$\Sigma_{qk}^{(t)} = corr_{qk}^{(t)}/e_{qk}^{(t)} - \mu_{qk}^{(t)}\left(\mu_{qk}^{(t)}\right)^\top, \text{ and} \tag{2.10}$$

$$\omega_{qk}^{(t)} = e_{qk}^{(t)}\Big/\sum_{q',k'} e_{q'k'}^{(t)}. \tag{2.11}$$



(a) All matches

(b) Best match only

(c) All local (equivalent to soft Stauffer-Grimson)

(d) Best local only (equivalent to hard Stauffer-Grimson)

Figure 2-3: *Model Update Options:* There are several possibilities for choosing which Gaussian mixture components should be updated given an observation. In this illustration, consider a single observed yellowish green pixel, depicted as a block. Suppose our background model at the same location has three mixture components: one green, one yellowish green, and one blue. These components are depicted as the stack of three spheres in the center of each subfigure. For simplicity, further suppose that we have a single background mixture component for each of the neighboring pixel locations, shown as additional blue, yellow, and green spheres. We can update all mixture components in the whole neighborhood whose Mahalanobis distance is less than τ_{match} (the red-highlighted spheres in Fig. 2-3(a)). Alternatively, one could update only the single best match (Fig. 2-3(b)) or perform updates locally, as if a pure Stauffer-Grimson model were being used (Fig. 2-3(c) and Fig. 2-3(d). In practice, all four methods yield similar performance, but hard local updates are significantly faster.

The question at this point is how to assign the update weights ρ_{pqk} amongst all of the Gaussian mixture components in the neighborhood of pixel p. There are several logical possibilities, including those shown in Fig. 2-3:

- *Pure Soft:* For each observation, c_i, we update all mixture components that could have generated it, weighting by the likelihood of being generated by that Gaussian, *i.e.*

$$\rho_{pqk} \propto \begin{cases} \mathcal{N}(c_p; \mu_{qk}, \Sigma_{qk}) & \text{if } p \in N_q \wedge d^2_{pqk} < \tau_{match} \\ 0 & \text{otherwise,} \end{cases} \tag{2.12}$$

  where τ_{match} is some threshold that allows us to avoid updating poor matches. If no mixture components pass the τ_{match} test, we assume some previously-unseen mixture component generated the pixel and we instantiate a new component k' at location p instead of performing an update.

- *Pure Hard:* We choose the single mixture component which was most likely to have generated the sample and update it alone, *i.e.*

$$\rho_{pqk} = \begin{cases} 1 & \text{if } (q,k) = \underset{(q',k'):q' \in N_i}{\arg\max}\ \mathcal{N}(c_p; \mu_{q'k'}, \Sigma_{q'k'}) \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

  If $d^2_{pqk} \geq \tau_{match}$ for the selected component, we perform the new component instantiation as was done for *pure soft* updates.

- *Soft Local:* We perform Stauffer- and Grimson-style soft updates by only updating mixture components at the same location as the observation, *i.e.*

$$\rho_{pqk} \propto \begin{cases} \mathcal{N}(c_p; \mu_{qk}, \Sigma_{qk}) & \text{if } p = q \wedge d^2_{pqk} > \tau_{match} \\ 0 & \text{otherwise,} \end{cases} \tag{2.14}$$

  handling outliers in the usual fashion.

- *Hard Local:* We perform Stauffer- and Grimson-style hard updates, handling outliers in the usual fashion, *i.e.*

$$\rho_{pqk} = \begin{cases} 1 & \text{if } p = q \wedge k = \arg\max_{k'} \mathcal{N}(c_p; \mu_{qk'}, \Sigma_{qk'}) \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

While the *pure soft* scheme is appealing from a Bayesian perspective, it (and *soft local*) also requires that we actually evaluate the likelihoods, $\mathcal{N}(c_p; \mu_{qk}, \Sigma_{qk})$. The *hard* approaches only require evaluating the much more computationally-efficient squared Mahalanobis distances.

In Fig. 2-4, we have plotted the relative computational costs of the various update methods, relative to the baseline traditional MoG approach (*hard local* with $W = \|\mathbb{N}_p\| = 1$). These plots aggregate the results from running with a variety of parameter settings on several different machines while processing the traffic sequence from Mittal and Paragios [72]. The *local* approaches are relatively unaffected by the neighborhood size, W, since they do not iterate over the whole neighborhood during the update phase. It is clear that the *pure soft* approach incurs a significant additional performance penalty due to its requirements of full likelihood evaluation and that potentially all mixture components in the local neighborhood about a pixel must be updated.

For the same set of experiments, we show in Fig. 2-5 the costs of producing the Mahalanobis distance maps required by the foreground/background classifier. Given an update scheme, we expect the computational cost to rise linearly with the neighborhood area (quadratically with the neighborhood diameter). The update schemes are independent of this step, so it is interesting to note that by either choosing *hard* updates and/or *local* updates, the Mahalanobis calculations become faster. When we do *hard* updates, we force each sampled pixel to affect exactly one mixture component. Similarly, *local* updates can only affect a smaller pool of components. The net effect is that a slightly more compact model can be learned. This more compact model is more computationally efficient as well because fewer Mahalanobis distance

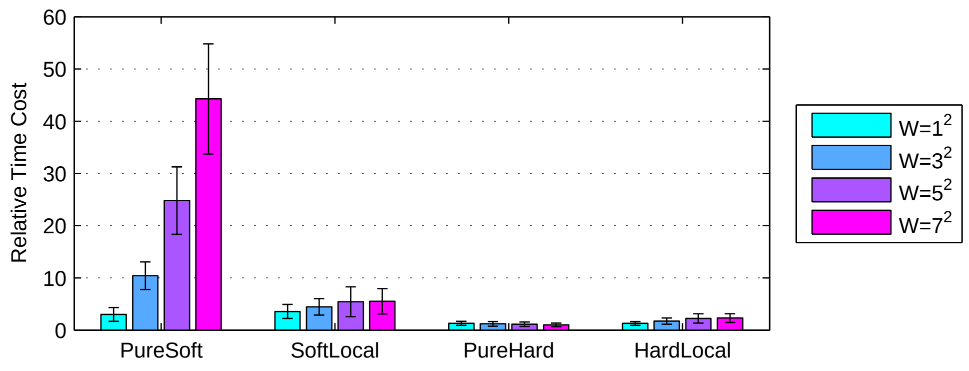


Figure 2-4: *Model Update Computation Costs:* This plot shows the relative length of time required to update the background model after background subtraction has been performed. All times are multiples of the fastest: *pure hard* updates with a neighborhood of size $W = \|\mathbb{N}_p\| = 1$, which is equivalent to the traditional optimized MoG approach. Soft updates are substantially slower than hard ones.

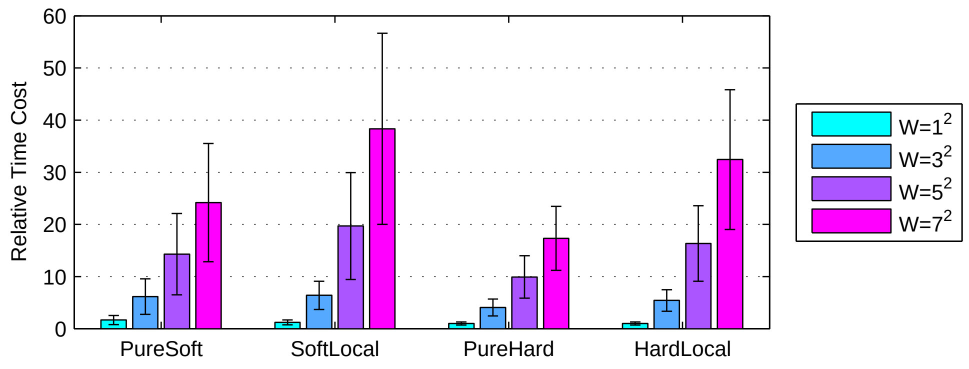


Figure 2-5: *Mahalanobis Map Computation Costs:* This plot shows the relative length of time required to compute the Mahalanobis distance map used as an input to the foreground/background classification (see §2.2.1). All times are multiples of the fastest: *pure hard* updates with a neighborhood of size $W = \|\mathbb{N}_p\| = 1$, which is equivalent to the traditional optimized MoG approach. The algorithm for computing the map is identical in all cases. *Pure hard* is fastest because its update mechanism tends to encourage a more compact model with fewer Gaussian mixture components.

calculations are necessary.

2.3 Experiments

To test our algorithm, we selected several videos from recent publications which attempt to provide better background subtraction in the face of waving trees and/or

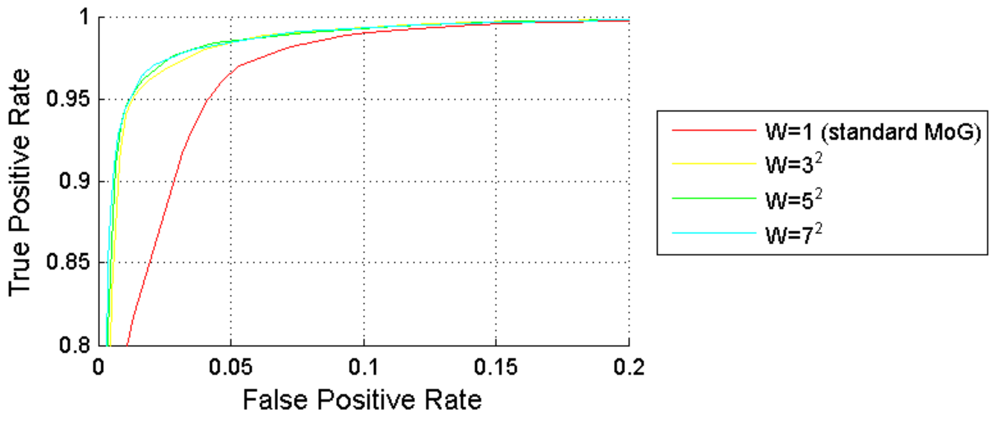


Figure 2-6: *ROC Curves for the Wallflower Dataset:* ROC data points for various neighborhood sizes for the Wallflower waving trees clip. $W = 1$ corresponds to the traditional MoG model.

rippling water. We then hand-labeled all foreground pixels in an evenly-spaced set of frames. Pixels that are ambiguous or are alpha blends of foreground and background are marked as “don’t-care” in our labeling and are ignored in our evaluation. Sample frames from the videos are given in Fig. 2-8.

2.3.1 Pixel-Level Foreground/Background Classification

If we vary the MRF parameters over the course of a collection of experiments and record the per-pixel classification rates, we are sampling points in the receiver-operator characteristics (ROC) curve. We then may estimate the overall ROC characteristics of the system by taking the convex hull of these points [90]. A ROC curve represents the tradeoffs between labeling actual foreground pixels as foreground (characterized the true positive rate) while avoiding incorrectly labeling background pixels as foreground (characterized by the false positive rate). An perfect classifier can be tuned to make no mistakes and it reaches the top left corner of an ROC plot. For a fuller discussion of ROC curves and related ways of evaluating classifier performance, see §A.1.

In Fig. 2-6, we show the ROC curve for a collection of experiments on the Wallflower waving trees video clip [107]. Each curve uses a different neighborhood size. For this clip, our method shows a clear advantage over the traditional MoG in

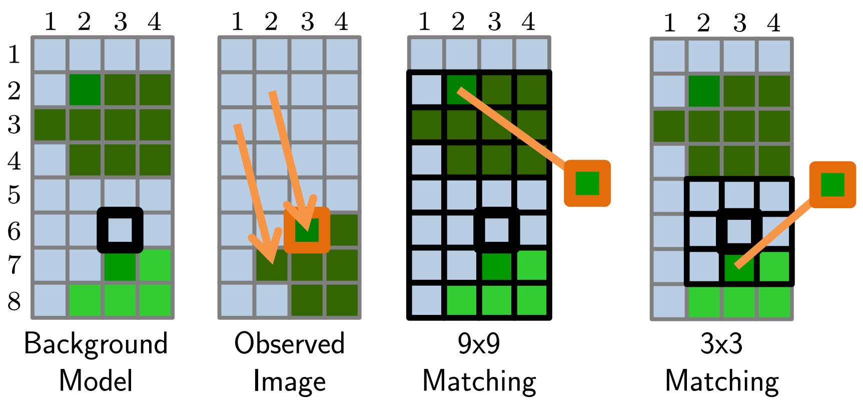


Figure 2-7: *Exploiting Repetitive Texture:* Consider a background model with one Gaussian per pixel learned from a stationary pair of leaves against the blue sky (leftmost image). We will be concentrating on what happens at the pixel location with the bold outline. Suppose a sudden gust of wind moves the top leaf down and right, as shown in the middle-left image. We now wish to find the best matching background Gaussian for the pixel outlined in brown. To match the Gaussian that actually generated it, we would need to have a 9×9 window (middle-right image); however, a smaller 3×3 window allows matching to a similar leaf in the model and labeling the pixel as background. When the dynamic textures are spatially repetitive, windows too small to capture all their movement can still be useful.

foreground detection rates. Using a 3×3 window is sufficient in this case because the tree motion is limited to a few pixels.

For videos where background objects move several pixels between pairs of frames, larger windows can provide additional benefits; however, they are often unnecessary. There is no expected benefit for choosing a window size greater than $W = \max(|d_x|, |d_y|)$ where $(d_x, d_y)^\top$ is the largest expected displacement vector arising from the dynamic texture. Fortunately, many types of dynamic textures are spatially repetitive and allow us to use much smaller windows, as illustrated in Fig. 2-7. For most scenes, we have found that a 3×3 window lowers the false positive rate of foreground detection without unduly raising the false negative rate or becoming too computationally expensive.

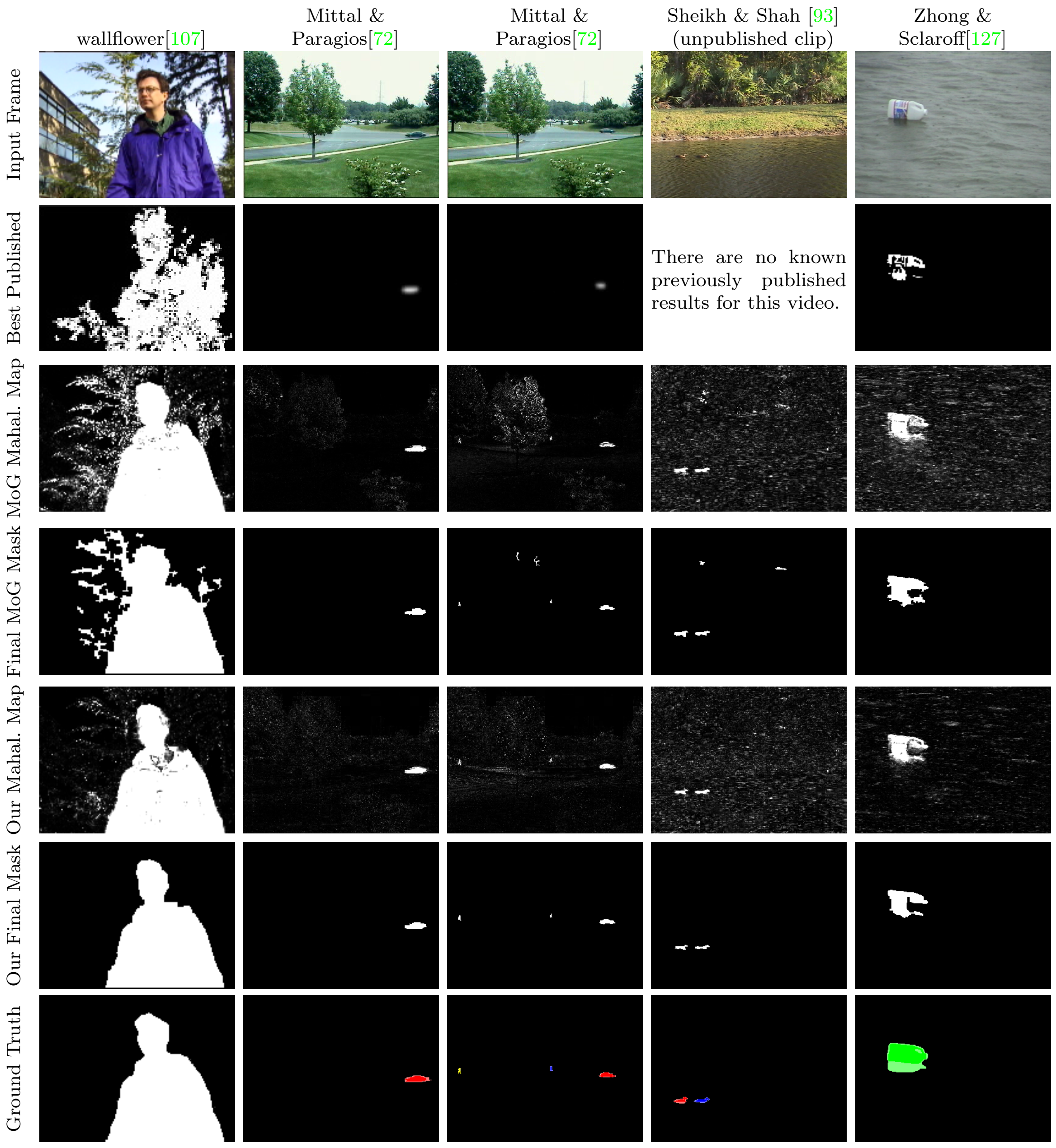


Figure 2-8: *Background Subtraction Results:* Selected background subtraction results comparing the best existing methods with a traditional MoG model and our extended MoG model. Column headings indicate the source of each video. For the Wallflower example, the "best published" result refers to the result from the proposed method in that paper [107]. The "Sheikh and Shah" clip was obtained directly from the authors, though its results were not published in their paper. Refer to the text in §2.3.2 for a discussion of these images.

2.3.2 Experiments on Various Scenes

In Fig. 2-8 we show comparative results from a few selected video frames of our test videos. In the first two rows, we have the input frames and the best known results to date. The third and fifth rows show the Mahalanobis distance to the closest matching appearance model when using a MoG and our model, respectively. The images are thresholded and intensity-scaled for visualization purposes. The fourth and sixth rows are final masks after applying an MRF and morphological operations to the Mahalanobis distance map. The final row contains hand-labeled ground truth where each object is given a different hue and don't-care pixels are shown in a lighter shade. The video clips are ordered from left-to-right as easiest to hardest.

The first column is from the classic wallflower paper by Toyama *et al.* [107]. For 200 frames, the scene is empty and an off-camera person vigorously and continuously shakes the tree, producing a semi-regular dynamic texture. A person enters the scene and a single frame is labeled with ground truth. The "Best Published" results are the best results from the original paper. For the MoG and our method, we used the same parameter settings (except the neighborhood size). When looking at the Mahalanobis distance maps, our approach suppresses the waving trees much more effectively than the traditional MoG and is still able to pick up the person very well.

The second and third columns are from frames 410 and 150 of Mittal and Paragios' traffic video [72] and we present their results in the second row. The graded appearance of these results suggests they have used higher-level modules to detect the vehicles and suppress any noise not corresponding to a vehicle detection. Also note that the detected foreground regions are significantly smaller than the entire vehicle. A MoG model is unable to fully suppress the false positives due to waving foliage, even as we allow its parameters to be optimized independently. Our model is able to suppress all false positives in these two frames and can even correctly detect the two pedestrians in frame 150. In Fig. 2-9, we show overlaid pixel-level and blob level detections in frame 574.

The fourth column is from frame 766 of a very challenging sequence courtesy of

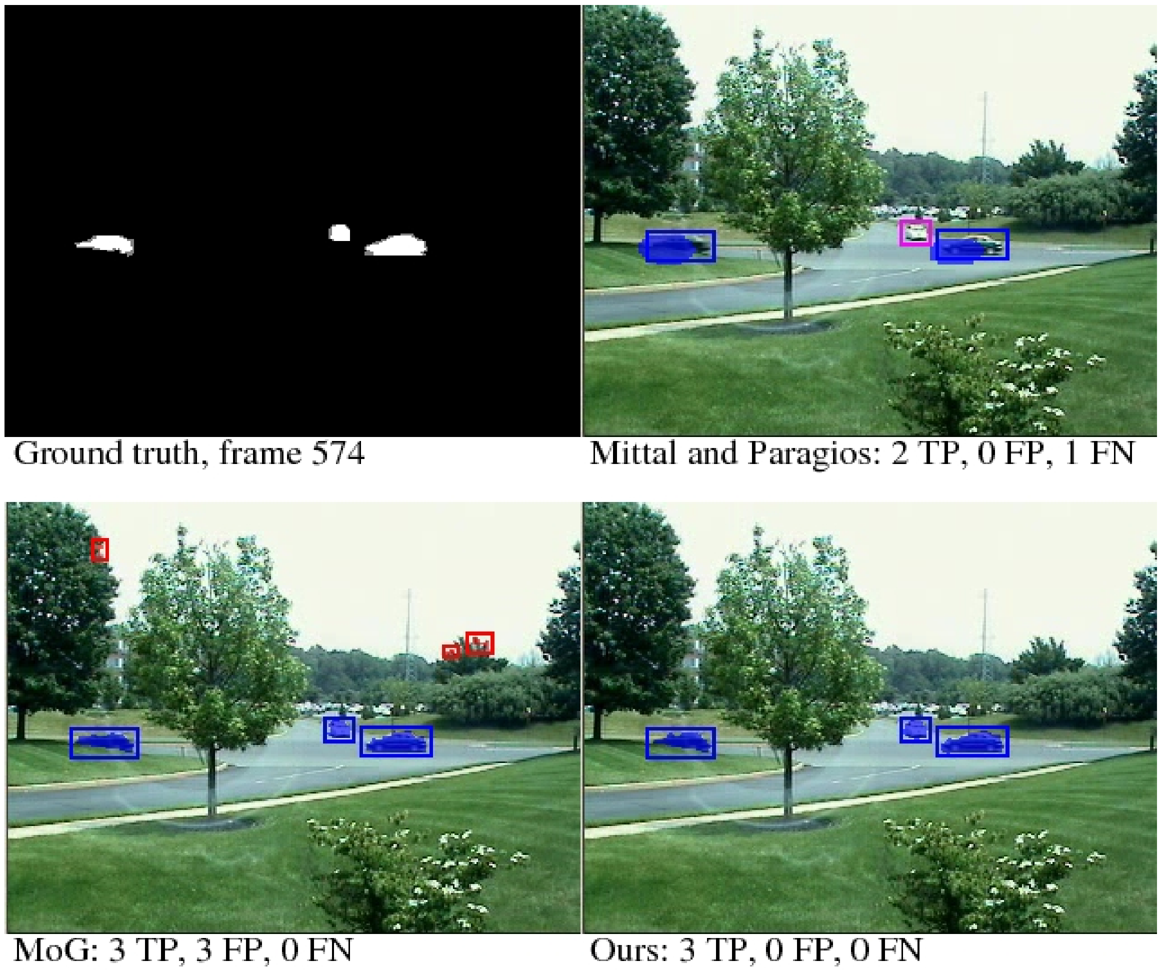


Figure 2-9: *Another Typical Result Image:* All highlighting rectangles are ground-truth positions. Their hue indicates a true positive object detection (blue), false positive (red), or missed detection (magenta). Mittal and Paragios' [72] system misses the car entering from a side street and has localization errors (the blue blob highlights are shifted relative to the ground truth bounding boxes). A tuned MoG model has false positives in the trees. Our model suppresses the false positives, captures the third car, and has better localization.

Sheikh and Shah. We are unaware of any existing published results on the sequence. The scene consists of rippling water, large-leafed tropical plants blowing in the wind, and two small ducks swimming by that are very close in color to the water. Our model is able to produce cleaner silhouettes and have fewer blob-level mistakes than the best MoG results.

The final column is from frame 36 of a Zhong and Sclaroff dynamic texture clip [127]. Our foreground detections are cleaner and more fully capture the bobbing jug.

In practice, we have found the performance of our method to be consistent with these results as it has been employed in traffic surveillance, indoor activity monitoring, and coastal ship tracking (with mild to moderate waves). For more challenging water scenery or places with very consistent dynamic textures, we have found the usage of optical flow and/or spatio-temporal derivatives to be useful as additional modeling features.

2.4 Summary

In this chapter, we have introduced a new image generation model that takes into account the spatial uncertainty of dynamic background textures. Our model is much more compact than recent methods ([127],[93],[72]) that have been introduced to handle this problem. Ours can be readily implemented in the familiar mixture of Gaussians framework and it performs better than competing methods. It is most useful at suppressing clutter from waving trees, but it also can be used for suppressing false positives arising from rippling water.

This chapter has described a specific example of finding where an existing model does not explicitly account for particular types of uncertainty and then expanding that model to make it explicit. Here we recognized that existing background subtraction models do not adequately account for local spatial perturbations in background textures such as leaves that blow in the wind. In other vision problems, it is common to have performance limited by simplifying modeling assumptions. There are often straightforward ways of relaxing those assumptions to deal with real-world issues.

Relating this back to the problem at hand, a single Gaussian model does not work well when there is a multimodal color distribution due to lighting changes and/or temporally regular dynamic textures. A Mixture of Gaussians (MoG) model helps address those issues, but we saw that a MoG model breaks down when the motion in a dynamic texture is infrequent enough that the mixture components are not learned well enough. Our MoG enhancements address this issue by explicitly allowing the data to match mixture components in a local window.

Throughout this thesis, we will continue to explore aspects of background subtraction. In §3 we present a dataset where a hundreds of simultaneously visible people, lighting issues, and crowds will challenge background subtraction enough that we resort to using a strong foreground object model instead. In §5, we will be able to return to using background subtraction to extract silhouettes for biometric purposes. In §6, we will combine background subtraction techniques with a foreground appearance model to help detect events in the presence of prevalent occlusion and lighting changes.

Chapter 3

Efficient Pedestrian Detection for Scene Activity Modeling

Portions of this chapter contains previously unpublished joint work with Jim Sukha, Krista Ehinger, and Geza Kovacs.

3.1 Introduction

In §1, we discussed the value of detecting objects of interest in a visual scene: given the locations of all these objects, we can perform higher-level functions such as event analysis (§6) and recognizing individual people (§5). Especially when moving objects are well-separated in space, model-free bottom-up approaches can be robust and computationally efficient (§2). Unfortunately, in some scenarios the objects of interest densely populate the scene. In this chapter, we will discuss alternative detection mechanisms and apply them to scenes densely populated with foreground objects.

Here we will be using a collection of videos taken of a concourse at a major transportation hub, the Great Hall at Grand Central Terminal in New York City. Typical video frames are depicted in Fig. 3-1. We note that the observed floor area in these videos is substantially larger than those typically used in academic research such as the videos we will use in §6, while containing a similar density of people (see

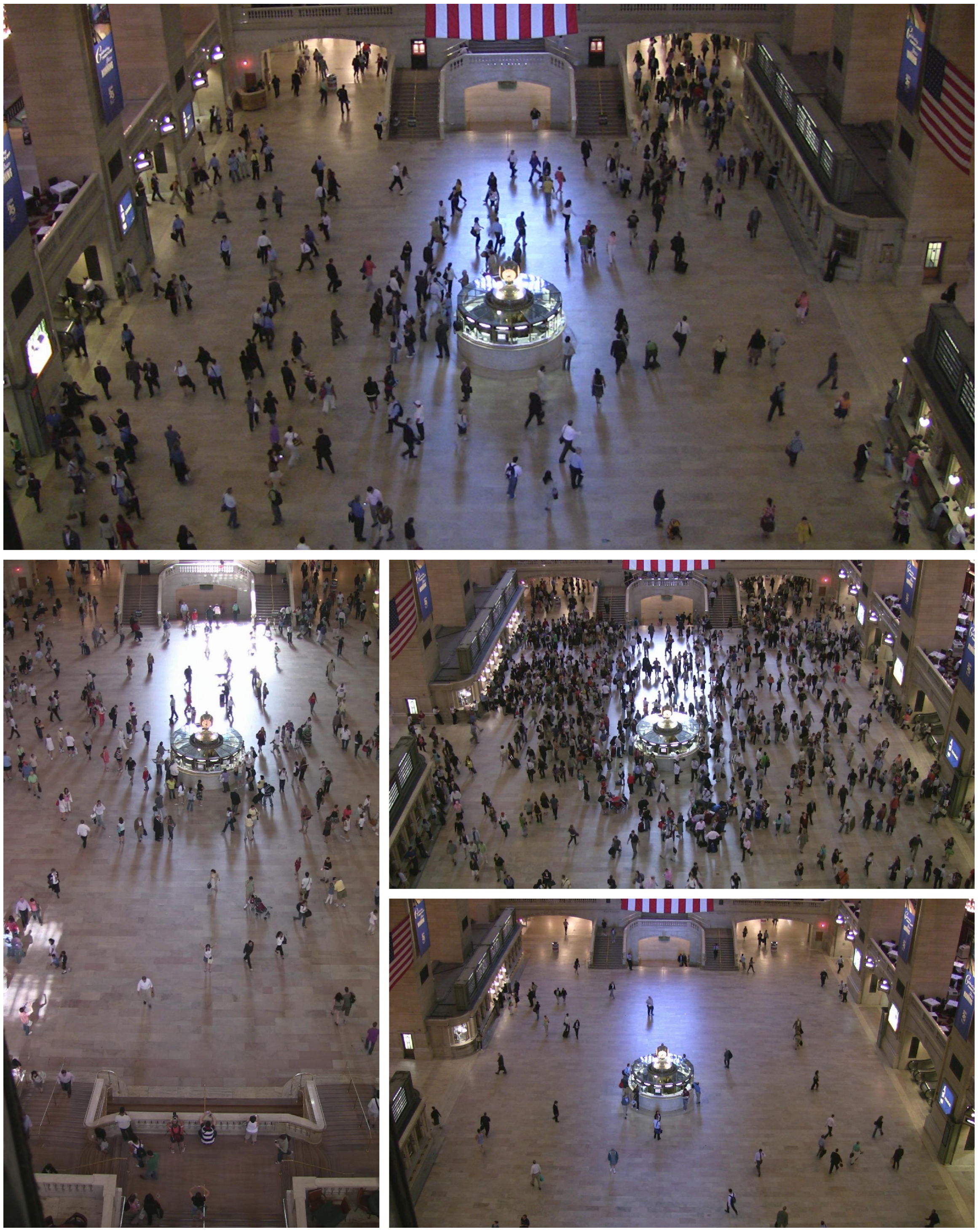


Figure 3-1: *Concourse at a Transportation Hub:* The top image shows typical conditions for our detection experiments in terms of lighting conditions and number of pedestrians (267 in this case). The leftmost image shows an alternative viewpoint with more extreme lighting conditions. At times, the concourse can be very heavily populated (middle right image) or sparsely populated (lower right image).

Statistic	Airport Hallway (Fig. 6-1)	Grand Central (Fig. 3-1)	Units
mean population	4.8	168.6	people/frame
floor area	36	1600	m^2
spatial density	13.5	10.5	people/$(10m)^2$
non-masked pixels	2.6×10^5	1.17×10^6	pixels
image density	18.2	144.6	people/$(1000px)^2$

Table 3.1: *Population Statistics for Some Dense Scenes:* The airport hallway clips from §6 have a similar number of people per m^2 as the concourse studied in this chapter, but the concourse views cover a much larger area with individual people occupying fewer image pixels. These concourse statistics were gathered from 37 randomly selected frames taken from two different camcorders during a single morning rush hour with moderate scene density (see Fig. 3-2 for typical images from this set).

Tab. 3.1). Although these videos were taken with a higher resolution camera[1], the standoff distance is much larger, so individual pedestrians occupy approximately 8 times less pixel area.

To emphasize the difficulty of using a blob-based tracker, we have hand-labeled bounding boxes for every person in one frame in Fig. 3-2. Note that the vast majority of bounding boxes have significant overlap with other boxes.

The remainder of this chapter is organized as follows:

- In §3.2, we will demonstrate the difficulties with using background subtraction on these videos.
- In §3.3, we show improved classification results with naïve feature point detection.
- We then discuss the Dalal and Triggs [19] pedestrian detector in §3.4 and our efficient implementation that uses commodity graphics card hardware.
- We summarize in §3.5.

[1]Unless otherwise noted, the Grand Central videos were recorded at 1440×1080 in progressive scan at 23.976 or 29.97 frames per second with a pixel aspect ratio of 1.33 : 1, using Canon HG10 and Canon HV30 camcorders. All images in this chapter have had their aspect ratios corrected. The standard definition (720×576) recordings we will see in §6 are interlaced at 25 frames per second, have square pixels, and were captured with a Canon MV-1.

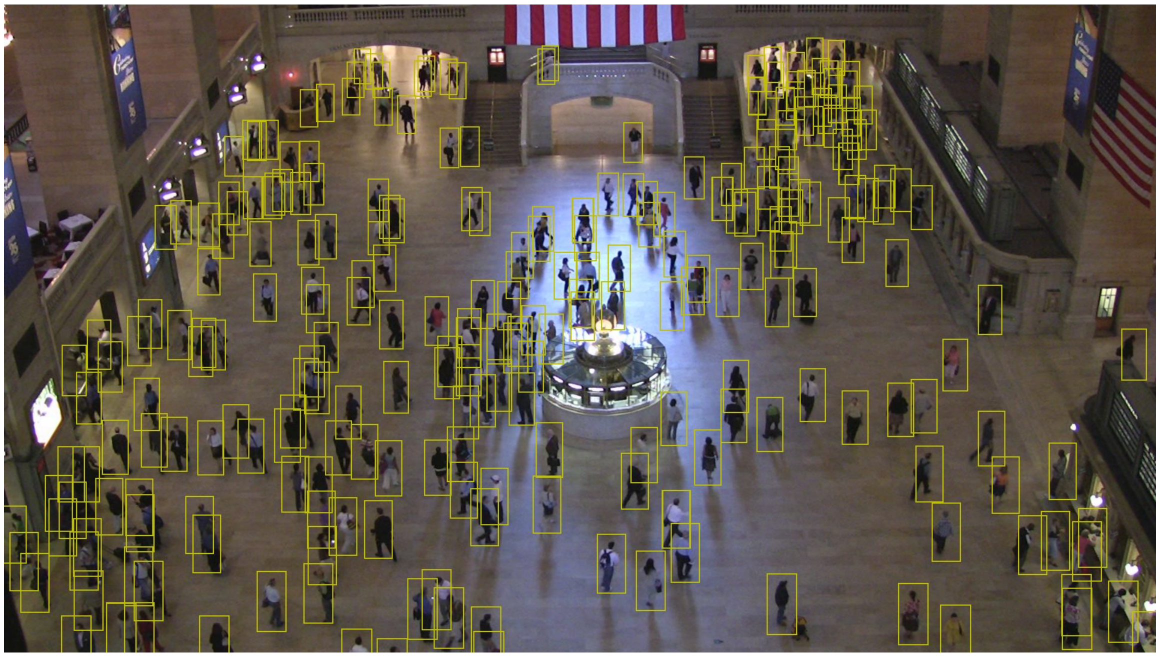


Figure 3-2: *Hand-labeled Locations of All the Pedestrians:* In this frame, there are a total of 267 pedestrians.

3.2 Challenges Using Background Subtraction

As suggested in chapters 1 and 2 and as we will explore again in chapters 5 and 6, one traditional approach to detecting people in these images would be to (1) use background subtraction, (2) group foreground pixels using connected-components analysis, then (3) consider each extracted blob to be a pedestrian.

We have seen that this approach can work well in sparse scenes, but it becomes more challenging to get good results in dense scenes. Dense scenes are more difficult because there is a high likelihood that two real-world objects will be close enough to each other in image space that their blobs get merged (as in Fig. 3-3). One must either (a) find a way to segment these blobs, or (b) try to detect when a person is isolated and only treat their blob as a pedestrian detection at those times, as we will do in §6. Non-ideal lighting conditions only make this problem more challenging because they make the background subtraction process less discriminative. The user is forced to trade off having (1) isolated but over-segmented blobs versus (2) merged blobs with a low per-pixel miss rate. In this section, we will temporarily sidestep the

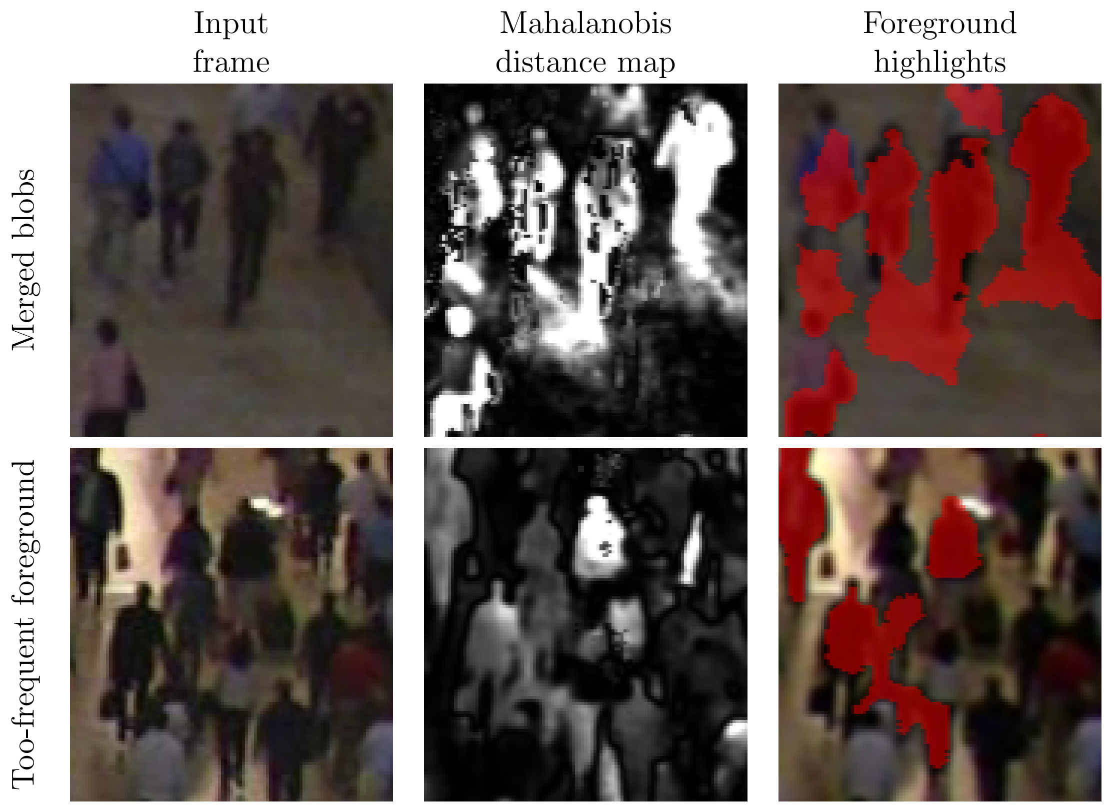


Figure 3-3: *Typical Background Subtraction Failures:* These image chips were extracted from background subtraction results for the frame in Fig. 3-2. In the top row, we see that the blobs from the three people in the top left get merged. The leftmost person with a blue shirt has an extra blob that includes his feet, his shadow, and the head of the woman below him with a pink blouse. The bottom row was extracted from the passageway near the top right of Fig. 3-2. In addition to merged blobs, most people are missed because pedestrians are so frequent in that area that no good background model is learned.

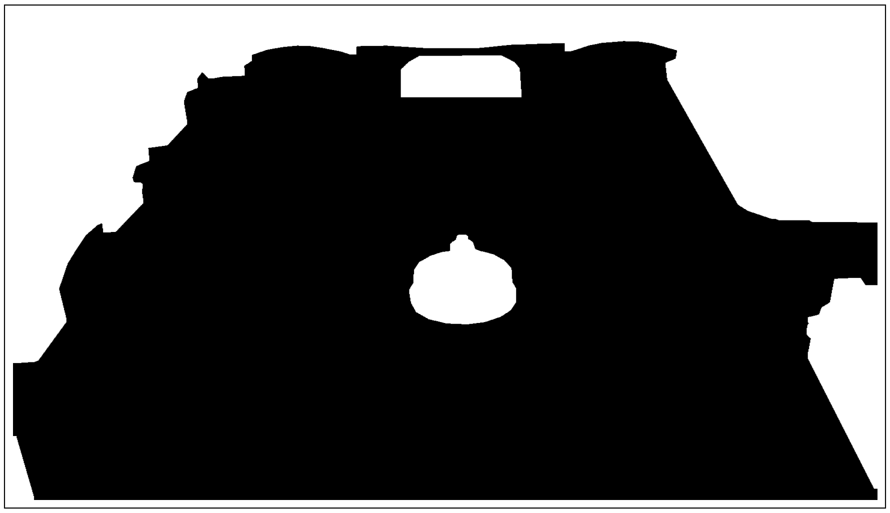


Figure 3-4: *Don't Care Mask:* We have created a don't-care mask for each video clip, this one corresponding to the video depicted in Fig. 3-2. The mask helps our background subtraction algorithm avoid false positives due to fluttering flags, diffuse reflections on the structural pillars, and registration changes by lighted signs (§3.2). For the HOG-based detector, it helps avoid persistent false positives in places like the balconies and banister columns (§3.4).

segmentation and grouping issues and show that even at a pixel level, it is difficult to achieve acceptable performance using background subtraction.

Experiment Notes: User-Supplied Mask and Ground Plane Registration

Before proceeding, we note that in this chapter, we will assume that the user has supplied an approximate mask indicating at which pixel locations a pedestrian can appear. For example, Fig. 3-4 shows the mask used for the video depicted in Fig. 3-2. Except where otherwise noted, we ignore detections and ground truth annotations lying on white areas of this mask. These masks were created with minimal human effort and we currently make no attempt to register the mask with the video (other than with the single frame used to create it). Frame-to-frame registration is an open research problem whose difficulty is exacerbated here by the large number of independently-moving objects. Correct frame-by-frame registration of the mask would likely improve the results for all approaches discussed in this chapter; however, based on our qualitative examination of the results, we do not believe that registration would change any of the conclusions we make in this chapter.

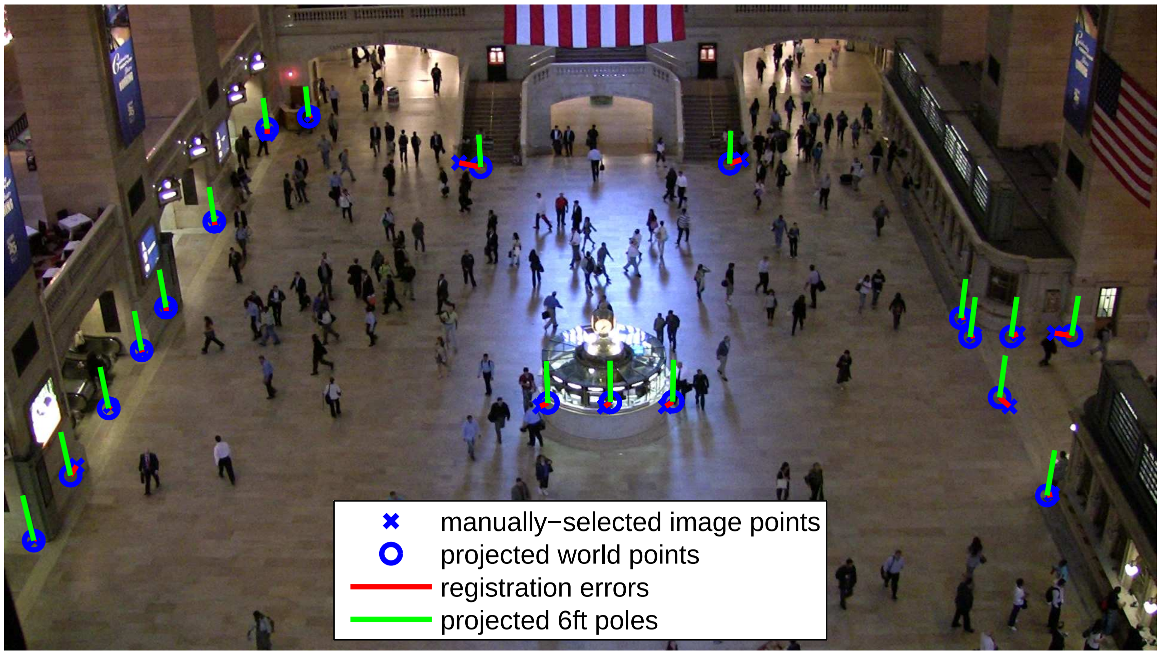


Figure 3-5: *Ground Plane Registration:* For the Grand Central video, we have manually created a ground plane registration for a single frame. Here we show the manually selected image points (blue $\times$'s) and lines (red) connected them to projections of their real world coordinates (blue $\circ$'s) given the registration. For reference purposes, we also show the projection of a 6 foot tall pole at each of these locations (green lines). The greatest registration errors occur on the stairway (where the world coordinates are less accurate) and the entryway center on the right (where the image coordinates are approximate).

In discussing registration, we also note that we have manually measured the relative locations of several points on the ground plane, allowing us to register against it, as seen in Fig. 3-5. This ground plane registration will be used in §3.4 and later sections of this chapter to improve the speed of and decrease the false positive rate of a strong model pedestrian detector[2]. In a full system, one might consider an automatic registration mechanism such as that of Hoiem, Efros, and Herbert [46].

Background Subtraction

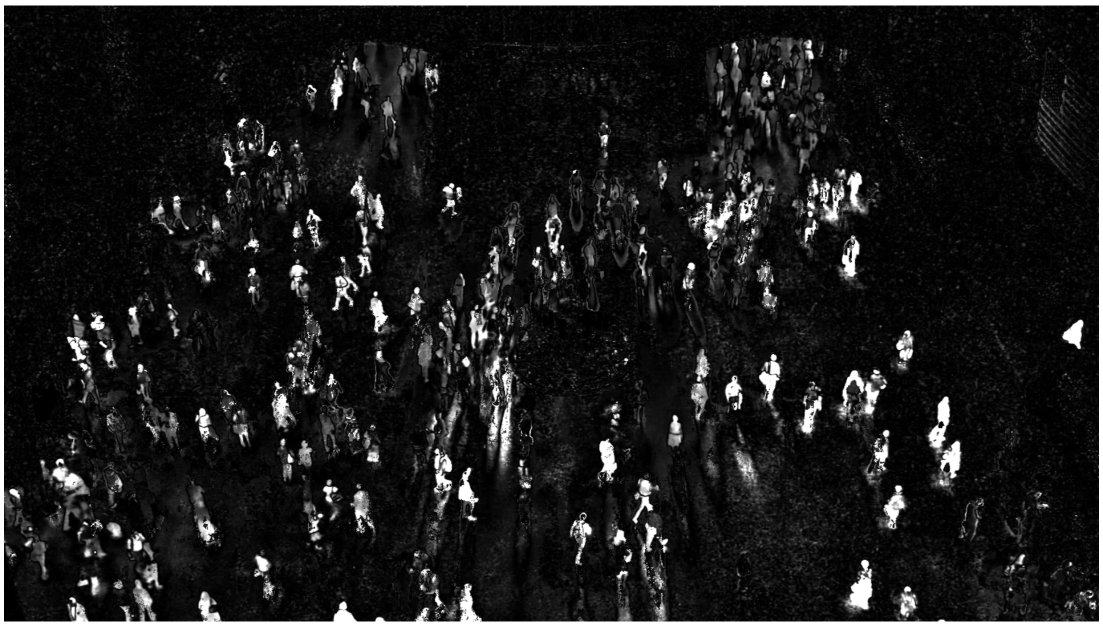
To test the utility of background subtraction in this environment, we use a Stauffer and Grimson-style [101] Mixture of Gaussians (MoG) to model the per-pixel background color distribution, as has been discussed in §1.3.1 and §2. Because the Grand Central scene is indoors and there is little motion of the background, it was not necessary to apply the neighborhood search MoG extensions of §2.

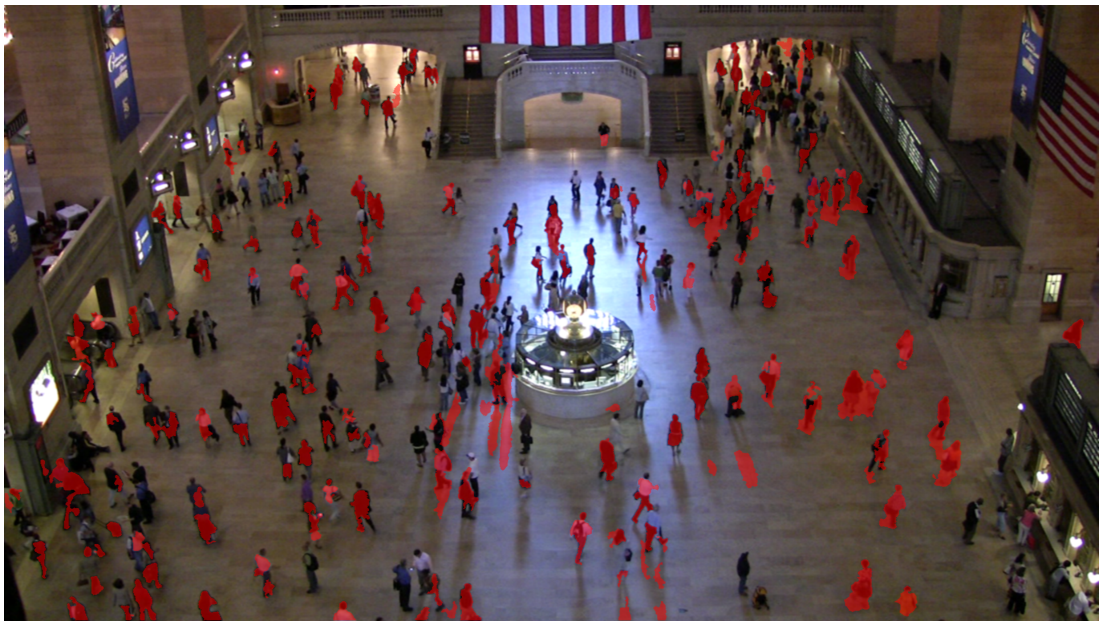
With some tuning, background subtraction is able to yield results typified in Fig. 3-6(b), where bright red highlights indicate foreground pixels. Although most pedestrians have at least some of their pixels highlighted, there are significant errors that would make it difficult to accurately track a large percentage of them. For example, some of the more notable errors are:

- *Merged blobs:* On the right side of the image, there are several blobs that correspond to multiple nearby pedestrians.
- *Shadows:* Especially in the lower center of the image, there are a number of cast shadows that are misclassified as foreground.
- *Fragmentation:* Throughout the scene, it is common for a person to have multiple disconnected blobs[3].

[2] For the speed tests in §3.4.5, both the CPU and GPU versions use the ground plane registration, thus the speedup numbers are fair.

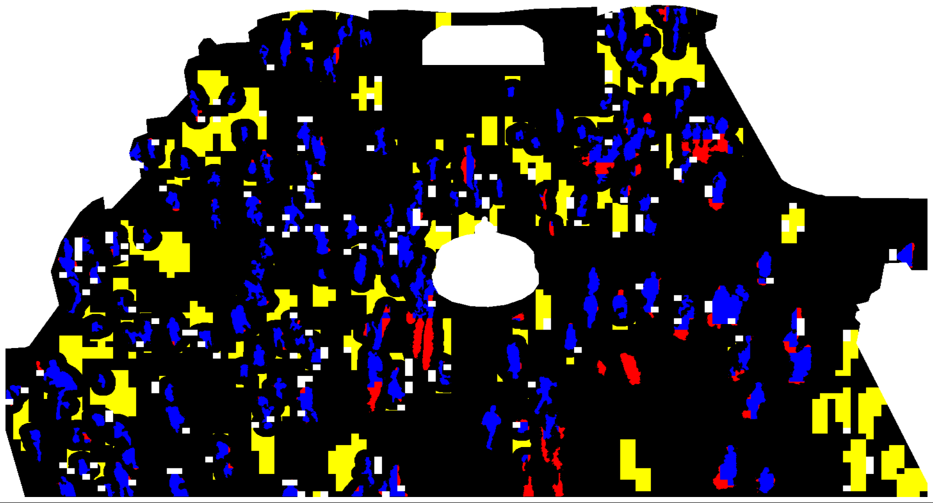
[3] As will be discussed in §6, we can trade off merged blobs with fragmentation artifacts by adjusting the MRF parameters. The parts of the Mahalanobis distance map (Fig. 3-6(a)) that sit between blob fragments often have a low estimated foreground likelihood (dark shading). If the MRF were biased sufficiently to join the fragments, there would be significant false merging of blobs throughout the scene (observe similar gray levels sitting between people).


(a) Mahalanobis Distance Map


🟥 foreground pixels

(b) Final Foreground Map as Highlights

Figure 3-6: *Background Subtraction Results:* Fig. 3-6(a) shows the Mahalanobis distance map for Fig. 3-2 (without the mask from Fig. 3-4 applied). We use an MRF to produce a spatiotemporally-smoothed foreground/background segmentation, shown as red highlights in Fig. 3-6(b).



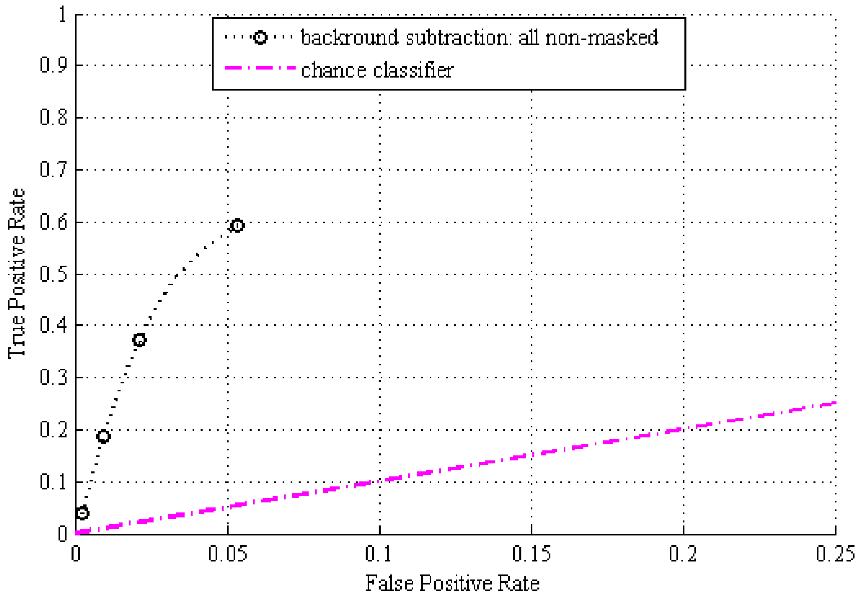
🟦 true positives 🟥 false positives 🟨 false negatives ⬛ true negatives ⬜ don't care

Figure 3-7: *Block-wise Background Subtraction Results:* Here we visualize how the background subtraction results shown in Fig. 3-6 are used to generate the statistics to create Fig. 3-8 (along with the results from other frames). Ground truth labels were made on 16×12 pixel blocks. To better visualize the results, pixels from pedestrian-labeled blocks have been turned to black if the distance to the nearest foreground detection is less than 1 block (hence the precise boundaries seen on the yellow labels). There are many missed regions and shadows cause many false positive foreground detections.

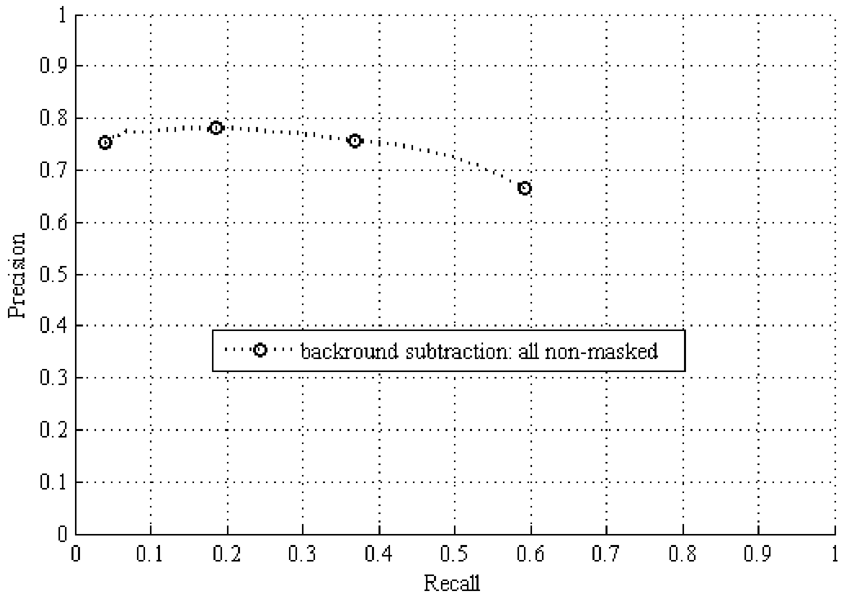
- *Glare:* In other video frames, glare above and below the information booth in the center of the image causes image saturation, reducing the discriminability of the background model.

Block-wise Analysis

To evaluate background subtraction, we randomly sampled 12 frames from the 1920×1080 video depicted in Fig. 3-2 and hand-labeled each 16×12 block of pixels as pedestrian, non-pedestrian, or don't-care. If a block had a pedestrian taking up more than about 10% of its area, the whole block was labeled as containing a pedestrian. Don't-care labels were reserved for blocks where the human annotator could not accurately judge the occupancy. Per-pixel labels were not used because they would have introduced significant numbers of additional don't-care labels and would have consumed an inordinate amount of annotation time.



(a) Background Subtraction ROC Curve



(b) Background Subtraction Precision-Recall Curve

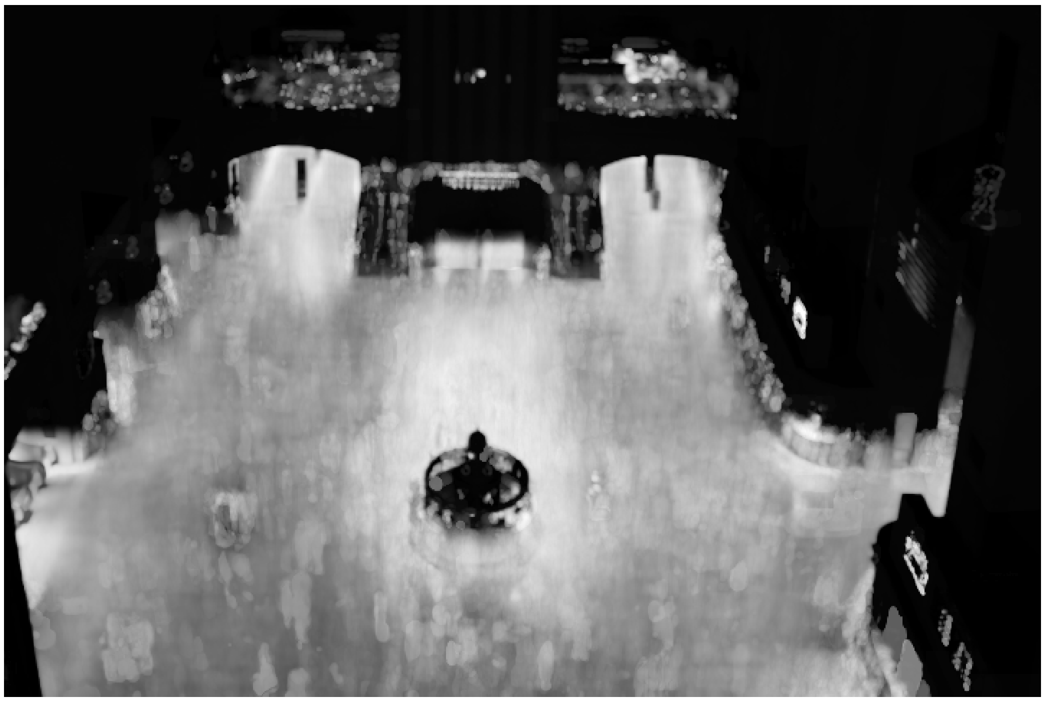
Figure 3-8: *Foreground Detection Curves:* These curves were produced by varying the required number of pixels within each 16×12 block that need to be detected as foreground before labeling the block as belonging to a pedestrian. There is substantial room for improvement.

In our evaluation, we choose some threshold τ, and if the number of detected foreground pixels in a block is greater than τ, the block is considered foreground and ideally the ground truth labeling is a pedestrian. If we vary τ from 0 to $16 \cdot 12 = 192$, we can produce a Receiver Operator Characteristic (ROC) curve (Fig. 3-8(a)) and a corresponding precision-recall curve (Fig. 3-8(b))[4]. Precision and recall are another pair of statistics that can be used for evaluating detector tradeoffs. Precision measures the fraction of detections that are true positives, and recall measures the fraction of positive examples that are detected. An optimal detector has a precision-recall curve that touches the top-right corner (*i.e.* 100% precision and 100% recall). Although the results in Fig. 3-8(a) are significantly above chance, a non-degenerate classifier cannot even achieve 80% precision. 70% precision can only be achieved at less than 55% recall.

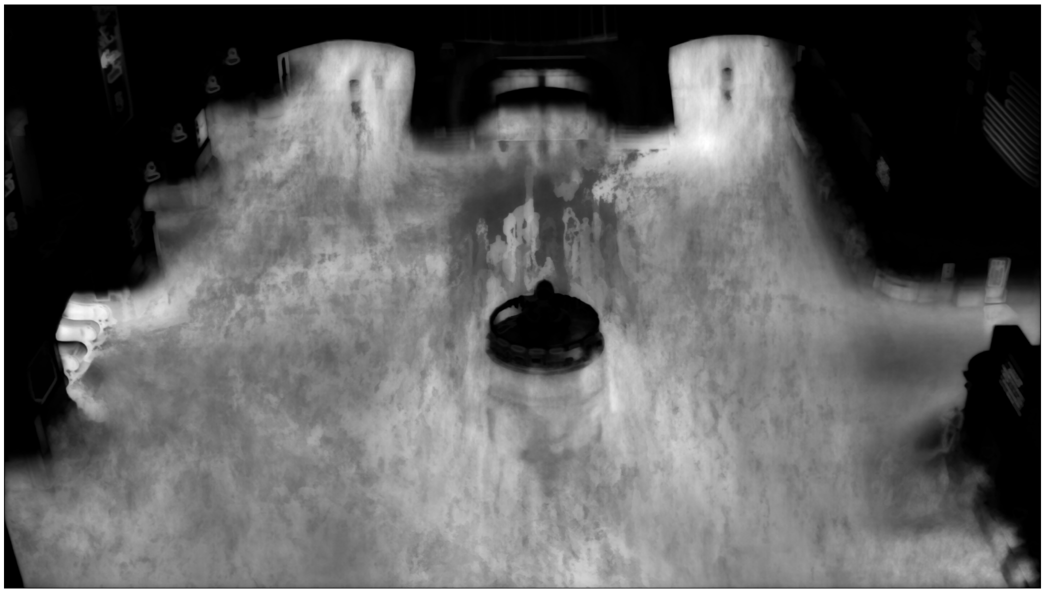
If we examine the results across the whole video clip, it becomes apparent that many of the background subtraction errors are systematic and are concentrated in different image regions. For example, in Fig. 3-9, we show the per-pixel foreground likelihood for two videos of the Great Hall. In both videos, there is substantial foot traffic in the area between the information booth (in the image center) and the archway on the far side of the image. On an overcast autumn afternoon, the foreground is correctly and frequently detected in that region (notice the bright white area between the information booth and archway in Fig. 3-9(a)). Unfortunately, severe glare on sunny summer mornings lowers the chances that background subtraction will detect pedestrians there (notice the dark areas between the same part of Fig. 3-9(b)).

To investigate the performance of background subtraction further, we manually identified three non-contiguous areas of the scene (see Fig. 3-10, where the camera faces east). Along the north and south (left and right, respectively) sides of the floor, the lighting is mostly diffuse resulting in little glare and very mild shadows (green regions). High glare east of the information booth tends to lower the true positive

[4]Scott, Niranjan, and Prager [90] showed that ROC curves are monotonic, that one can take the convex hull of a set of ROC points, and that one can always create a classifier whose performance sits along a straight line between two other points in ROC space. Davis and Goadrich [25] provide a description of various ways to correctly find the feasible set of precision-recall points and how to interpolate between them in a precision-recall plot.



(a) Accumulated Foreground (Overcast Afternoon)



(b) Accumulated Foreground (Sunny Morning)

Figure 3-9: *Lighting Conditions and Background Subtraction:* In Fig. 3-9(a), we visualize the likelihood that a given pixel in a video will be labeled as foreground. Lighter shading corresponds to a higher likelihood of a foreground pixel being detected at that location. The likelihood is estimated over a 1 hour video, recorded on an autumn afternoon with overcast skies. In Fig. 3-9(b), we visualize a similar likelihood measure for the same scene, but on a summer morning with clear skies, using a different camcorder.

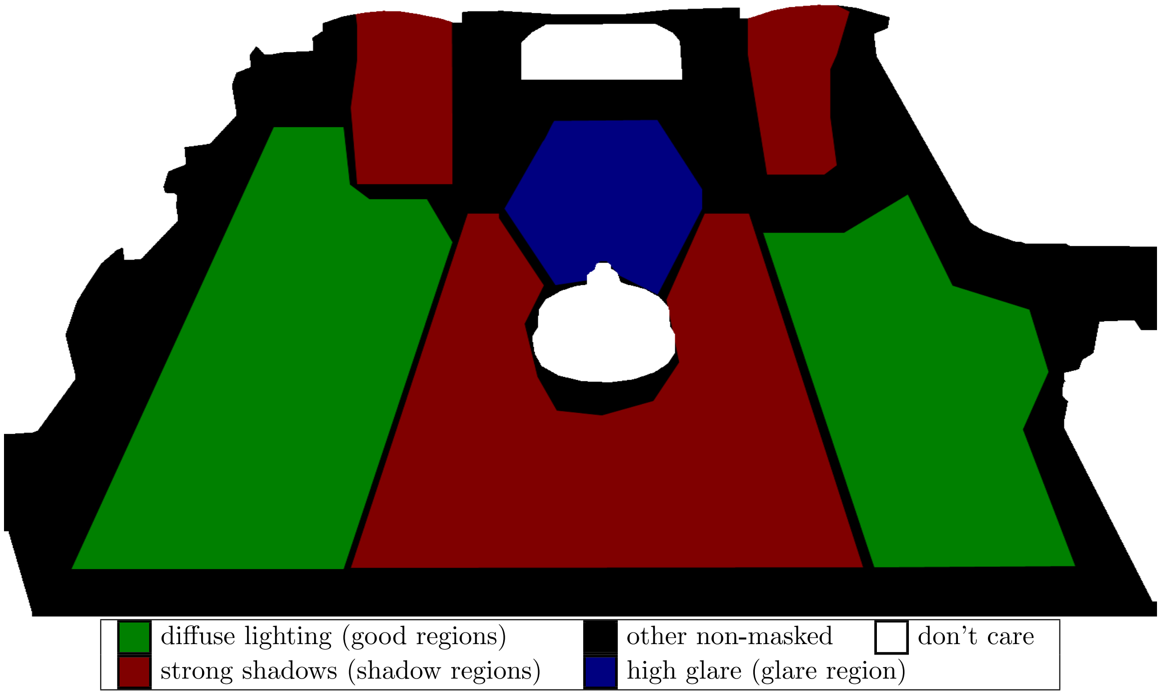
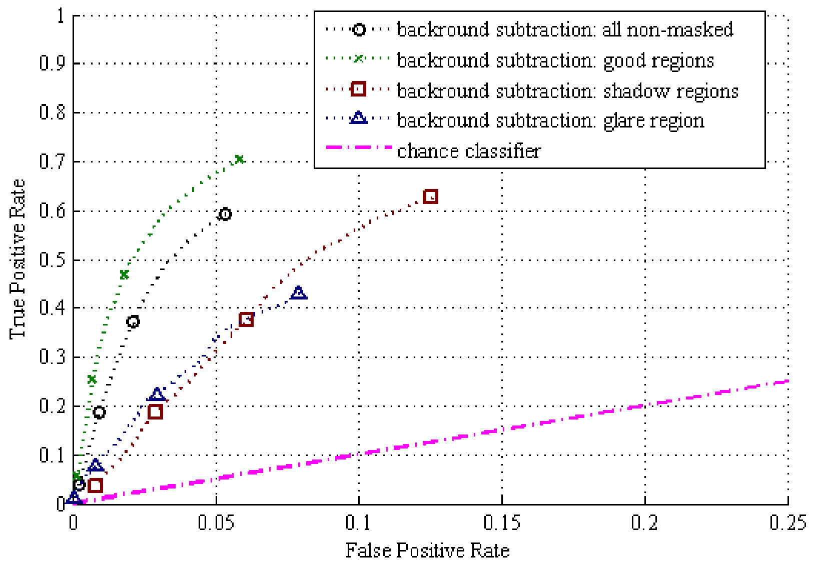

🟩 diffuse lighting (good regions)
🟥 strong shadows (shadow regions)
■ other non-masked
🟦 high glare (glare region)
□ don't care

Figure 3-10: *Qualitative Regions:* Manually-chosen regions used when investigating the performance of background subtraction.

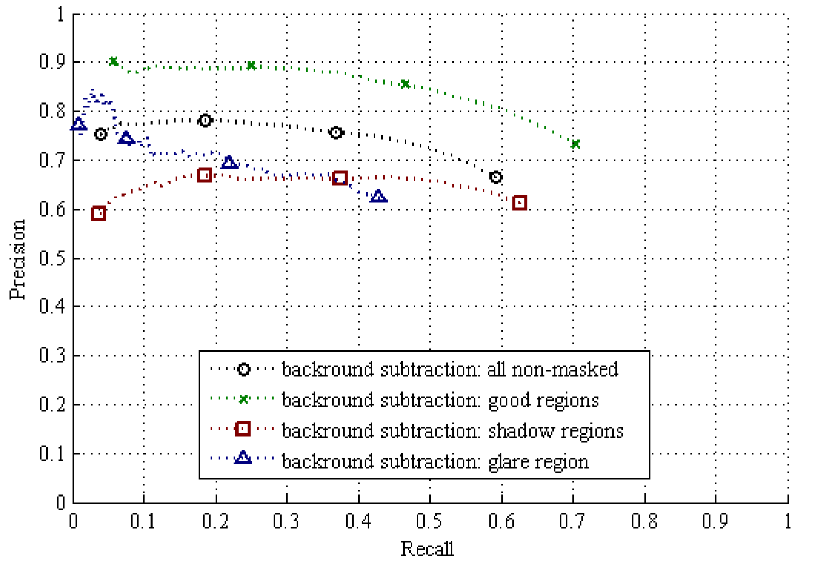
rate of foreground detections (blue region), and deep cast shadows in other areas raise the false positive rate (red regions). Black areas in Fig. 3-10 are only considered when evaluating results from all regions. In Fig. 3-11, we see that in the regions with diffuse lighting ("good regions"), performance improves in terms of the true positive rate and precision. In the shadow and glare regions, performance is notably worse than in the good regions or across the whole image.

Background Subtraction Summary

In this section (§3.2), we have seen that we can detect foreground pixels on many pedestrians using background subtraction. It is a general method with a weak model that requires very little tuning or training. Unfortunately the results are far from ideal. We would especially like to improve the results in difficult image regions that have glare and shadows. Further, we have been ignoring the difficult problem of merging fragmented blobs and segmenting merged blobs to extract individual pedestrians from the foreground mask so they can be tracked.



(a) Regional Background Subtraction ROC Curves



(b) Regional Background Subtraction Precision-Recall Curves

Figure 3-11: *Foreground Detection Curves by Region:* Here we expand on the results shown in Fig. 3-8, showing the ROC and precision-recall curves conditioned on blocks belonging to each of the image regions depicted in Fig. 3-10. The "good" regions have the best results because the lighting conditions are best there. Results are worst in the "shadow" and "glare" regions where strong direct light and shiny floors present challenges for background subtraction. Even if we only wished to analyze the "good" regions, these results are poor enough a non-specialized tracker built on top of them (like the solution we will explore in §6) will likely have a prohibitively high error rate.

3.3 Naïve Feature Point Detection

In examining the Grand Central scene (see Fig.3-1), we note that the floor has very little texture. Pedestrians on the other hand tend to induce intensity edges and corners. As an alternative to using background subtraction for detecting people, we briefly consider a system that tries to detect corners and other features that can be easily tracked. This approach is only suitable for scenes where the background is textureless and we use it primarily as a secondary performance baseline against which we will compare a strong pedestrian detection model in §3.4.

This second system uses a detector to find small local features that are suitable for tracking. In particular, we use the `GoodFeaturesToTrack` function from the OpenCV library [78] that implements the Shi and Tomasi feature point detector [94]. We run the algorithm independently on each of the annotated test frames. Though more costly than background subtraction, detecting and tracking points with this algorithm is 1–2 orders of magnitude faster than state-of-the-art strong model detectors such as Dalal and Triggs' that we will discuss in §3.4 (when both are implemented on a CPU). Like background subtraction, no training is needed and little parameter tuning is necessary.

Algorithm 3.1 *Classification Evaluation Algorithm:* We use this algorithm for evaluating how well feature point detection does at classifying blocks of pixels as pedestrians or not. We assume that a block with at least one detected feature point is classified as belong to a pedestrian. The issue of clustering detections to find individual pedestrians is sidestepped.

```
1: for all minimum detector point qualities, $q_{min} \in [10^{-4}, 10^{-1}]$ do
2:     for all minimum distances between detected points, $d_{min} \in \{1, 2, ..., 10\}$ do
3:         for all test images, $I$ do
4:             Acquire detections using cvGoodFeaturesToTrack($I$, $q_{min}$, $d_{min}$).
5:             Discard masked detections.
6:             Discard redundant detections in each $16 \times 12$ annotated block.
7:         end for
8:         Gather ROC and precision-recall statistics for each $16 \times 12$ block.
9:     end for
10: end for
11: Compute the ROC envelope and feasible set of precision recall points from all
    experiments.
```

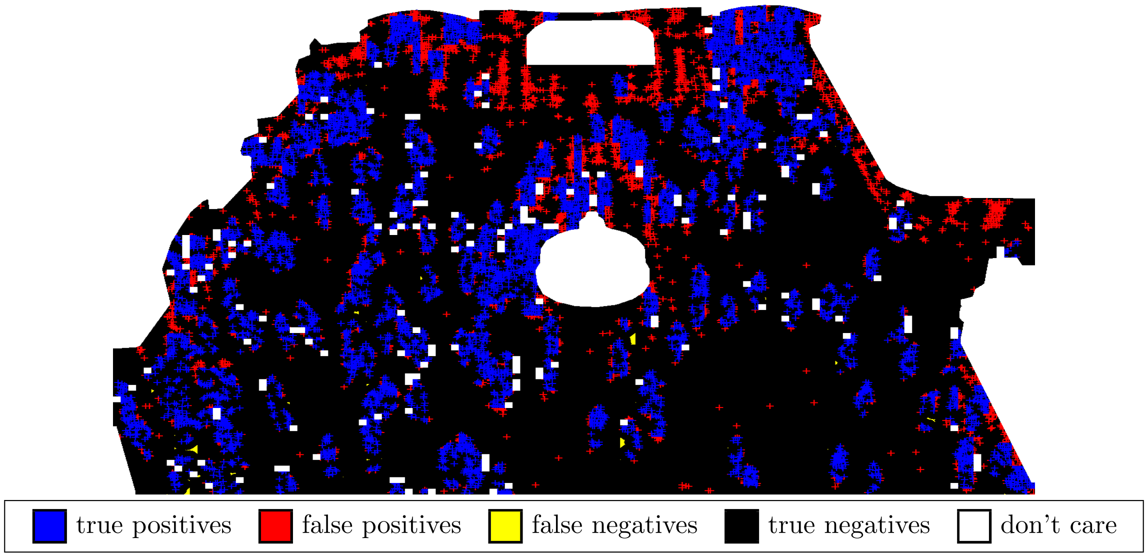
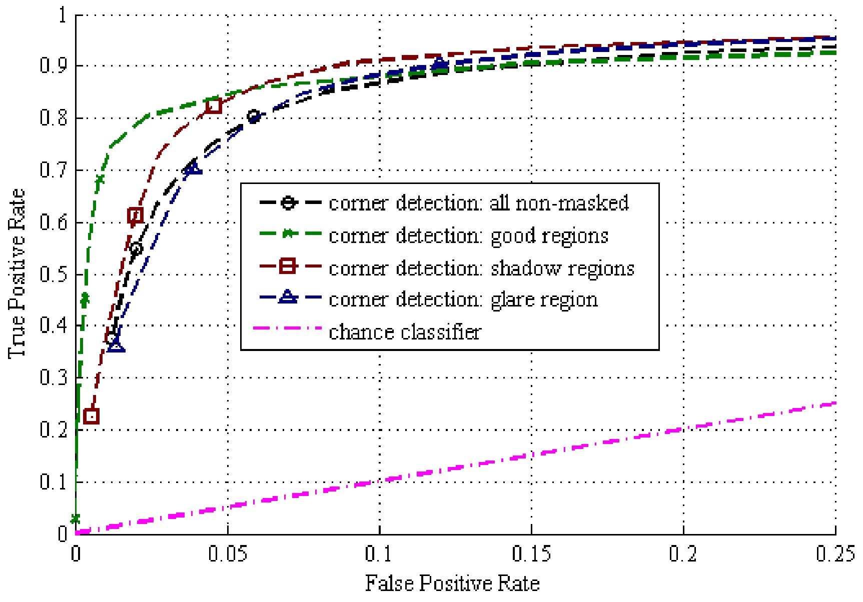


Figure 3-12: *Block-wise Corner Detection Results:* Here we show the feature point detections as + signs on a sample test frame, depicted in a manner analogous to Fig. 3-7. With the parameter settings chosen here, there are nearly no false negatives (yellow areas far from any detection), but there are many false positives (red + signs). These false positives occur due to shadows, compression artifacts, and most importantly, building geometry.

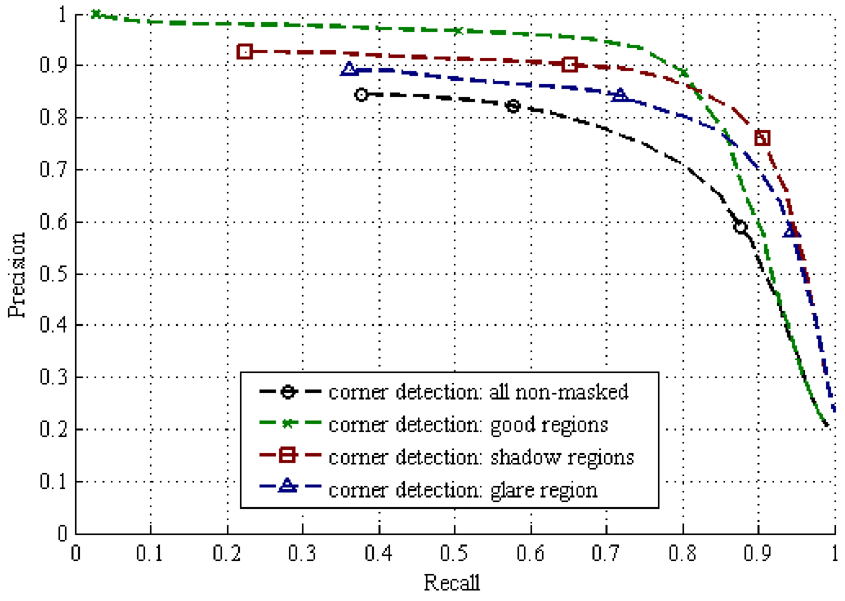
As with fragmentation in background subtraction, we ignore the problems of grouping point detections and tracking them over time for these experiments. We also use the mask (Fig. 3-4) from the background subtraction experiments. Our test procedure is given in Alg. 3.1.

In Fig. 3-13 we see much better results than we did with background subtraction (Fig. 3-11). As expected, the "good regions" have the best results, but it is interesting that "all non-masked" results are the worst. This is because it includes non-floor parts of the image like the stairs and ticket counters that have high false positive rates, as seen in Fig. 3-12. Although we could try to mask out those areas too, we'd miss the many people who walk in front of those areas, which are semantically meaningful (loiterers, people purchasing tickets, etc.).

In §4, we will see that tracking these feature points over time will allow us to do some modeling of scene-level activities, but the results will not be as compelling as we wish them to be. We note that to be able to achieve the reasonable classification rates



(a) Regional Corner Detection ROC Curves



(b) Regional Corner Detection Precision-Recall Curves

Figure 3-13: *Feature Point Detection Curves:* Using a feature point detector to find pixel blocks containing pedestrians yields better results than background subtraction (compare with Fig. 3-11). "All non-masked" is so bad because the architectural features in the black areas (non-good, non-glare, non-shadows) of Fig. 3-10 generate a huge number of false positives. Although these results are better than background subtraction, we cannot achieve better than 85% precision for a non-degenerate classifier when considering all non-masked areas.

seen in this section, we rely heavily on the background being textureless and shadows being soft or non-existent. For many site monitoring scenarios, this is practical, but not for all. This suggests that if we use a detector more tuned to find pedestrians, we may get substantially better results. We will confirm this hypothesis in §3.4.

3.4 Efficient HOG-based Pedestrian Detection

3.4.1 Strong Model Motivation

In typical site monitoring applications, walking humans are the primary objects of interest; we need not concern ourselves with detecting objects like pencils, boats, cars, or telephones. Supplying evidence of this xassertion are examples such as a US Department of Transportation study finding that improved crosswalk safety can be achieved through automatic pedestrian detection [48]. Gavrila has also worked in the automotive sector attempting to detect pedestrians to mitigate potential vehicle-human collisions [36]. The US Defense Advanced Research Projects Agency (DARPA) funded a multi-year multi-institution research initiative called the Visual Surveillance and Monitoring (VSAM) project[12]. In it, the focus was on detecting and tracking vehicles and people and characterizing their interactions. More recently a series of challenge problem datasets have been produced for the PETS workshops (*e.g.* [29, 30]) where participants are tasked with tracking pedestrians and their belongings so that security events can be automatically detected.

In these settings, a critical task is being able to robustly detect and localize pedestrians. In scenes like the Grand Central one studied in this chapter, the detection task is made more challenging by the presence of crowds because many people are always or almost always occluded by other people. We saw in earlier sections that simpler techniques like background subtraction that are agnostic to the type of foreground object do not perform sufficiently well in these settings. Because of this, it can be advantageous to spend the time developing and training a strong appearance model instead of using more generic bottom-up techniques like background subtraction.

3.4.2 HOG Descriptors

For the work in this chapter, we have chosen to use Dalal and Triggs' [19] pedestrian detection algorithm, based on Histograms of Oriented Gradients (HOG). Variants of this algorithm are widely used because

- it performs well,
- it can be re-implemented without too much trouble, and
- the authors have released their source code.

The algorithm uses a collection of HOG descriptors tiled over a detector window. Each of these descriptors captures local shape information by building a histogram of image gradients. A detector window is characterized by a feature vector formed by concatenating the local descriptors.

To illustrate this process more concretely, we will describe the algorithm with a standard descriptor configuration. In doing so, we will make frequent reference to Fig. 3-14. At every location and size in the input image where we want to test for the presence of a pedestrian, we will construct a detection window descriptor as follows.

After extracting a test window, we rescale it to a canonical size (64×128 pixels in this case) and perform gamma correction: $I_\gamma(x, y) = \sqrt{I(x, y)}$, where $I(x, y)$ is the intensity of the pixel at position (x, y) in the cropped and rescaled input image.

We then extract block descriptors at every point on a regular 7×15 grid whose points are 8×8 pixels apart. This grid is illustrated as dots on the "Block Descriptor Locations" image in Fig. 3-14. A block descriptor uses data from a 16×16 block of pixels in $I_\gamma(x, y)$. This means that each block overlaps with each of its 4 neighbors by 50%.

Each block descriptor is computed by breaking up its input into a non-overlapping 2×2 grid of cells, each 8×8 pixels in size. In Fig. 3-14, cyan boxes correspond to blocks and blue boxes within them correspond to cells. From the pixels within a cell,

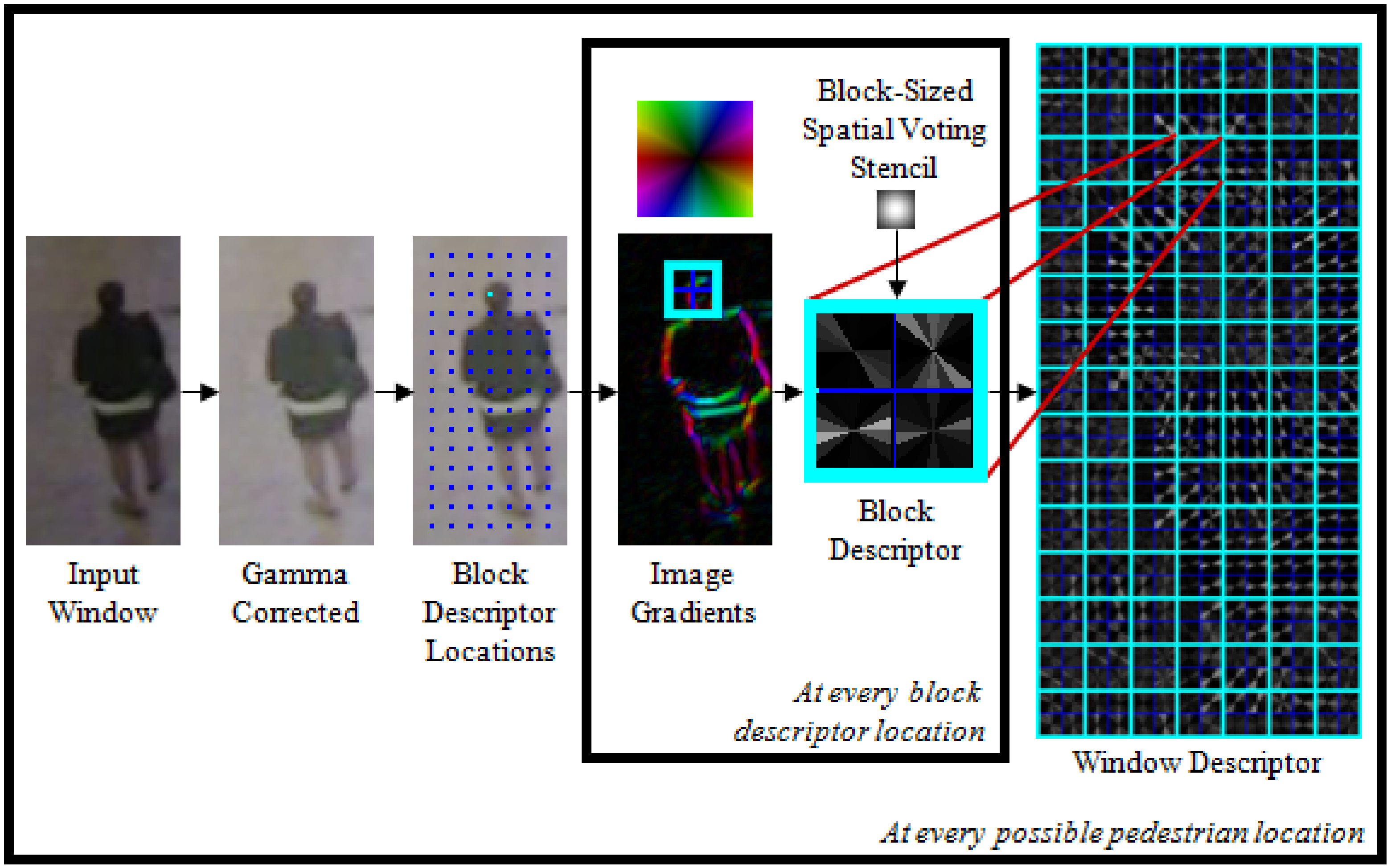

Input Window
Gamma Corrected
Block Descriptor Locations
Image Gradients
Block-Sized Spatial Voting Stencil
Block Descriptor
At every block descriptor location
Window Descriptor
At every possible pedestrian location

Figure 3-14: *Computing a HOG Descriptor:* In these illustrations, a window descriptor is built up from 7×15 evenly spaced and partially overlapping block descriptors. Each block descriptor contains 4 non-overlapping cells. Above the "Image Gradients" image is its color key. Gradients with low magnitude are dark, and the direction is encoded as a hue. See §3.4.2 for further details.

we produce a weighted histogram of image gradient orientations:

$$\mathrm{h}(o; b_x, b_y, \mathbb{C}) = \sum_{x,y \in \mathbb{C}} \underbrace{\mathrm{w}(x,y)}_{\text{spatial kernel}} \cdot \underbrace{||\mathrm{g}(x,y;b_x,b_y)||}_{\text{gradient magnitude}} \cdot \delta\left(o, \underbrace{\left\lfloor \Omega \cdot \left(\frac{\angle \mathrm{g}(x,y;b_x,b_y)}{\pi} \mod 1 \right) \right\rfloor}_{\text{histogram bin (from gradient direction)}} \right), \tag{3.1}$$

where $\mathrm{h}(o; b_x, b_y, \mathbb{C})$ is the weight in a cell's orientation bin. The bin is indexed by o. (b_x, b_y) is the pixel location of the center of the block to which the cell belongs, and $\mathbb{C}$ is the set of all pixel locations within the cell. $\mathrm{w}(x, y)$ is the "Block-Sized Spatial Voting Stencil" depicted in Fig. 3-14. It is used to emphasize gradients near the center of the block and it has the form of a Gaussian:

$$\mathrm{w}(x,y) = e^{-(x^2+y^2)/(2\sigma^2)} \tag{3.2}$$

for some user-selected kernel bandwidth σ. The histogram is built up from image gradients:

$$\mathrm{g}(x,y;b_x,b_y) = \begin{pmatrix} \partial \mathrm{I}_\gamma(x+b_x, y+b_y)/\partial x \\ \partial \mathrm{I}_\gamma(x+b_x, y+b_y)/\partial y \end{pmatrix}, \tag{3.3}$$

and it has $\Omega = 9$ bins.

All the cell histograms corresponding to a block are concatenated together to form a feature vector of length

$$\frac{1 \text{ element}}{\text{orientation bin}} \cdot \frac{9 \text{ orientation bins}}{\text{cell}} \cdot \frac{(2 \cdot 2) \text{ cells}}{\text{block}} = 36 \frac{\text{elements}}{\text{block}}. \tag{3.4}$$

This feature vector is then normalized so that the squares of its elements sum to 1. The normalized feature vector is the block descriptor. Note that the histograms are computed at the cell level, but spatial voting weights and normalization are associated with blocks.

The set of all block descriptors for a detector window are concatenated to form the window descriptor. For the standard descriptor layout, it has

$$\frac{36 \text{ elements}}{\text{block}} \cdot \frac{(7 \cdot 15) \text{ blocks}}{\text{window}} = 3,780 \frac{\text{elements}}{\text{window}}. \tag{3.5}$$

In Fig. 3-14 there is a rendering of the descriptor for the input image window shown in that same figure. Gray wedges show the histogram mass corresponding to gradients pointing in the same direction as the wedges. For clarity, block descriptors are tiled next to each other in this rendering, even though neighboring blocks share input data (but the data have different spatial weights for each block).

We note that the actual descriptor computation differs slightly from our description above. Eqn. 3.1 shows each gradient casting all of its voting mass into a single bin. It is more robust to proportionally spread the voting mass between the two closest bin centers, and we do so in practice. Also, our inputs are color images, so the image gradients have 3 channels, not just 1. We compute color gradients for each of the red, green, and blue channels. At each pixel, we retain the gradient with the highest magnitude. Finally, in this section we have given specific parameter settings such as the number of histogram bins (9), arrangement of cells within blocks (2×2 cells of size 8×8), and the arrangement of blocks within windows (7×15 blocks spaced at every 8×8 pixel locations). The algorithm is able to use other layouts, but we found that Dalal and Trigg's default choices produced noticeably better results than other ones with which we informally experimented.

3.4.3 Classification

Support Vector Machines

Given a descriptor for a detection window, we now wish to classify that window as a pedestrian or non-pedestrian. A typical way to cast classification problems is to learn a function that returns a positive number for feature vectors extracted from windows with pedestrians and a negative number for feature vectors corresponding to non-pedestrian windows.

A desirable property of this function is that, given a training set, it maximizes the difference between the minimum positive number for pedestrian windows and the maximum negative number for non-pedestrian windows. More formally, suppose we have a set of n training samples:

$$\mathbb{D} = \{\ (\boldsymbol{x}_i, l_i) \mid \boldsymbol{x}_i \in \mathbb{R}^p\ \wedge\ l_i \in \{-1, +1\}\ \}_{i=1}^{n} \tag{3.6}$$

where $\boldsymbol{x}_i$ is a p-dimensional feature vector and l_i is its corresponding label: $+1$ if $\boldsymbol{x}_i$ represents a pedestrian and -1 if not. We would like to find a separating hyperplane with a normal vector $\boldsymbol{w}$ and bias w_0 that minimizes $||\boldsymbol{w}||$ while classifying all feature vectors in the training set correctly, *i.e.*

$$l_i \cdot \left(\boldsymbol{w}^\top \boldsymbol{x}_i - w_0\right) \geq 1 \quad \forall\, i \in \{1, ..., n\}. \tag{3.7}$$

Our real-valued classifier function is then

$$\mathrm{f}(\boldsymbol{x}) = \left(\boldsymbol{w}^\top \boldsymbol{x} - w_0\right). \tag{3.8}$$

Given a novel feature vector $\boldsymbol{x}$, we classify it as a pedestrian if $\mathrm{f}(\boldsymbol{x}) > 0$ and as a non-pedestrian if $\mathrm{f}(\boldsymbol{x}) \leq 0$.

This form of classifier is called a Support Vector Machine (SVM). SVMs are widely used in machine learning problems because efficient solvers are available, classification can be done very efficiently, and they tend to perform well with real-world test data. Since being introduced by Vapnik in 1963, SVMs have been generalized to support non-separable training data (*i.e.* when no $\boldsymbol{w}$ and w_0 exist that satisfy Eqn. 3.7) and nonlinear decision boundaries. An excellent tutorial by Burges [11] goes into detail about how SVMs work and how to find the optimal $\boldsymbol{w}$ and w_0 given training data. For the work described in this chapter, we use a linear SVM solver based on the SVMlight package developed by Joachims [50].

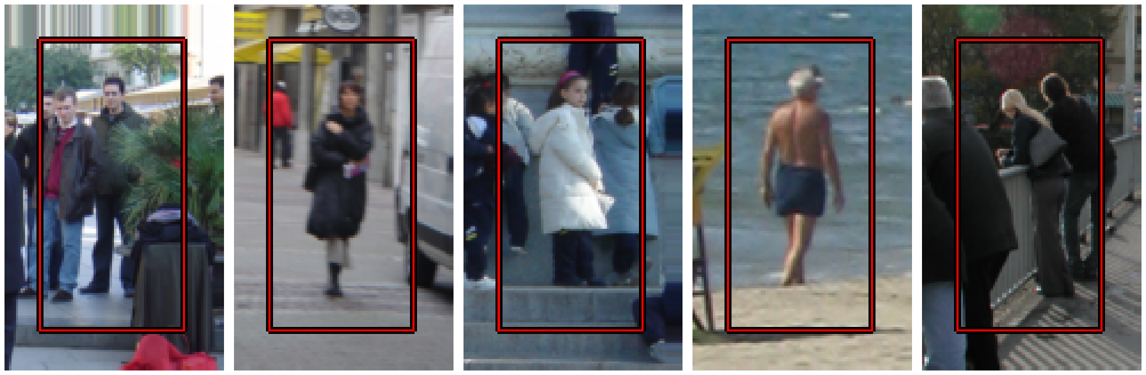


Figure 3-15: *Positive Training Images from the* INRIA-orig *Dataset:* The red rectangles delimit the detector window boundaries.

Figure 3-16: *Negative Training Images from the* INRIA-orig *Dataset:* All INRIA negative images have no humans visible anywhere in them.

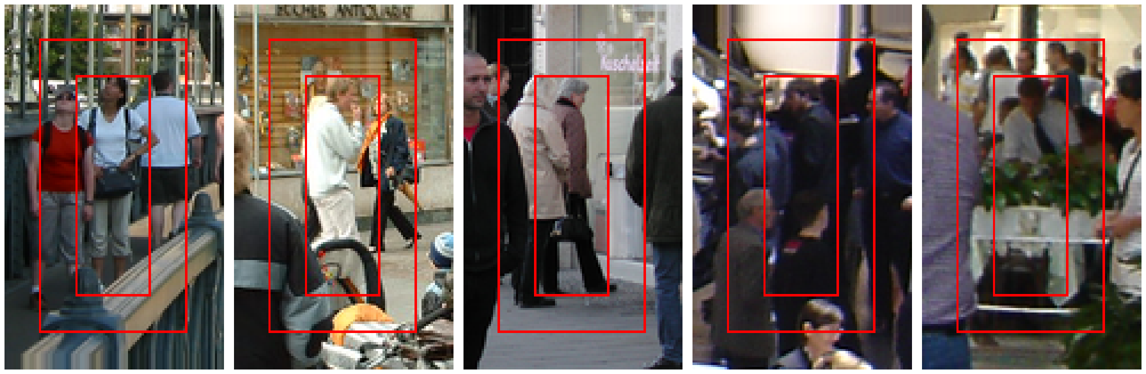


Figure 3-17: *Positive Training Images from the* INRIA-new *Dataset:* These images are people that were excluded from the *INRIA-orig* dataset. The outer red box shows the detector window boundary. The inner red box shows the canonical pedestrian size within a window.

	INRIA -orig	*INRIA -orig-hard*	*INRIA -new*	*INRIA -new-hard*	*Grand Central*
positive images	614		614		25
positive examples	2,416		3,384		4,260
negative images	1,218	1,218	1,218	1,218	N/A
negative examples	9,120	10,935	9,120	10,935	113,787

Table 3.2: Training Datasets

Datasets

To test the performance of our pedestrian detector, we have used several related datasets. In these classification experiments, individual image chips are extracted and a single window descriptor is built and tested for each chip.

We have five training datasets (see Tab. 3.2 for key statistics):

- ***INRIA-orig:*** This is the dataset provided by Dalal and Triggs and used in the 2005 paper [19]. The images are typical of photographs taken on vacations. Most were taken outdoors and ones with humans are nearly always taken at near eye-level (see Fig. 3-15 for typical examples). Pedestrians with non-trivial amounts of occlusion were omitted from the training set. All negative examples were randomly sampled from images containing no visible humans (see Fig. 3-16 for examples of full images).

- ***INRIA-orig-hard:*** After training the classifier on the *INRIA-orig* dataset, the negative training images were exhaustively scanned for false positive windows, which we call "hard" negative examples, following Dalal's convention. These samples are used as additional negative training samples and a new classifier is trained, as was done by Dalal and Triggs [19].

- ***INRIA-new:*** In the *INRIA-orig* dataset, the positioning of bounding boxes was qualitatively imprecise for positive examples. We re-annotated all positive images to provide more consistent alignment. To do so, we instructed the annotator to mark the vertical image extent of living tissue in a person and the centroid of their head. The head centroid was used to choose the horizontal position of the bounding box. Additionally, we added many positive examples that were excluded from *INRIA-orig* due to high occlusion (see Fig. 3-17 for some examples of pedestrians that were excluded from *INRIA-orig*). The negative training set is identical to that of *INRIA-orig*.

- ***INRIA-new-hard:*** As was the case for the *INRIA-orig-hard* dataset, a new classifier was created using hard negative examples.

- ***GrandCentral:*** We have independently created a dataset from the Grand Central scene featured throughout this chapter. To do so, we randomly selected a collection of frames from a 1 hour video and hand-labeled the position of every pedestrian. Low contrast and significant occlusions prevented the usage of the same annotation protocol that was used for *INRIA-new*. Instead the tip of each pedestrian's head was manually marked. We automatically chose the bounding box coordinates by assuming all pedestrians are $1.75m$ tall, by using camera calibration to find their image height, and by assuming that pedestrians appear as nearly vertical in image space[5]. Negative examples were acquired by randomly sampling detector windows and discarding any that are close to a pedestrian[6]. Crowd density, background clutter, and perspective distortions are noticeably different from the INRIA photographs. Also, the negative examples from this dataset frequently contain partial pedestrians or pedestrians of incorrect scale, by design.

For the classification experiments, we have three test datasets (see Tab. 3.3 for key statistics):

- ***INRIA-orig:*** Like the *INRIA-orig* training set, we re-use Dalal and Triggs' test set. The images in the test set are independent of those in the training set.

[5]We did not correct for skew induced by having an image plane that is not perpendicular to the ground plane. Doing so using nearest-neighbor sampling induces dramatic false gradients. We believe that the best way to handle this situation would be modify the HOG voting system to correct for the perspective distortions after computing image gradients.

[6]Two windows are "close" if they pass the PASCAL Visual Object Challenge 50% overlap test [106]. See §3.4.4 of this thesis for more details.

	INRIA-orig	*INRIA-new*	*GrandCentral*
positive images	288	288	12
positive examples	1,132	1,178	3,808
negative images	453	453	N/A
negative examples	995,452	995,452	27,808

Table 3.3: Test Datasets for Classification

	Included Training Sets				
Classifier Name	*INRIA -orig*	*INRIA -orig-hard*	*INRIA -new*	*INRIA -new-hard*	*Grand Central*
INRIA-orig	✓				
INRIA-orig + hard	✓	✓			
INRIA-orig + hard + GrandCentral	✓	✓			✓
INRIA-new			✓		
INRIA-new + hard			✓	✓	
INRIA-new + hard + GrandCentral			✓	✓	✓

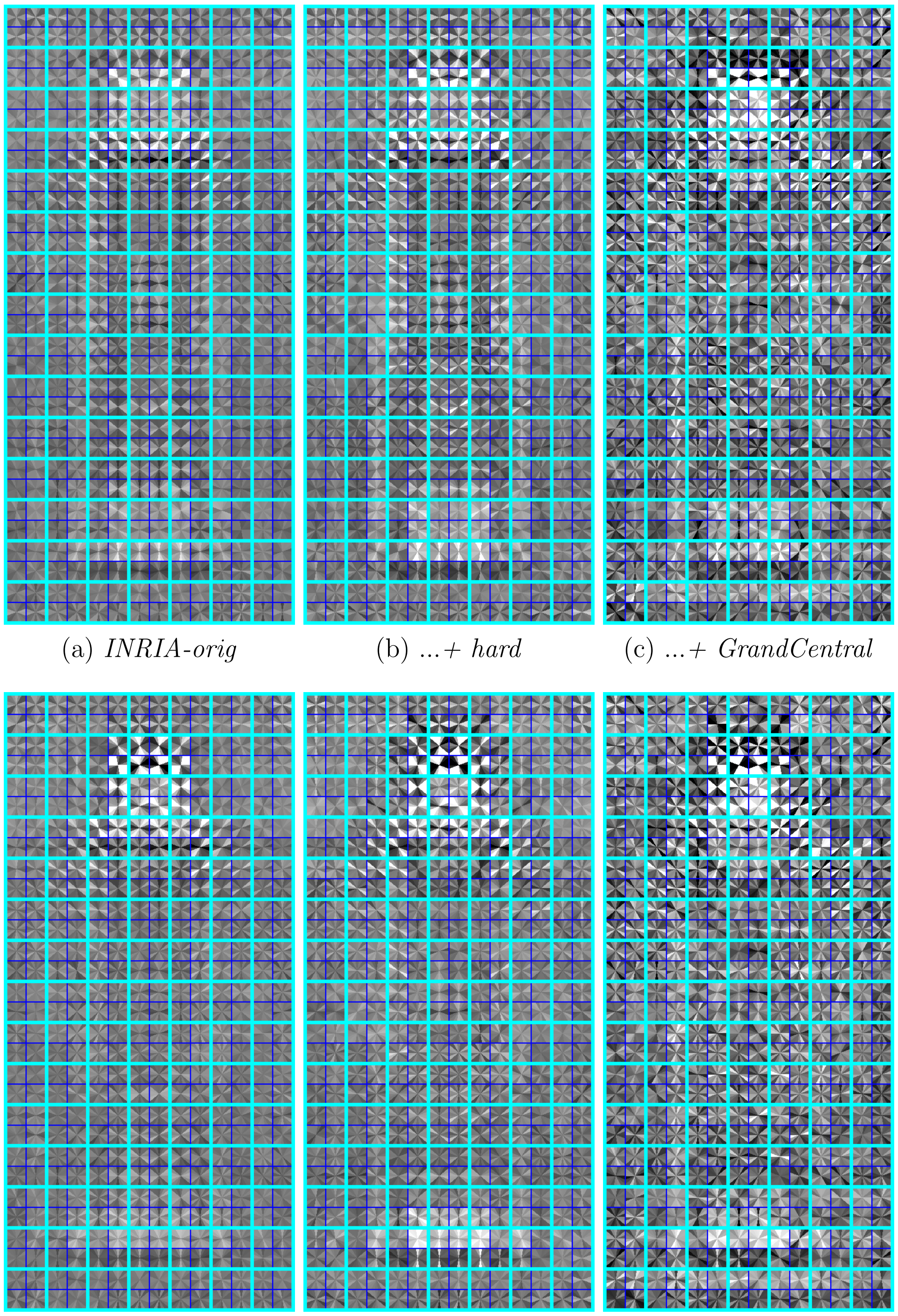
Table 3.4: Training Set Combinations for HOG Classifier Evaluation

- ***INRIA-new:*** We also made the test annotations qualitatively more self-consistent and added in a number of pedestrians not included in the *INRIA-orig* test set.

- ***GrandCentral:*** We randomly sampled frames from a second Grand Central video. This video is 5.5 hours long, with one hour overlapping the time of the training video, but it was taken from the opposite side of the room. Positive and negative examples were extracted in the same manner as the training set.

Classification Results

For each training set combination shown in Tab. 3.4, we created a classifier. Because we are using a linear SVM for classification, the normal of the separating hyperplane, $\boldsymbol{w}$, can be rendered in a manner similar to window descriptors. Recall that the *Window Descriptor* in Fig. 3-14 shows the voting mass in each orientation histogram bin as a grayscale wedge. In Fig. 3-18, we show a similar plot for the classifier normals, where elements of $\boldsymbol{w}$ that are 0 are rendered as 50% gray wedges. If an element is



(a) *INRIA-orig* (b) ...+ *hard* (c) ...+ *GrandCentral*

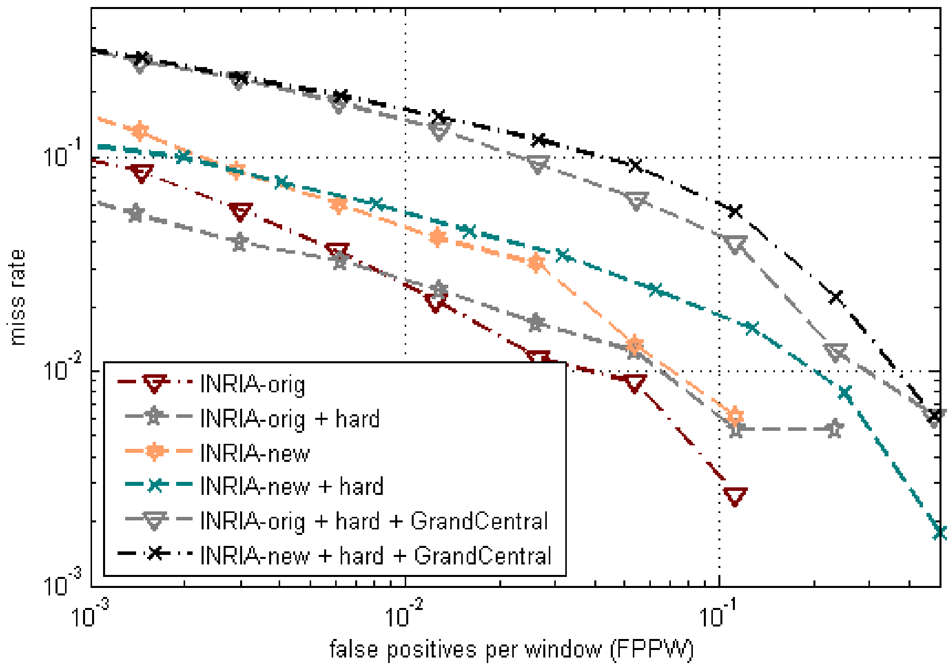
(d) *INRIA-new* (e) ...+ *hard* (f) ...+ *GrandCentral*

Figure 3-18: *Classifier Renderings:* Here we show renderings of the $\boldsymbol{w}$ portion (see Eqn. 3.8) of several learned classifiers (see Tab. 3.4).

exactly 0, then the corresponding histogram bin is ignored during classification (see Eqn. 3.8). A white wedge corresponds to histogram bins that should contain a large amount of voting mass if the image window corresponds to a pedestrian. Bins where nearly no voting mass should be found are shaded black.

There are several noteworthy aspects of the classifiers depicted in Fig. 3-18. In Fig. 3-18a, the classifier using the *INRIA-orig* training set has learned to look for heads, shoulders, and feet. It is biased against vertical edges (horizontal gradients) just below the shoulders and feet. It is also biased against horizontal edges in the leg and upper arm areas. When additional negative training data are included (Fig. 3-18b), the rendering has higher contrast in many areas, indicating that the learning algorithm has implicitly chosen to build a stronger, more-tuned classifier. If we add in the very large *GrandCentral* training data, the classifier undergoes much more substantial changes (see Fig. 3-18c). The same general pattern of additional complexity being captured by larger training sets is observed with the *INRIA-new* training data in Fig. 3-18d–f. Note that especially in the head region, the classifier is much more discriminative when using the *INRIA-new* dataset (Fig. 3-18d). This is because *INRIA-new* was created with the explicit purpose of registering the training data more consistently, especially with respect to the head.

In Fig. 3-19, we show the classification results on the *INRIA-orig* test set using each of the classifiers described in Tab. 3.4. We include ROC curves because they are widely used in the literature, but we will concentrate our discussion on the precision-recall curves because the size of the positive and negative classes is so different. These plots show that there is a clear advantage to matching the annotation protocols for the training and test data: ignoring the *GrandCentral* data for a moment, training based on *INRIA-orig* is clearly superior to *INRIA-new* because it takes into account the imprecise nature of the test annotations. In precision-recall terms, adding the extra *hard* training data helps substantially. This is unsurprising because this extra data is sampled from the same types of images that are found in the test set. Also unsurprising is the fact that adding in the extremely large *GrandCentral* training set makes the results far worse. That dataset is large enough that it causes the classifier



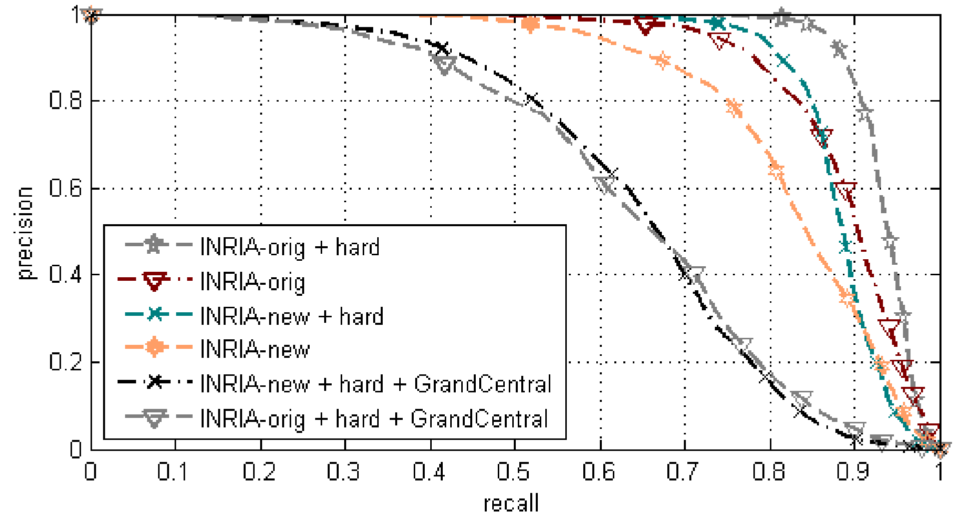


Figure 3-19: *Classification Results on* INRIA-orig*:* In the top graph, we show ROC curves for each of the classifiers described in Tab. 3.4 when tested against *INRIA-orig* (see Tab. 3.3). Note that both axes use logarithmic scales. The closer the curve is to the bottom and left of the plot, the better. In the bottom graph, we show precision-recall curves for the same experiments. The closer the curve is to the top and right, the better. In both plots, the legend is sorted by the area under the (full) curve. The more similar the training set is to the *INRIA-orig* test set, the better the results.

to be overfit to it, when used on general vacation-style imagery.

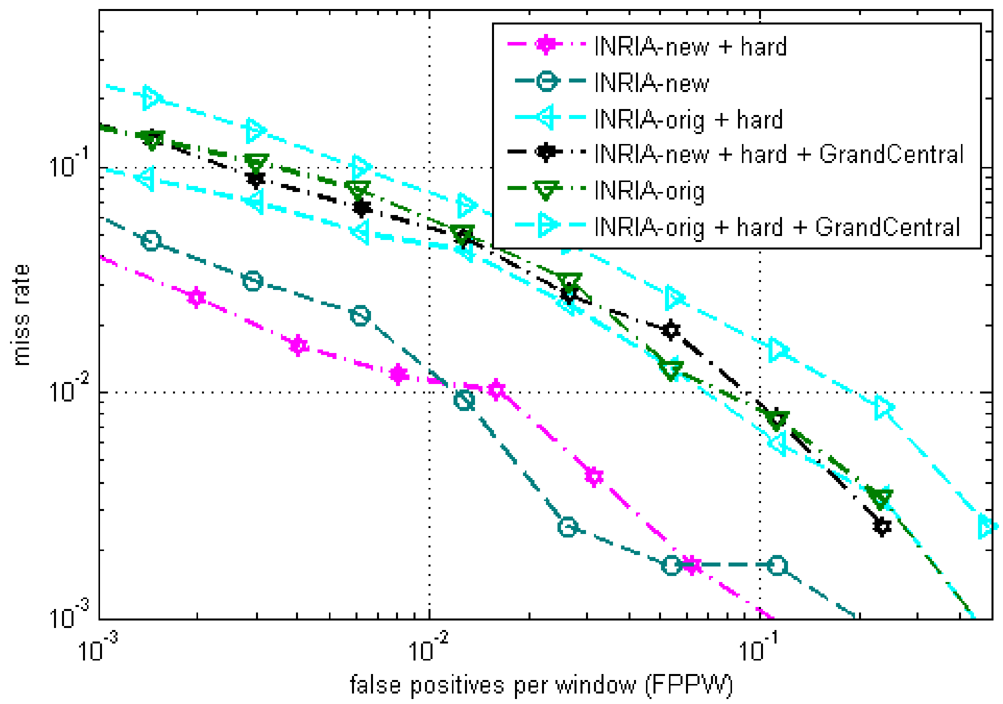
In Fig. 3-20, we show the classification results on the *INRIA-new* test set. We see that the best results are achieved when the matched *INRIA-new + hard* training data are used. This more-precise test set (relative to *INRIA-orig*) allows the classifiers to achieve better performance than was seen with the original annotations. This is especially apparent when observing that training with the *GrandCentral* data, there is less evidence of overfitting (compare the green and magenta curves here to the magenta and black curves in Fig. 3-19). This improvement in results is encouraging because we will eventually want to search for the precise locations of pedestrians in images.

We show our results for the *GrandCentral* test set in Fig. 3-21. This test set differs from the *INRIA* ones in that there is far more occlusion of pedestrians, the viewpoint is different, and the scene is indoors in a single setting. Also, the negative examples frequently include parts of people or people at the wrong scale. When adding *hard* training data to either *INRIA-orig* or *INRIA-new*, the performance gets worse. This is because the *hard* data causes the classifiers to too heavily tune towards discriminating against outdoor scenes. We also note that the results using the *INRIA-new* classifier are substantially better than those of the *INRIA-orig*. Once we add in the *GrandCentral* training data, the results improve dramatically and *INRIA-new* versus *INRIA-orig* is less important.

In these classification experiments, we have seen that substantial performance improvements in the pedestrian detector can be achieved by matching the types of scenes and the annotation methodology between training and test sets. We have also seen that our *GrandCentral* data is challenging even for a strong detector like Dalal and Triggs'. Next we will investigate the task of finding all pedestrians and their locations within complete images, not just pre-cropped image chips.

3.4.4 Detection and Localization

Recall that we are using a pedestrian detector so that we can find and track people in videos. The mechanism just described in §3.4.3 classifies an image window as either



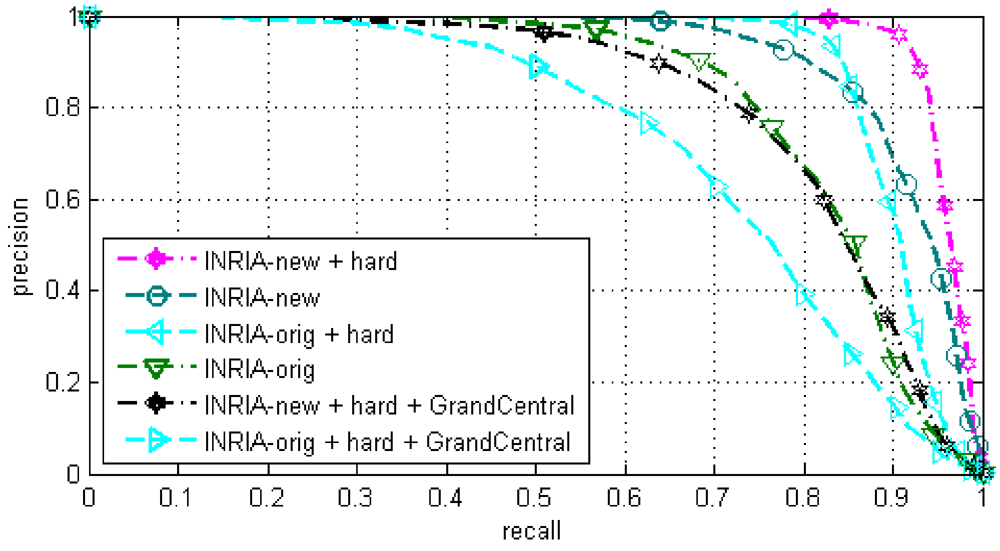
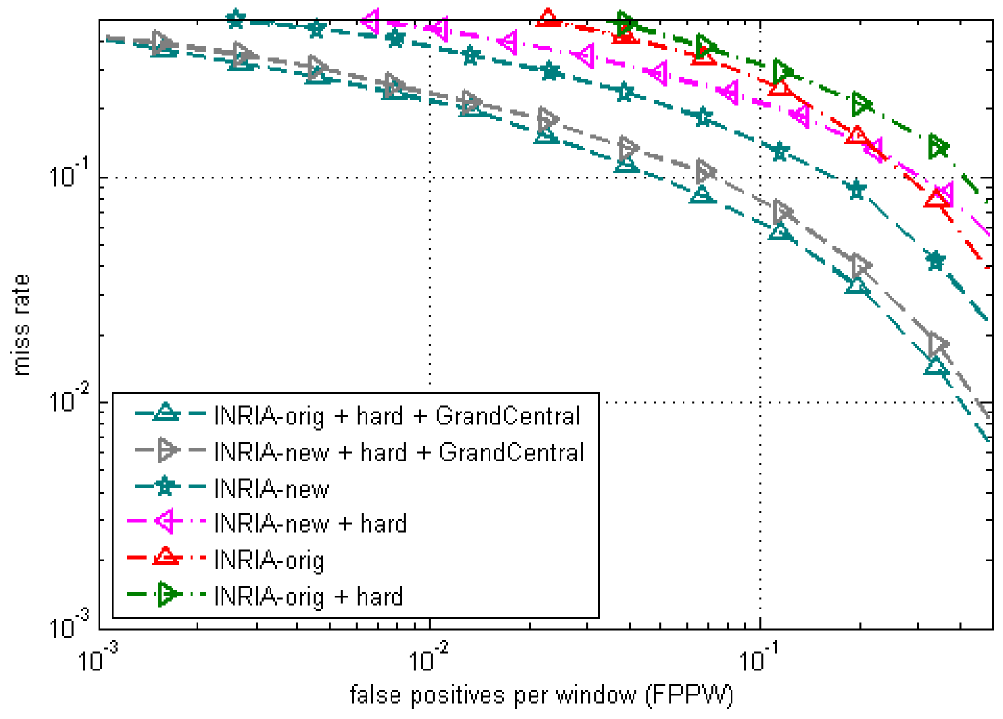


Figure 3-20: *Classification Results on* INRIA-new*:* See Fig. 3-19 for a description of how to interpret these ROC and precision-recall plots.



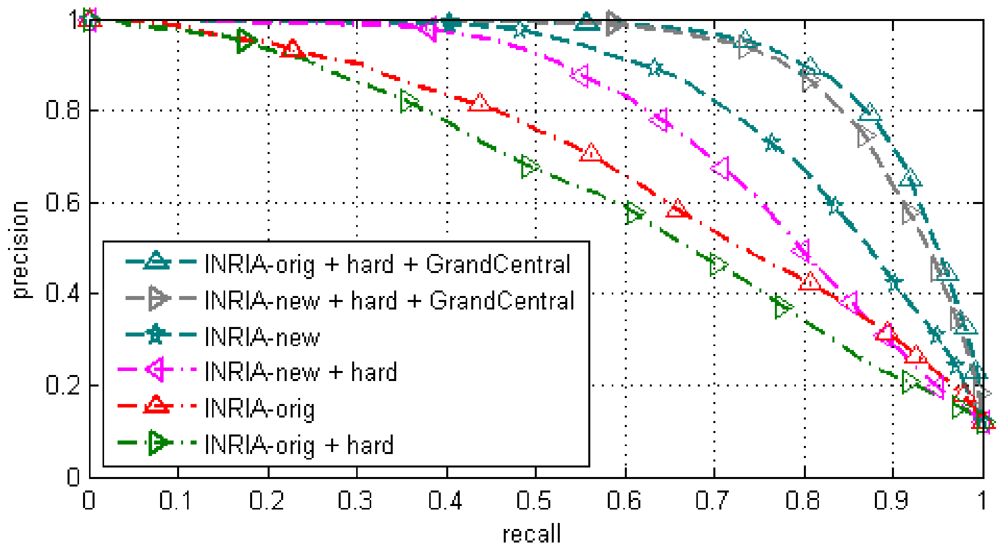


Figure 3-21: *Classification Results on* GrandCentral*:* See Fig. 3-19 for a description of how to interpret these ROC and precision-recall plots.

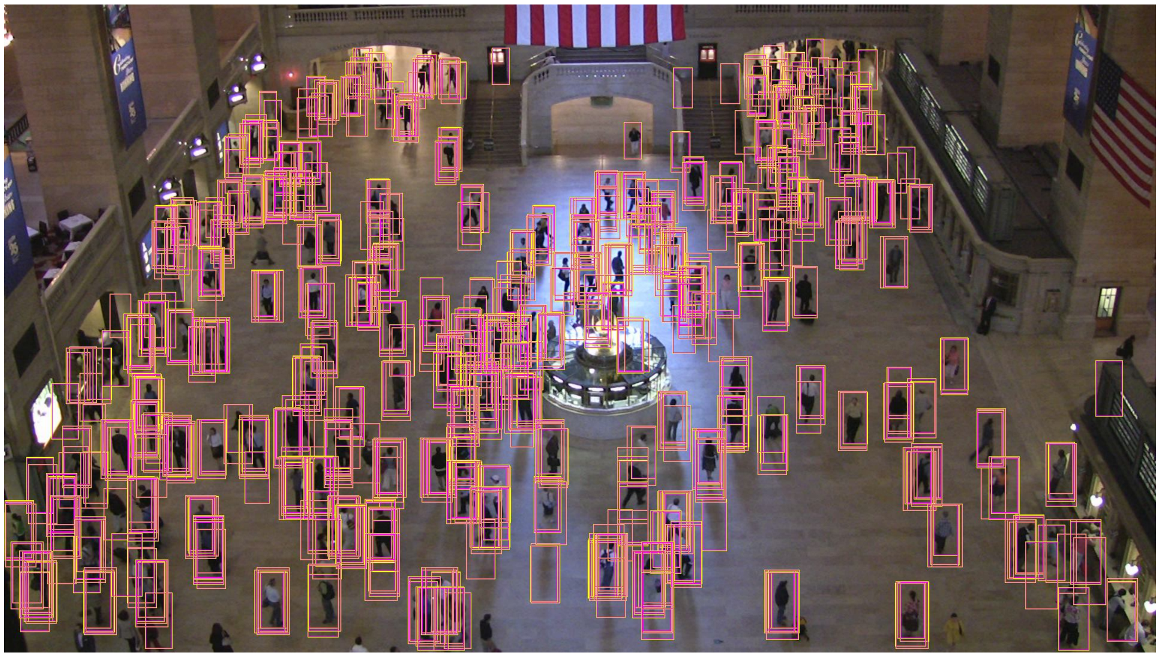


Figure 3-22: *Multiple Detections:* The magenta and yellow bounding boxes indicate every positive detection made by the *INRIA-new + hard + GrandCentral* classifier in this image. Note that for nearly every actual pedestrian, there are many positive detections.

a pedestrian or a non-pedestrian. This mechanism works well when we supply it with image chips that have either (a) a pedestrian of the correct size who is centered in the window or (b) no pedestrians at all.

We saw hints of these constraints breaking down when observing that the curves in Fig. 3-21 look worse than those in Fig. 3-20 or Fig. 3-19. The negative data for the *INRIA* test sets include only windows from images with absolutely no humans present. Those test sets do not attempt to evaluate the performance of detecting humans at the wrong scale, of having false positives due to misaligned windows, or of dealing with significant occlusions. In all three test sets, we ignored the problem of how to interpret many neighboring windows with high detection scores. These clusters of high scores can be due to multiple people being present or because the partial translation invariance in the HOG feature is designed to allow for small misalignments.

In Fig. 3-22, we show all windows where the *INRIA-new + hard + GrandCentral* classifier has a positive score (Eqn. 3.8). Note that for nearly every actual person, there are numerous positive classification results. To be able to track the pedestrians, we want to select just one positive detector window per actual person.

Meanshift and Non-maxima Suppression

To solve the multiple-detection problem, we use the meanshift algorithm [15] and non-maxima suppression, as was done by Dalal and Triggs [19]. Meanshift is used to find the primary modes in classifier scores. Non-maxima suppression is then used to cull modes that are too close to each other.

To illustrate how these work, consider the simulated data shown in Fig. 3-23. In it, there are pedestrians located at the following image locations: $(0, 0)$, $(5, 0)$, and $(8, 3)$. We simulated a collection of positive classification scores and locations by (a) randomly sampling positions around the three ground truth pedestrian locations and then (b) randomly sampling classification scores. In the figure, the size of a blue dot is proportional to its score, with larger dots corresponding to cases where the classifier is more confident that a pedestrian is present.

The meanshift algorithm requires that the user choose a spatial kernel that represents an object's shape (at least roughly). Starting with the kernel centered at a given data point, all points are re-weighted according to the kernel function. The kernel is then shifted to the mean location of the re-weighted points. This process is repeated iteratively until convergence. We used a Gaussian kernel for the simulation shown in Fig. 3-23 and for the real-data experiments described later in this section[7]. The trajectory of one such kernel is shown in the middle plot of Fig. 3-23. The ellipses shown the kernel contour at 2.5 standard deviations.

More formally, the update equation for a meanshift kernel's position $\boldsymbol{y}^{(j)}$ at step

[7]For the simulation, our kernel bandwidth is $\Sigma = \begin{pmatrix} 0.45^2 & 0 \\ 0 & 9^2 \end{pmatrix}$. For the results that we will discuss later when we present Fig. 3-24, $\Sigma = \begin{pmatrix} 4^2 & 0 & 0 \\ 0 & 8^2 & 0 \\ 0 & 0 & 1.3^2 \end{pmatrix}$. There is a third dimension because we run meanshift over location and scale. For the ROC and precision-recall curves discussed later in this section, the meanshift bandwidth was one of the optimization parameters.

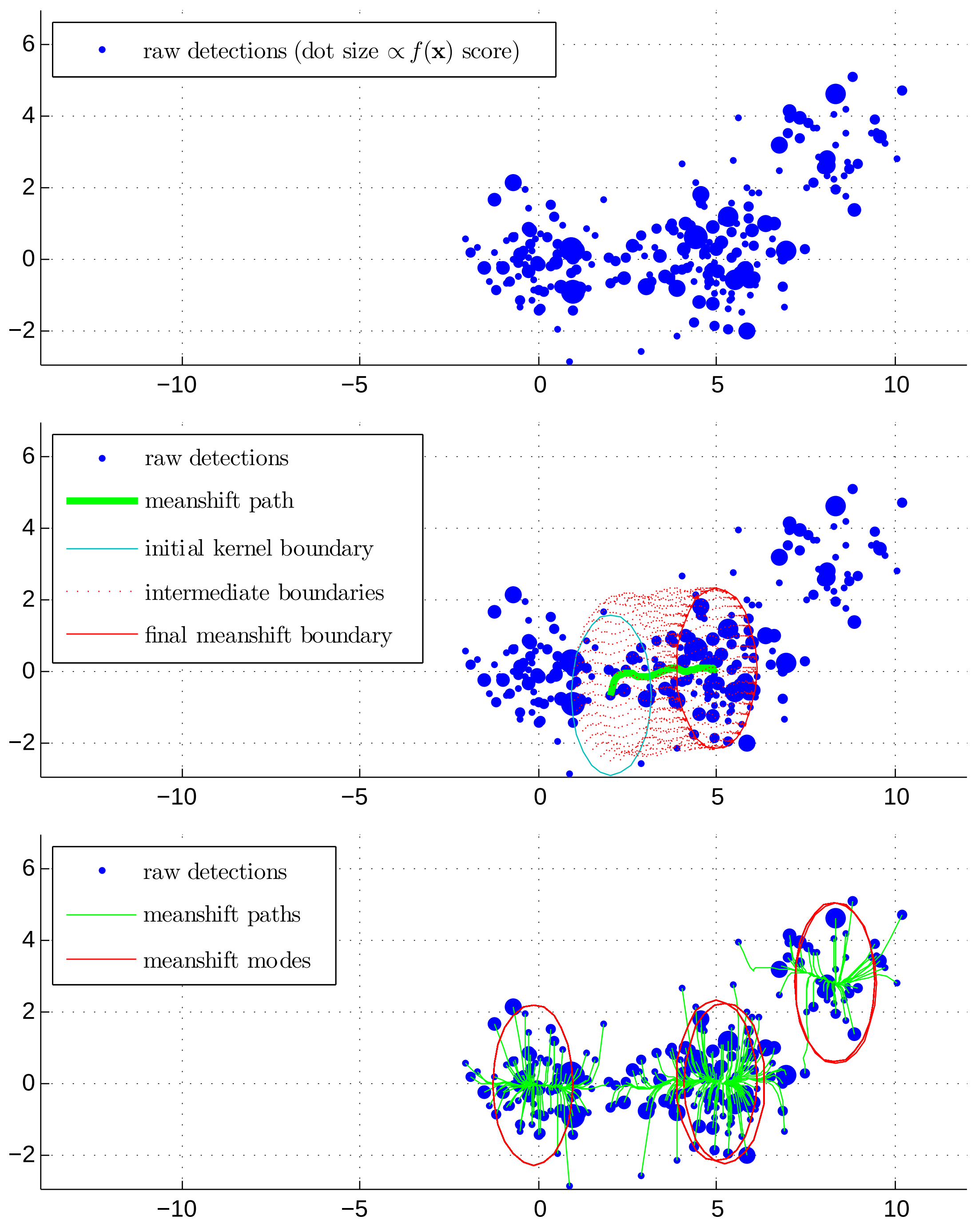


Figure 3-23: *Meanshift Illustration:* The blue dots represent hypothetical (x, y) image locations of detection windows with positive classification scores (see Eqn. 3.8). The size of each dot is proportional to its score. In this simulation, there are pedestrians at $(0, 0)$, $(5, 0)$, and $(8, 3)$. The middle plot shows the path followed by the meanshift algorithm for a single data point. The bottom plot shows the meanshift trajectories for all data points and the final mode locations.

j is given by

$$\boldsymbol{y}^{(j)} = \frac{\sum_{i=1}^{\hat{P}} \boldsymbol{x}_i \overbrace{\mathrm{f}(\boldsymbol{x}_i) \exp\left(-\frac{1}{2}\left(\boldsymbol{x}_i - \boldsymbol{y}^{(j-1)}\right)^\top \Sigma^{-1} \left(\boldsymbol{x}_i - \boldsymbol{y}^{(j-1)}\right)\right)}^{\text{point re-weighting}}}{\underbrace{\sum_{i=1}^{\hat{P}} \mathrm{f}(\boldsymbol{x}_i) \exp\left(-\frac{1}{2}\left(\boldsymbol{x}_i - \boldsymbol{y}^{(j-1)}\right)^\top \Sigma^{-1} \left(\boldsymbol{x}_i - \boldsymbol{y}^{(j-1)}\right)\right)}_{\propto \text{ kernel's supporting weight } (w)}} \tag{3.9}$$

where $\{\boldsymbol{x}_i\}_{i=1}^{\hat{P}}$ is the set of locations where there are positive classification scores, $\mathrm{f}(\cdot)$ is the classifier scoring function from Eqn. 3.8, and $\boldsymbol{y}^{(j-1)}$ is the kernel's position at the previous iteration.

In the bottom plot in Fig. 3-23, we show the update trajectories (green curves) and final kernel positions (red ellipses) resulting from starting a meanshift process at each data point.

After finding the final kernel positions, we need to cluster them to remove redundant detections. To do so, we retain the kernel with the greatest supporting weight (see Eqn. 3.9) as the first mode. We then examine each other final kernel position, in decreasing order of supporting weight. We keep all kernel positions that are at least one kernel bandwidth, Σ, away from all other existing modes. In Fig. 3-23, this means that we will keep one mode near each of $(0, 0)$, $(5, 0)$, and $(8, 3)$. Redundant overlapping red ellipses in the bottom plot will be discarded, and only one of the two ellipse clusters near $(5, 0)$ will be retained. The full algorithm for culling redundant detections is outlined in Alg. 3.2.

Detector Evaluation

For the classification experiments, the input was a single image window and the output was a score. Good classifiers give large positive scores when the input window contains a pedestrian and large negative scores when the input window does not contain a centered and properly sized pedestrian. For our detection experiments, the input is a full image and the detection system is responsible for outputting a list of

Algorithm 3.2 Meanshift and Non-maxima Suppression

```
1:  function RemoveRedundantDetections($\{\boldsymbol{x}_i\}_{i=1}^{\hat{P}}$)
2:      $\{(\boldsymbol{y}_i, w_i)\}_{i=1}^{\hat{P}} \leftarrow$ Meanshift($\{\boldsymbol{x}_i\}_{i=1}^{\hat{P}}$)
3:      $\{(\boldsymbol{y}'_i, w'_i)\}_{i=1}^{\hat{P}} \leftarrow$ SortByDecreasingW($\{(\boldsymbol{y}_i, w_i)\}_{i=1}^{\hat{P}}$)
4:      return NonMaximaSuppression($\{(\boldsymbol{y}'_i, w'_i)\}_{i=1}^{\hat{P}}$)
5:  end function

6:  function Meanshift($\{\boldsymbol{x}_i\}_{i=1}^{\hat{P}}$)
7:      for $i \in \left(1, ..., \hat{P}\right)$ do
8:          $y_i \leftarrow x_i$
9:          $w_i \leftarrow \mathrm{f}(x_i)$
10:         repeat
11:             Update $y_i$ using Eqn. 3.9
12:         until convergence
13:     end forreturn $\{(\boldsymbol{y}_i, w_i)\}_{i=1}^{\hat{P}}$
14: end function

15: function NonMaximaSuppression($\{(\boldsymbol{y}'_i, w'_i)\}_{i=1}^{\hat{P}}$)
16:     $M \leftarrow \emptyset$                          ▷ the set of all modes (final detections)
17:     for $i \in \left(1, ..., \hat{P}\right)$ do                      ▷ for each final kernel
18:         $closeToExistingMode \leftarrow False$
19:         for $m \in M$ do                           ▷ for each discovered mode
20:             if $||y'_i - m||_\Sigma < 1$ then
21:                 $closeToExistingMode \leftarrow True$
22:             end if
23:         end for
24:         if $\neg closeToExistingMode$ then      ▷ only if far from all existing modes
25:             $modes \leftarrow (modes \cup m)$
26:         end if
27:     end for
28:     return $M$
29: end function
```

all pedestrian locations. To evaluate this list, we use the PASCAL Visual Object Challenge (VOC) protocol [106], which we summarize here.

Suppose that for each of I images, we have a set $\mathbb{D}_{gt}$ containing P ground-truth pedestrian bounding boxes,

$$\mathbb{D}_{gt} = \left\{ \boldsymbol{x}_j = (i_j, x_j, y_j, w_j, h_j)^\top \right\}_{j=1}^{P}, \tag{3.10}$$

where the j^{th} bounding box's top left corner is at (x_j, y_j) in image i_j and its size is $w_j \times h_j$ pixels. Further suppose we have a set $\mathbb{D}_{det}$ containing $\hat{P}$ detections,

$$\mathbb{D}_{det} = \left\{ (\hat{\boldsymbol{x}}_i, \hat{s}_i) \;\middle|\; \hat{\boldsymbol{x}}_i = (i_i, \hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i)^\top \;\wedge\; \hat{s}_i = \mathrm{f}(\hat{\boldsymbol{x}}_i) \right\}_{i=1}^{P},$$

where $\hat{s}_i$ is the classification score for detection i.

We wish to find a bipartite matching $\mathrm{m}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j)$ between the detections and the ground truth, subject to the constraints

$$\mathrm{m}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) \in \{0,1\} \qquad \forall\; i \in \{1, ..., \hat{P}\} \;\wedge\; j \in \{1, ..., P\} \tag{3.11}$$

$$\mathrm{m}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) = 0 \qquad \text{if } \neg\mathrm{close}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) \tag{3.12}$$

$$\sum_{i'=1}^{\hat{P}} \mathrm{m}(\hat{\boldsymbol{x}}_{i'}, \boldsymbol{x}_j) \in \{0,1\} \qquad \forall\; j \in \{1, ..., P\} \tag{3.13}$$

$$\sum_{j'=1}^{P} \mathrm{m}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_{j'}) \in \{0,1\} \qquad \forall\; i \in \{1, ..., \hat{P}\}, \tag{3.14}$$

where $\mathrm{m}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) = 1$ means that detection $\hat{\boldsymbol{x}}_i$ is a *close* and non-redundant match to the pedestrian at $\boldsymbol{x}_j$. This system is underconstrained, so we choose a greedy mapping $\mathrm{m}^*(\cdot,\cdot)$ by iteratively attempting to match the detection with the highest score, $\hat{s}_i$, to an unmatched ground truth bounding box.

We consider a detection and a ground truth bounding box to be close if the ratio of intersection to union areas is sufficiently high, *i.e.* if the following is non-zero:

$$\mathrm{close}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) = \begin{cases} 1 & \text{if } \tau_{det} < \big(\mathrm{A}_{\text{intersect}}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) \,/ \mathrm{A}_{\text{union}}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j)\big) \\ 0 & \text{otherwise} \end{cases}, \tag{3.15}$$

where $\tau_{det} = 50\%$ is the minimum overlap threshold. The intersection and union[8]

[8]We follow the convention found in the VOC source code that the union area is the area of the tightest axis-aligned bounding box that contains the (axis-aligned) bounding boxes from the detection and putative matching ground truth. Except in degenerate cases, this is an over-estimate of the true area of the union of the two input bounding boxes. This causes Eqn. 3.15 to be biased against simultaneous boundary misalignments in both the x and y directions, relative to misalignments in only one dimension.

areas are given by

$$\begin{aligned}
\mathrm{A}_{\text{intersect}}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) = & \max(0, (\min(\hat{x}_i + \hat{w}_i, x_j + w_j) - \max(\hat{x}_i, x_j))) \\
& \cdot \max\Big(0, \Big(\min\Big(\hat{y}_i + \hat{h}_i, y_j + h_j\Big) - \max(\hat{y}_i, y_j)\Big)\Big)
\end{aligned} \tag{3.16}$$

$$\begin{aligned}
\mathrm{A}_{\text{union}}(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) = & \max(0, (\max(\hat{x}_i + \hat{w}_i, x_j + w_j) - \min(\hat{x}_i, x_j))) \\
& \cdot \max\Big(0, \Big(\max\Big(\hat{y}_i + \hat{h}_i, y_j + h_j\Big) - \min(\hat{y}_i, y_j)\Big)\Big) .
\end{aligned} \tag{3.17}$$

Given the input data and matching just described, we can compute the entries for a 2×2 confusion matrix as

$$TP = \sum_{i=1}^{\hat{P}} \sum_{j=1}^{P} \mathrm{m}^*(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j) \qquad \text{(true positives)} \tag{3.18}$$

$$FP = \hat{P} - TP \qquad \text{(false positives)} \tag{3.19}$$

$$FN = P - TP \qquad \text{(false negatives)} \tag{3.20}$$

$$TN = W - TP - FP - FN \qquad \text{(true negatives)} \tag{3.21}$$

where W is the total number of classifier evaluations produced by the sliding window detector. These entries provide the sufficient statistics for a point on an ROC or precision-recall curve.

Detection Results

For our detection experiments, we use the same input images and annotations that were used to generate the *GrandCentral* classification test set (see Tab. 3.3). We compute the detection scores using a perspective-aware scanning window that searches for pedestrians who are $1.58m$ to $1.84m$ tall. In Fig. 3-24, we show the detection results from one frame. In this and other frames, false positives are most commonly caused by (a) parts of the building with pedestrian-sized features, (b) oblique shadows, and (c) poor localization. False negatives are most commonly caused by (a) people whose appearance is very similar to the background and (b) dense crowds.

In Fig. 3-25, we show the precision-recall curves for each of our classifiers when

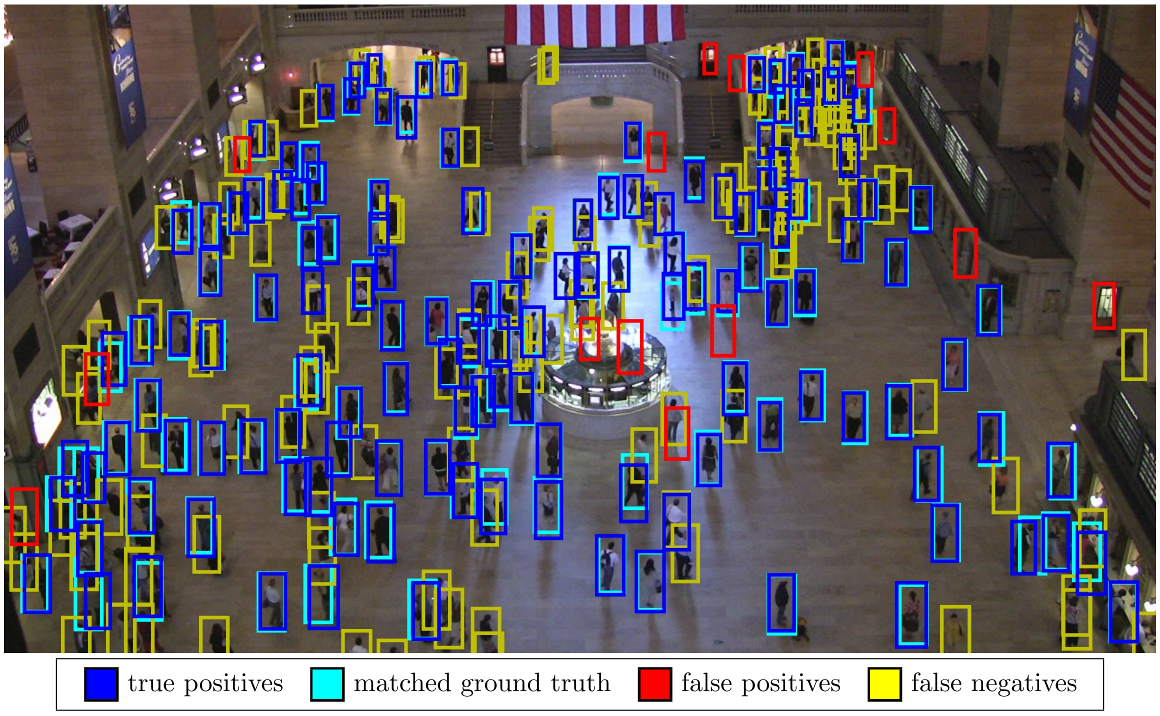


Figure 3-24: *HOG-based Detection Results:* For this particular image, there are 138 true positives, 14 false positives, and 129 false negatives.

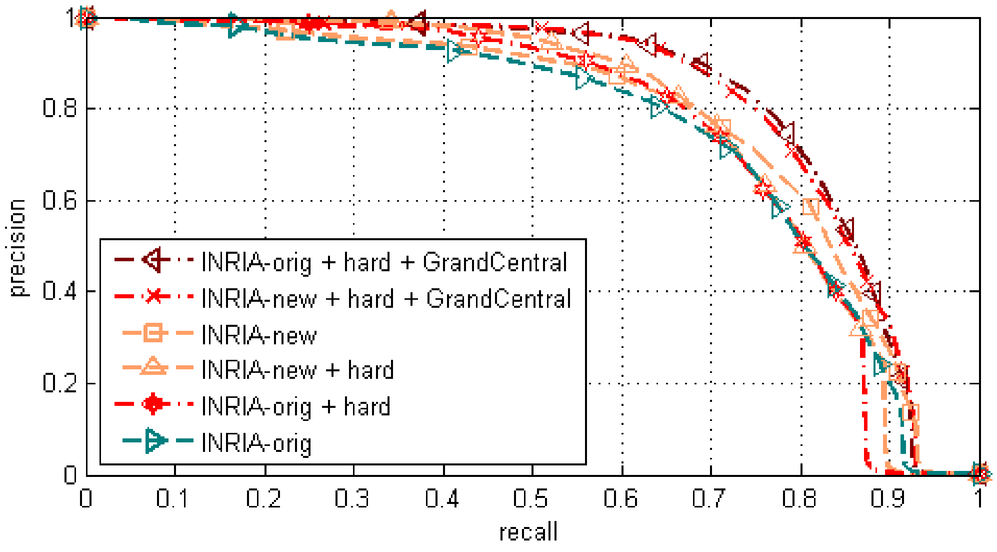


Figure 3-25: *Detection Precision-Recall Curves:* Here we show the precision-recall curves for running our full-frame detector on the *GrandCentral* test set. ROC plots are not shown because they are less meaningful for detection experiments. Due to the test protocol, detection results are less sensitive to the training set (compare the recall spreads to those in Fig. 3-21).

running our detection experiments. When we were discussing classification results in §3.4.3, the curves were created by varying the classifier's w_0 bias term (see Eqn. 3.8). For the detection experiments, we vary w_0 as well as the meanshift kernel bandwidth (see Eqn. 3.9). We then plot the achievable precision-recall curve using the ROC-transform method of Davis and Goadrich [25]. For the detection experiments, we do not show ROC curves because the high positive : negative example ratio makes those curves less amenable to visual analysis.

If we compare our detection results (Fig. 3-25) to the classification results on the *GrandCentral* dataset (lower plot in Fig. 3-21), several notable trends are observable.

- GrandCentral *classifiers still perform best:* In the 80-95% precision ranges, detection using *GrandCentral*-trained classifiers outperforms those using only *INRIA* data by approximately 10% in recall. Better results are to be expected because the training and test data are so similar.

- *Capped maximum recall:* For the detection results, a non-degenerate solution never achieves perfect recall. This is because we adjusted w_0 so that at most 5% of all windows could be considered as potential detections (see Eqn. 3.7). From a runtime perspective, more permissive values of w_0 than this result in prohibitively large false positive rates given our $O(\hat{P}^2)$ meanshift and non-maxima suppression implementations. Approximate implementations that approach $O(\hat{P})$ could be used (see Comaniciu and Meer[15]), but in a real system, one would probably not want to operate with such a permissive classifier that the final detection decisions are dominated by meanshift and non-maxima suppression.

- *Higher recall for most* INRIA*-only classifiers:* The *INRIA* and *GrandCentral* training sets use different annotation protocols, which greatly affects classification results when tested against independent windows (see Fig. 3-21). With the detection experiments, meanshift coupled with the VOC evaluation protocol allow us to overcome many of the annotation issues. If we chose a value higher than 50% for the minimum intersection/union threshold, τ_{det}, then we

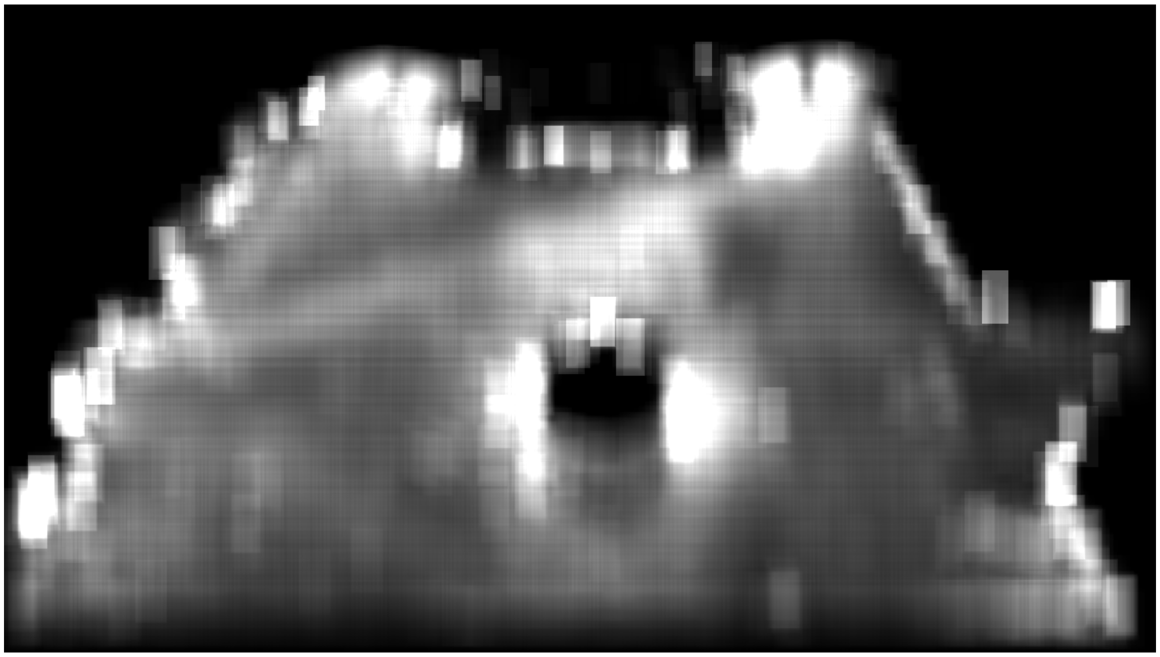


Figure 3-26: *Accumulated HOG Detection Windows:* In a manner similar to Fig. 3-9's, we visualize the likelihood of a final detection window overlapping each pixel, using the same test video that was used in Fig. 3-9(b).

expect that the recall would drop again and annotation issues would return to prominence.

- *Lower recall for the* GrandCentral *classifiers:* The *GrandCentral* training and test data were annotated using the same protocol, so the benefits seen when using *INRIA* classifiers for detection are applicable to the classifiers trained with *GrandCentral* data. For the classification experiments, all positive examples were aligned exactly by the hand annotations, effectively simulating perfect suppression of multiple detections. In the detection experiments, we are subject to an imperfect classifier as well as imperfect meanshift and non-maxima suppression.

If we examine the spatial distribution of foreground (Fig. 3-9(b) in §3.2) and the distribution of detections (Fig. 3-26), we see that the pedestrian detector is able to still find the pedestrians in the area between the information booth in the center of the image and the east stairway above it. That part of the video experiences significant glare.

One of the most common mistakes in the pedestrian detector is that it consistently believes that some building features are pedestrians. For example, the bright spot near the center right edge in Fig. 3-26 is due to the doorway on the far side of the exiting hallway (see Fig. 3-24 for a frame when the doorway is falsely detected).

We can also make direct comparisons to the experiments described in §3.2. In Fig. 3-27, we can see that only the strong pedestrian model is able to achieve more than 85% precision over a feasible range of parameters. Furthermore, recall that the background subtraction experiments only produce labelings for each block of 16×12 pixels. These plots effectively assume that we can optimally solve the difficult segmentation and grouping problem to produce individual pedestrian detections after doing background subtraction. Given the meanshift and non-maxima suppression in the HOG system, no such problems exist for it.

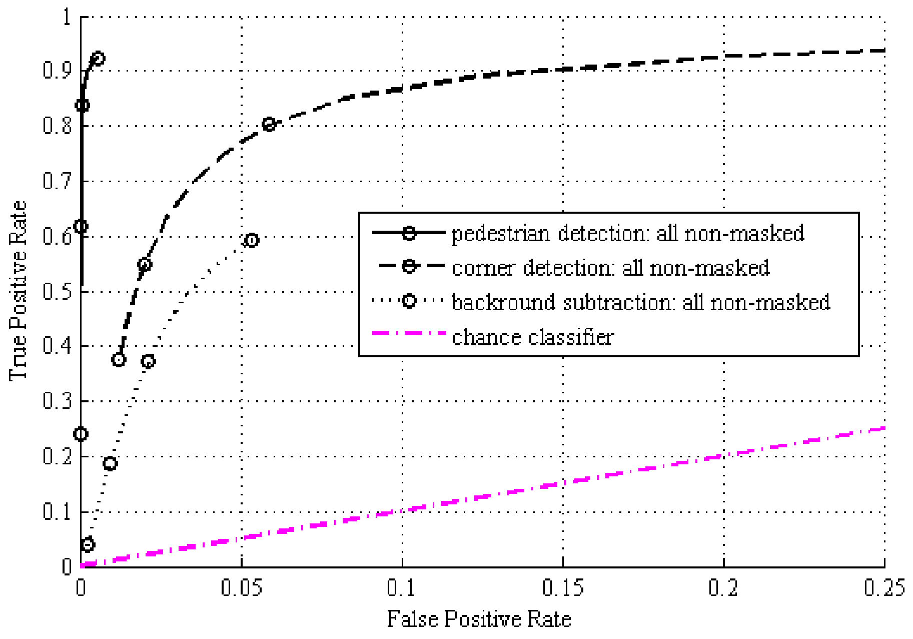
Broader Lessons

Before proceeding, we wish to highlight a few broader lessons about detection and localization in surveillance settings.

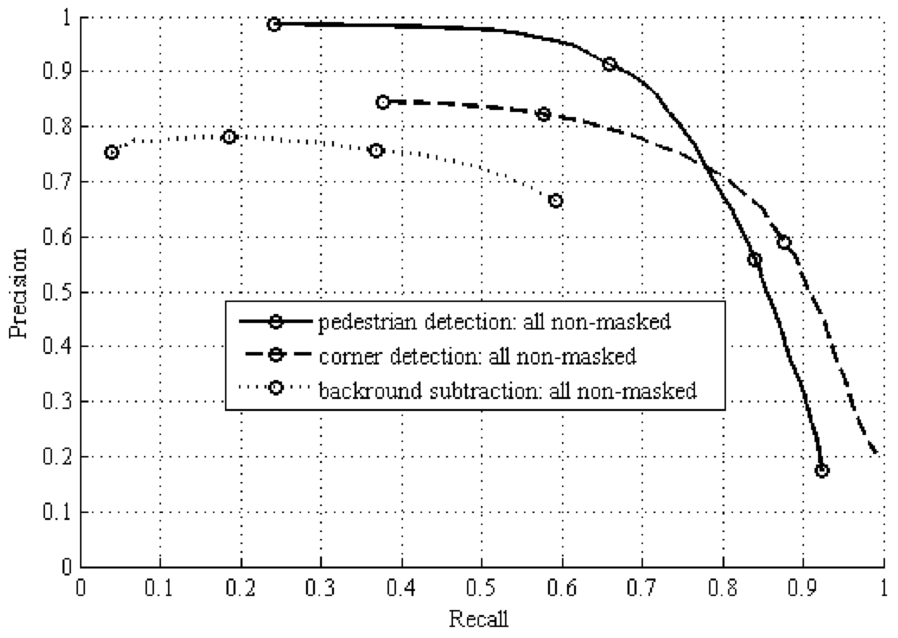
- *Well-trained strong models* can outperform generic weak models, as evidenced in Fig. 3-27.

- *Precise localization in training* is much less important in detection and localization experiments than classification ones, at least when one only requires approximate localization such as a 50% VOC overlap, as evidenced by the similarity of results in Fig. 3-25 of classifiers trained with *INRIA-orig* versus the more precise *INRIA-new* datasets. The reason is that as long as a detection fires close to the correct location, it is counted as a match. For more stringent localization requirements, we expect that the precision in the training set will be more important as well.

- As with classification experiments, *matching training and test distributions* improves the test results. In Fig. 3-25, we see that augmenting a generic *INRIA*



(a) ROC Curves



(b) Precision-Recall Curves

Figure 3-27: *Background Subtraction versus Pedestrian Detection:* Here we overlay the pedestrian detection results over those from background subtraction (first shown in Fig. 3-8) and feature point detection (see Fig. 3-13). The HOG detector produces far better detections. Although feature point detection has higher precision than the HOG detector for very high recall rates, the HOG detector's false positives are more randomly-distributed, making it still be more suitable for tracking applications.

dataset with annotations from the same scene (but still using a different viewpoint and a different camera) significantly improves the results. In general, it is known that the more representative a training set is of the test conditions, the better the detector should perform.

- *Strong models can be very slow.* As we will discuss in the next section, a single-threaded CPU-only implementation takes a minute per frame to compute (on high definition farfield video frames), which is prohibitively expensive. This is the case even though a very efficient linear SVM classifier is used. To use in realtime or even semi-realtime settings, one needs to (a) optimize the implementation, (b) find more efficient algorithms, or (c) find a way to exploit alternative resources. In §3.4.5, we will explore the last option.

3.4.5 Data-Parallel Implementation

Motivation

Before using our pedestrian detector for activity modeling applications in §4, we first visit the issue of runtime cost. Even after applying a number of key optimizations to the code provided by Dalal and Triggs[9], it takes nearly one minute to run the pedestrian detector on a single frame using a latest-generation CPU. In this section, we will show that porting key elements of the pedestrian detector code to use the data-parallel hardware on a consumer-grade graphics card results in processing rates that approach that of CPU-only background subtraction.

Given a set of computational tasks, the throughput of a system is the quotient of the number of tasks being processed in parallel and the time to process a single task. One can improve the performance by either increasing the amount of parallelization and/or decreasing the latency between starting an individual task to completing it.

[9] The primary algorithmic change was to locally adapt the scale search range based on perspective calibration. This reduces the search space for the scanning window classifier, by reducing $\hat{P}$ it allows meanshift and non-maxima suppression to run faster, and it reduces the false positive rate in the final output. We also applied various micro-optimizations such as avoiding unnecessary memory copies and restructuring loops for better cache performance. These optimizations do not change the final output, but have a significant beneficial impact on runtime.

Historically, mainstream CPU manufacturers have primarily focused on improving performance by lowering latency because the more quickly a task can be completed, the more tasks that can be done in a fixed amount of time. Process and architectural improvements have allowed exponential growth rates in single-threaded processor speed for decades. Unfortunately, we are nearing the end of significant speed improvements of this type [104]. The smallest silicon features are now less than 100 atoms thick, making significant issues out of quantum effects like tunneling. Furthermore, high clock speeds tend to require high power consumption and in turn make heat dissipation an increasing challenge. It is unrealistic to expect improvements in CPU clock speed to yield the $1,500\times$ speed improvement required for our HOG code to run in true real-time.

Fortunately, a secondary market of computational devices has grown out of a need for better special effects and physics computations in entertainment software. Most of the difficult computational tasks in this realm are "embarrassingly-parallelizable": a massive number of tasks need to be performed which can be computed independently from other tasks. As a result, graphics processing units (GPUs) are designed to improve throughput by increasing the number of tasks being handled in parallel. Furthermore, demand in the entertainment sector has resulted in GPUs shifting from having fixed hardware pipelines that only do polygonal rendering to having more general computation abilities. Both the generality of the hardware and toolkit support have improved to the point that general computation can now be performed on many modern GPUs. In early 2009, a typical high-end consumer GPU card has a few hundred floating point units (FPUs) running at 1.4GHz and has a peak performance of nearly 1 TFLOPS (trillion floating point operations per second). In contrast, a high-end heavily overclocked 4GHz quad-core consumer CPU has difficulty achieving 10 GFLOPS under ideal real-world conditions [7]. At the time this thesis was written, the cost of a single one of these high-end CPUs is $3\times$ to $4\times$ the cost of consumer-grade 1 TFLOP GPU or about the same price as an industrial-grade 1 TFLOP GPU. The two orders of magnitude difference in throughput/dollar allows a whole class of problems to be considered that would otherwise require economically infeasible large

CPU clusters.

Implementation Choices

Several portions of the HOG algorithm are embarrassingly-parallelizable[10], most notably image scaling, gradient computations, building of window descriptors (§3.4.2), classifying the descriptors (§3.4.3), and meanshift. All but the last of these also exhibits very predictable memory locality. Each output element relies on a fixed set of input data and memory can be organized so that these data lie in a small number of contiguous memory blocks. We have ported each of the algorithms with good locality to run on NVIDIA graphics cards that support the Compute Unified Device Architecture (CUDA). We use a similar code structure to that of Wojek *et al.* [122]. Due to licensing restrictions, we were not able to obtain direct access to their code.

Our code is written the “C for CUDA” language[11], which is effectively a superset of C++. Special tags are used to mark C++ code that will run on the GPU, and an NVIDIA-supplied preprocessor divides the source into portions that run on the CPU and those that run on the GPU. The CPU portions may be compiled by any standard C++ compiler, and the GPU portions use NVIDIA's compiler. To have performant CUDA code, most of the programmer effort is spent on optimizing memory access patterns. An experienced software engineer should typically expect to spend about the same amount of time writing optimized CUDA code as they would spend writing optimized C++ code with assembly language compiler intrinsics. For algorithms that are well-adapted to the GPU architecture, it is common to have a 100× speedup for CUDA code relative to CPU code.

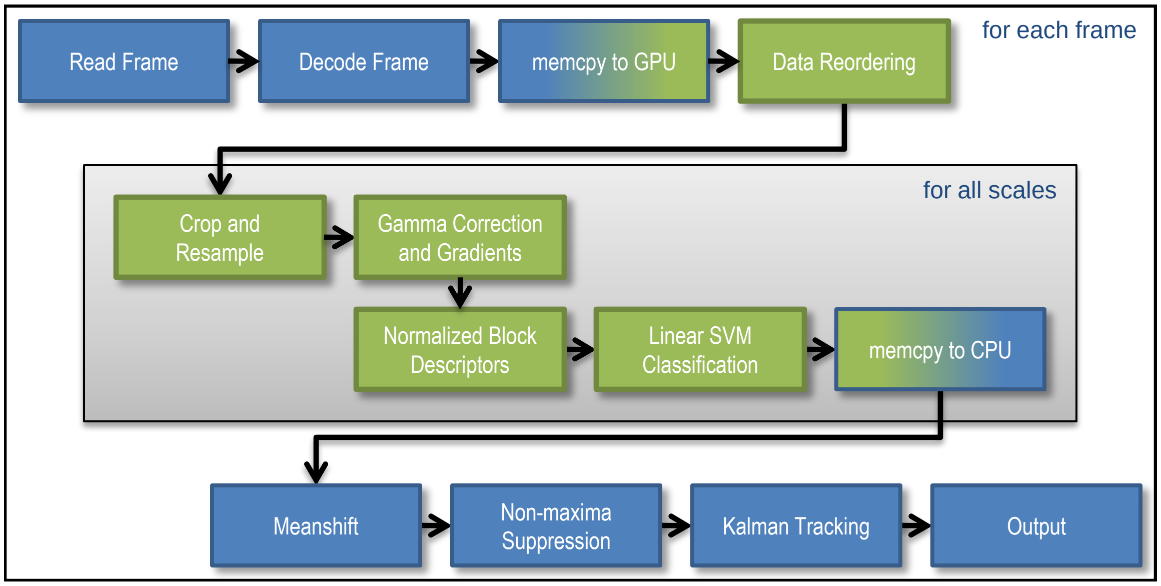


Figure 3-28: *GPU-assisted HOG Tracking Pipeline:* These are the stages in our pedestrian tracking application with a GPU-assisted HOG detector. Blocks in green are executed on the GPU. Blue blocks are executed on the CPU.

Implementation Details

For those interested in the implementation details of our CUDA port, we provide some high-level information here, drawing connections to the implementation of Wojek *et al.* [122] where appropriate. This section will be most meaningful to those with a prior understanding of the CUDA programming model. Good sources of information include the *NVIDIA CUDA Programming Guide* [17] and *GPU Gems 3* [75]. We are separately distributing the source code for the "read frame" through "non-maxima suppression" stages shown in Fig. 3-28[12].

As indicated in Fig. 3-28, we first read input images or video frames and decode them on the CPU. The raw RGB images are transferred to the GPU synchronously. We do not spatially smooth padded image borders as done by Dalal and Triggs and

[10] "Embarrassingly-parallelizable" is a technical term for problems in which there are a very large number of outputs and each output can be computed independently of all others. Algorithms can be written for these problems that approach a linear speedup in the number of available parallel processors.

[11] "C for CUDA" is typically called just CUDA since it is the only language currently used in a widespread fashion that exploits the Compute Unified Device Architecture (CUDA) developed by NVIDIA.

[12] See the thesis author's website at `http://people.csail.mit.edu/dalleyg` for a link to the CUDA-enabled HOG source code.

by Wojek *et al.* because we are using very large input images with small pedestrians, thus we care less about detecting pedestrians that overlap an image boundary. An efficient GPU-based smoother could be added if needed, with a negligible overall speed impact.

Like Wojek *et al.* , once an input frame is on the GPU we reorder its data so that we can take advantage of dedicated hardware on NVIDIA cards for caching access to 2D arrays.

As an extension to the pipeline, our application is perspective-aware. This allows us to choose a collection of scales for each image row that are feasible given camera calibration information. Thus when we resample the input image to test a given window detector size, we extract only an image band. In the Grand Central scene, pedestrians vary in image size by a factor of 2. By adding perspective awareness, we are able to avoid the cost of running the detector with infeasible scales. More importantly, this helps us avoid obvious false positives such as 4 meter tall people. We have made this extension to both the CPU and GPU implementations, so the speedup numbers we discuss in this thesis are fair.

In our timing test application discussed in the next section, we use nearest neighbor resampling with bilinear interpolation to rescale the input image. This is the same type of rescaling used in our CPU code, the code of Dalal and Triggs, and by Wojek *et al.* We have also implemented a resampler that uses Lanczos interpolation [108]. The Lanczos implementation is half as fast and trades off having some ringing artifacts with fewer aliasing artifacts.

In the implementation of Wojek *et al.* , they next do color decomposition, data type conversion, and gamma compression in one kernel, then they compute the color gradients in three additional kernels. CUDA code is compiled so that a single function, a kernel, is evaluated on all processors simultaneously. All caches are flushed between kernel calls, so it can be advantageous to combine multiple small kernels when each has similar memory access patterns. In our implementation, the decomposition and type conversion was done back when we did the data reordering. Because the memory access patterns are amenable to it, we combine the gamma correction and gradient calculations into a single CUDA kernel. Wojek *et al.* chose to use a separable convolution implementation for their gradients, which is optimal for large convolution stencils. Fortunately, a faster and much more straight-forward solution is available because Dalal and Triggs found that unsmoothed central difference operators, *i.e.* $(-1, 0, 1)$, work best. This unified kernel is one of the least time-consuming portions of our pipeline.

The most important difference between our implementation and that of Wojek *et al.* is the way we calculate and store the block-level HOG descriptors. Because GPUs are massively parallel, the most common bottleneck is memory access. The hardware tends to have very high memory bandwidth but unfortunately latencies are measured in several hundred clock cycles. To handle latency better, GPUs are able to allocate hundreds of thread contexts on each processor. When one thread is blocked on memory, another waiting thread can be resumed in a single clock cycle. A common optimization technique is to try to maximize the "occupancy": the fraction of hardware supported thread contexts with a thread allocated to it. Using too many registers or user-managed cache are the primary reasons a CUDA kernel doesn't achieve full occupancy.

For our block descriptor kernel, we chose to focus on minimizing the number of data transfers to main memory rather than maximizing the occupancy. Wojek *et al.* have a 50% occupancy for their block histogram kernel and a 67% occupancy for their block normalization kernel. We compute the cell histograms and then normalize them in a single kernel that has only 18.8% occupancy. Each GPU multiprocessor has a collection of threads that work collaboratively. Each thread is responsible for computing a single block descriptor. This computation is done by having the collection jointly page in horizontal strips of gradients into a user-managed cache called "shared memory." Each thread then individually does the appropriate histogram voting. Because we have a regular memory access pattern, we can use a special type of cached memory called "constant memory" to store the spatial voting kernel. The block histograms are also stored in shared memory. After all the histogram voting has been performed, each thread normalizes its descriptor, then the collection of threads collaborate to efficiently write the descriptors out to the GPU's main memory.

Our memory access pattern minimizes the number redundant gradient loads. By normalizing the block descriptors immediately, we avoid costly memory transfers to the GPU caches. Because our algorithm requires less total bandwidth, we see better performance despite having a much lower occupancy.

For the SVM evaluation, each thread is responsible for a single detector window classification score. We have coordinated the memory layout of the block descriptors with this step so that we can efficiently stream in contiguous blocks of memory using a collection of threads. As the memory blocks are streamed, each thread updates a running dot product. Because we have ordered the data well, all threads in a collection are guaranteed to be accessing the same element of the SVM classification vector. This allows us to use the constant memory at full speed. Wojek *et al.* used a different memory access pattern that is less efficient.

We note that some of the design decisions of Wojek *et al.* may have been driven by the desire to avoid some very severe penalties on their graphics card (NVIDIA 8800 Ultra) for accessing the GPU's main memory in suboptimal ways. Newer cards based on the Tesla architecture have relaxed many of these penalties, allowing more varied memory access patterns.

Like Wojek *et al.* , we transfer the SVM scores to the CPU and use it to compute the meanshift modes and perform non-maxima suppression. We believe a GPU-based meanshift implementation could be made that is at least 10× faster than on the CPU.

When processing videos, we also do Kalman tracking with greedy data association, as was described in §1.4.

In summary, when we produced our CUDA port of the HOG code we found it beneficial to concentrate on minimizing the amount of memory traffic between the large main main memory of the GPU and the on-chip user-managed caches. We minimize this traffic by arranging our data so that contiguous memory blocks can be loaded whenever necessary and by organizing our computation threads so they can maximally share cached data with other threads. We found this emphasis on memory efficiency to be more important that trying to maximize the number of allocated thread contexts (occupancy). Although many of the particular optimizations may change with future hardware, we expect that the trend of the last several decades in hardware design will continue: main memory continues to have higher latencies over time relative to the clock speed of the compute cores. As such, it will continue to be important to optimize algorithms so they can exploit local memory access patterns, at least when the best runtime performance is desired.

Timing Results

We now examine the runtime benefits of our CUDA port. For all the performance tests, we used a mid-range high-performance desktop machine with the following configuration:

- *CPU:* Intel Core i7 920 (2.66GHz, 4 cores)
- *GPU (graphics card):* NVIDIA GeForce GTX 280 (1GB of RAM, 240 FPUs)
- *OS:* Debian Lenny, 64-bit
- *Compiler:* gcc 4.3.2[13]
- *CUDA:* version 2.1

All tests were performed on the video from which we extracted the *GrandCentral* test data.

In Tab. 3.5, we show timing results for an instrumented test version of our CPU code and our CUDA-based GPU port. We discuss each row here.

- *read input (CPU):* In both cases, we read a single JPEG image and decode it on the CPU. Our GPU port is fast enough that this alone takes up 17% of the total runtime. Note that our full tracking application is able to read video files and it decodes frames faster than in this test application.

[13]Some portions of the GPU interfacing code were built with gcc 4.1.3 due to limitations in NVIDIA's compiler.

Processing Step	Time (CPU Impl.)	Time (GPU Impl.)	GPU Impl. Speedup
read input (CPU)	68.0 ms (0%)	68.0 ms (17%)	
GPU resizer setup		18.1 ms (5%)	
resize	1,045.8 ms (4%)	43.0 ms (11%)	24.3×
gradients	5,636.2 ms (24%)	34.4 ms (9%)	164.0×
normalized block descriptors	13,412.3 ms (57%)	137.2 ms (35%)	97.7×
window classification	3,159.1 ms (14%)	31.4 ms (8%)	100.6×
cleanup	23.8 ms (0%)	45.5 ms (12%)	0.5×
detection (CPU)	16.2 ms (0%)	15.0 ms (4%)	1.1×
TOTAL	23,082.4 ms	392.3 ms	58.8×

Table 3.5: *CUDA-HOG Speedup Analysis:* "CPU Impl." refers to our optimized CPU-only version of Dalal and Triggs' code. "GPU Impl." refers to our CUDA-based GPU implementation. The GPU time is wall time for each step, including CPU time, GPU time, and data transfers.

- *GPU resizer setup:* When running the CUDA version, we reorganize the input image into a data structure optimized for 2D locality and transfer it to the GPU's memory ("GPU resizer setup"). This allows repeated resizes of the same input image to be done faster.

- *Per-Scale Steps:* We run the scanning window detector over a range of image scales, one at a time. We show the total time of each step, summed across all scales.

  - *resize:* Because this operation is bound by memory bandwidth on the GPU, we only achieve an 24.3× speedup. When using Lanczos interpolation instead of nearest neighbor, as in our full tracking application, the speedup is reduced to 11.3× (not shown in the table).
  - *gradients:* GPUs are specifically designed to optimize highly-localized independent operations like gradient computations. We see a 164.0× speedup.
  - *normalized block descriptors:* For both the CPU and GPU implementations, the main bottleneck is computing and normalizing the block histograms. We spent most of our development effort on optimizing the memory access patterns for this step, resulting in a 97.7× speedup.

  - *window classification:* When evaluating the linear SVM, we dynamically extract the right block descriptors to simulate the formation of each window descriptor. Subject to round-off errors, the computed results are identical to what would be seen if window descriptors were explicitly constructed, but this offers significant performance improvements. We see a 100.6× speedup on the GPU. For the CUDA port, these times include the cost of transferring the results back to the host's system memory.
  - *cleanup:* Freeing memory takes a surprisingly non-trivial amount of time (12% of the overall runtime) for the CUDA port. Reusing data structures should make this cost go to zero. This phase is less costly in our full tracking application.

- *detection (CPU):* We have not ported meanshift or non-maxima suppression to the GPU, so this step is run on the CPU for both implementations. In scenes with more pedestrians, the cost of this step becomes more significant. The GPU version of the code takes slightly longer because there are a larger number of positive detections it must prune. This difference is due to some subtle changes in the training set which we will discuss momentarily.

Overall, we achieve a 58.8× speedup in this simplified test application[14].

We have also extended Dalal and Triggs' `classify_rhog` application to do GPU offloading as well as frame-to-frame tracking, as shown in Fig. 3-28. The architecture of that implementation makes it less amenable to profiling individual processing stages, so we do not provide a breakdown of its timing like we have done for the speed testing application. The GPU-enhancements and optimizations in it consistently result in a 76× speed boost when processing real videos.

[14]Care should be taken when comparing performance numbers to the GPU implementation by Wojek *et al.* [122] (34× speedup) and Zhang and Nevatia [123] (20×). Based on a collection of informal experiments and personal correspondence with Wojek, we estimate that his speedup would not change significantly on our hardware: our CPU's increased performance relative to theirs is roughly equivalent to the performance boost of our GPU versus theirs. On the other hand, Zhang and Nevatia used an older workstation-class graphics card with a very different architecture, so it becomes difficult to estimate its relative performance without being able to do direct comparisons. Various technical and licensing concerns with have prevented us from making that direct comparison.

	Pipeline Steps Included in the Timing Measurements				
Algorithm	Input and Preproc.	Low-Level Detection	Object Detection, Tracking, and Output	Time per Frame	Frames per Sec.
background subtraction (w/ MRF)	✓	✓		0.37 s	2.71
background subtraction (no MRF)	✓	✓		0.14 s	7.11
pedestrian detection (CPU)	✓	✓	✓	51.79 s	0.02
pedestrian detection (GPU)	✓	✓	✓	0.68 s	1.47

Table 3.6: *Runtime Performance Comparison:* Here we show the runtime costs of the various algorithms discussed in this chapter. Note that all of these are operating on high-definition 1920 × 1080 video frames. Timing information for the pedestrian detection is for our actual full tracking application, not the simplified timing application used to generate Tab. 3.5. The GPU-enabled pedestrian detector is within a factor of 2 of the runtime cost compared to a multithreaded background subtractor with an MRF, even though the latter only produces foreground blobs and not final segmented pedestrian detections.

Processing Step	Time / Frame
input and preprocessing	74 ms
Mahalanobis distance map	39 ms
MRF classification	228 ms
MoG update	27 ms
TOTAL	369 ms

Table 3.7: *Background Subtraction Timing:* Here we show additional timing information for doing background subtraction on our primary *GrandCentral* test video. This application is multithreaded. Decoding of video frames is performed in a background thread. We also used multithreaded implementations of the Mahalanobis distance map calculator, a Gibbs-sampling MRF solver, and the MoG updates.

In Tab. 3.6, we show timing information for each of the detection algorithms discussed in this chapter, so we can see how fast our HOG-based detector is in comparison.

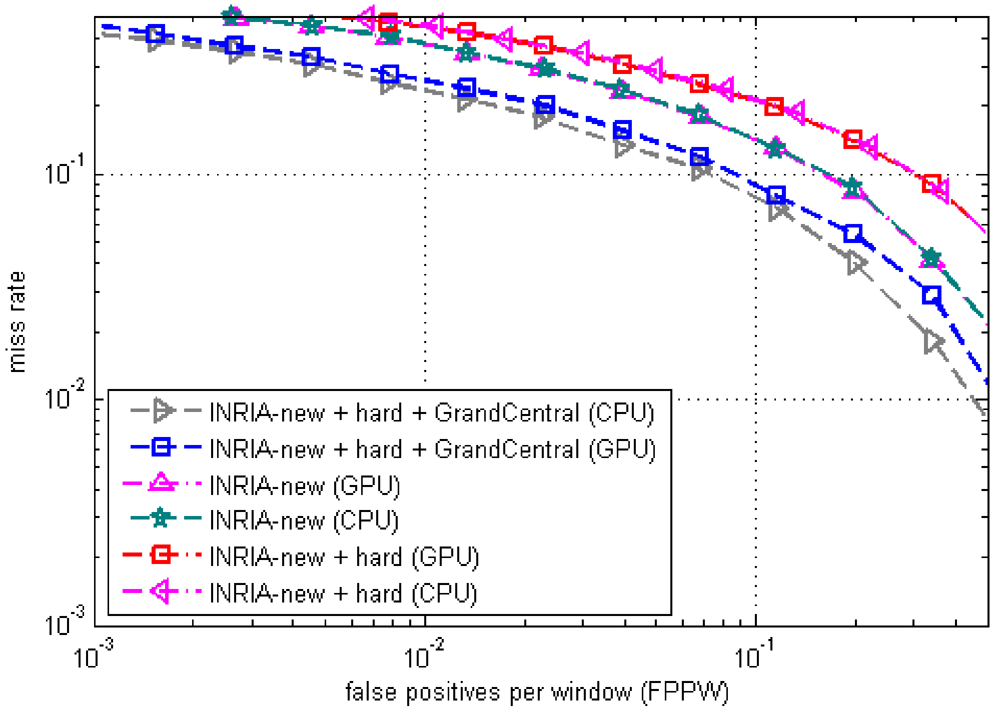
In the first rows, we see that background subtraction is able to process this high-definition video at 2.71 frames per second. This is equivalent to 18.3 fps on a VGA-resolution (640 × 480) video. The primary bottleneck is the MRF foreground/background classifier. If we instead use simple thresholding, the application performance approaches 7.11 fps (equivalent to 48 fps on VGA frames), as seen in the second row. We note that the MRF produces much cleaner foreground masks than simple thresholding. To provide extra context, in Tab. 3.7 we show timing information for the parts of the background subtraction application. Because the scene is dense enough that a sophisticated segmentation algorithm is needed to robustly track pedestrians, the following costs are not included: detecting actual pedestrians from the foreground blobs, tracking pedestrians, and output.

In the last two rows of Tab. 3.6, we show average per-frame timing information from processing an entire 1 hour video using our full CPU and GPU HOG implementations, respectively. Unlike the simplified application used to produce Tab. 3.5, this one explicitly produces full window descriptor vectors for the CPU implementation. Across the whole video, it also has a slightly lower density of pedestrians than the single frame used in generating Tab. 3.5, and this lowers the relative cost of the $O(\hat{P})$ meanshift and non-maxima suppression implementations. For these reasons, the GPU pedestrian detection is reported as being 76× faster than the CPU in contrast to the 58× speedup seen in Tab. 3.5. We note that our pedestrian detection application is only 1.8× slower than multithreaded background subtraction with an MRF[15].

Quality of Results

We chose to write our GPU port so that it computes the same values as the CPU implementation, even if the execution structure of the low-level code is significantly different. To demonstrate that we achieve the same performance, we show ROC and precision-recall curves for one representative set of classification experiments in Fig. 3-29. When using the *INRIA-new* and *INRIA-new + hard* training sets, there is no discernable difference between our GPU and CPU results. We do note that there is a small performance drop for the GPU implementation when *GrandCentral* training data is added. This is due to an implementation detail that currently prevents us from sampling our training data as finely for the GPU implementation.

[15]We believe that a GPU implementation of background subtraction with an MRF could easily achieve a 10× to 100× speedup relative to the current CPU implementation.



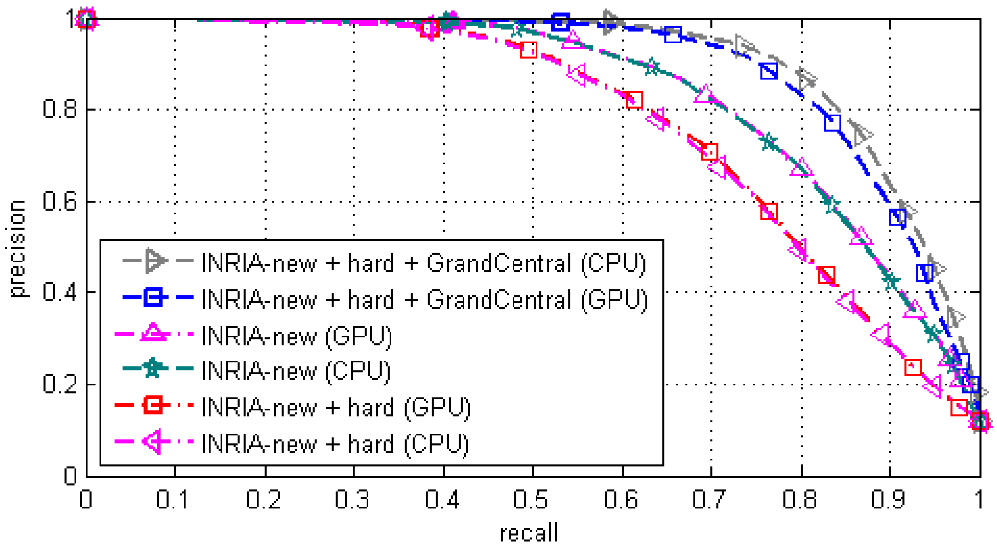


Figure 3-29: *CPU versus GPU Classification Results:* Here we show selected classification results on the *GrandCentral* test set. The CPU results match the *INRIA-new* ones shown in Fig. 3-21.

HOG Summary and Future Work

In §3.4, we have demonstrated that using a HOG-based pedestrian detector yields good results even on our very challenging *GrandCentral* test set. By porting key portions of the algorithm to take advantage of the extra processing power available on consumer-grade graphics cards, we are able to speed up the detector by 58× to 76× relative to a CPU-only implementation. This puts us within a factor of 2 of the speed of multithreaded background subtraction with an MRF and it makes it practical to do off-line analysis of high-definition video or quasi-realtime analysis of standard-definition video, even in the presence of moderately dense crowds.

Because of constraints imposed by physics on hardware manufacturers (the speed of light, quantum effects, *etc.*), much of the future computational throughput capacity will take the form of offering more concurrency. Because many vision problems are embarrassingly-parallelizable, their solutions can take advantage of this hardware. In these cases, it is common to see throughput increases of two orders of magnitude, as we have seen in most of the HOG steps we have ported to the GPU.

A number of additional optimizations would likely yield additional speed improvements:

- *Concurrency improvements*
  - *Multiple GPUs:* Multiple GPUs could be installed in a system and have input frames assigned to them in a round-robin fashion. This should yield a near-linear speedup in the number of GPUs.
  - *Asynchronous I/O:* A non-trivial amount of the algorithm time is dedicated to transferring data back and forth between the host memory and the graphics card. Recent GPUs allow the user to simultaneously have the GPU be executing a function while the CPU loads the next input and/or transfers existing results back to main memory.
  - *CPU-GPU pipelining:* In our current implementation, either the CPU or the GPU is actively computing results. Never are both active. We could pipeline the implementation so that

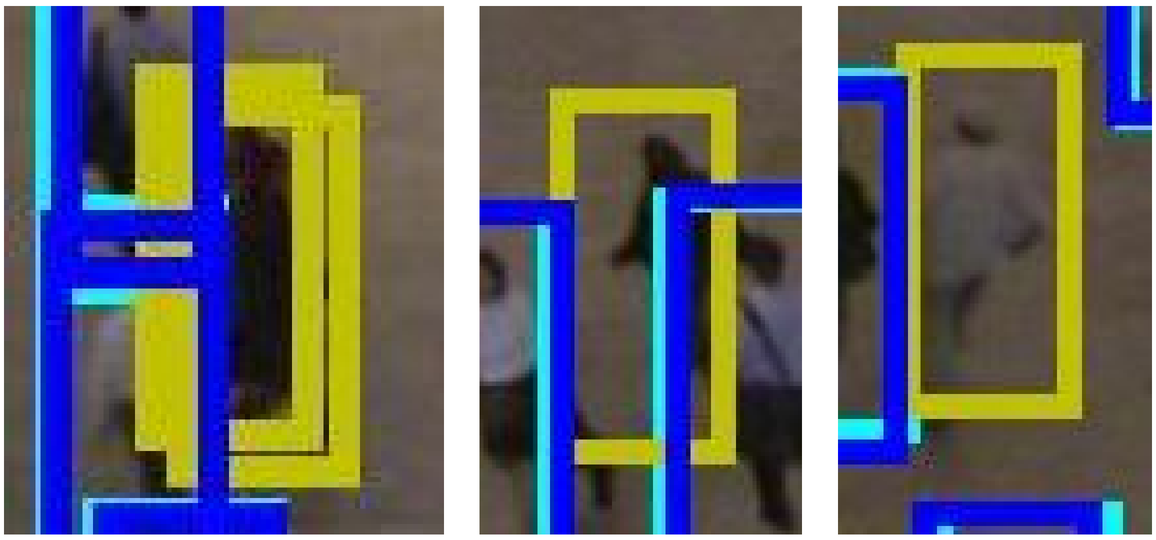
        * decoding frame t can happen while
        * frame $t-1$ is being transferred to the GPU,
        * GPU-based detection is happening on frame $t-2$,
        * frame $t-3$'s detection results are being transferred off the GPU, and
        * meanshift, non-maxima suppression, tracking, and output is being done on frame $t-4$'s detections

- *More offloading to the GPU*

    - *Decoding:* We currently decode the input images and/or video on the CPU. Some GPUs have dedicated hardware to accelerate decoding. Since the encoded stream is smaller than decoded frames, doing the decoding on the GPU would not only offload computation, but it also reduces memory transfer costs.
    - *Meanshift:* An initial GPU port of the meanshift code with a $10\times$ speedup should be straightforward since each meanshift track can be computed independently. Extra thought would be required to devise a way of effectively sharing input data across multiple tracks to minimize data transfers between graphics card memory and on-chip GPU caches. If this extra optimization is possible, such an algorithm could be expected to improve the speedup to $100\times$. The most difficult portion would be finding an efficient way of grouping all co-incident final kernel locations.
    - *Non-maxima suppression:* With care, it may be possible to have an efficient and accurate non-maxima suppression implementation on the GPU (*e.g.* Neubeck and Van Gool [74] have developed parallel solutions to similar non-maxima suppression problems).

- *Algorithmic changes*

    - *Integral Histograms:* If the Gaussian spatial voting kernel were eliminated, we would be free to use an integral histogram to efficiently compute



(a) Occlusion (b) Perspective Distortion (c) Low Contrast

🟦 true positives 🟦 matched ground truth 🟥 false positives 🟨 false negatives

Figure 3-30: *Common False Negatives:* The most common sources of false negatives for the single-frame pedestrian detector are (a) missing or spurious gradients due to occlusion, (b) image skew induced by perspective distortion, and (c) low contrast against the background. The problems with low contrast are less prevalent than occlusion and distortion issues.

arbitrarily-sized cell descriptors. In making this suggestion, we do note that an efficient GPU cascade detector would be more complex than the CPU implementation by Zhu *et al.* [130]. Current GPUs are highly optimized for regular memory access patterns with minimal code branches, but detector cascades tend to randomize the access patterns and branching decisions.

In inspecting the detection results on the *GrandCentral* videos, the two biggest challenges for the detector are occlusion and perspective distortions. See Fig. 3-30 for examples. Here are some ideas we have on mitigating those challenges.

- *Explicit occlusion reasoning*
  - Simulate occlusions in training data.
  - Do Hough voting à la Leibe, Seemann, and Schiele [63].

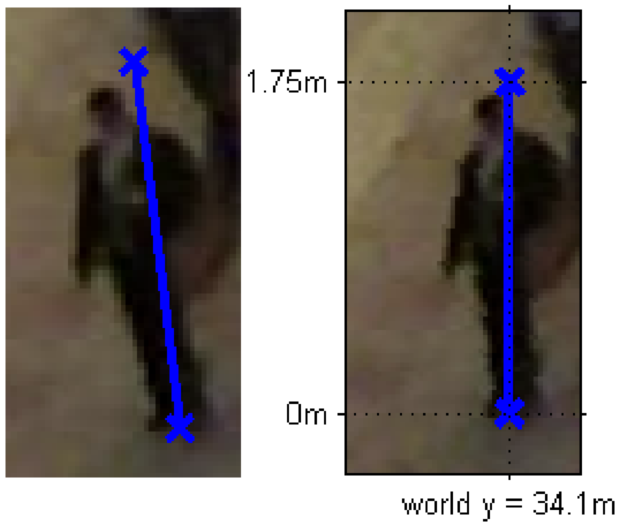


Figure 3-31: *De-skewing Issues:* A potential solution to some perspective distortion problems is to reproject the images to a plane perpendicular to both the ground plane of the and the optical axis. As evidenced by the ragged edges in the arms and legs in the right image, this would induce many false gradients. The upper "$\times$" markers do not exactly coincide with the tip of the man's head due to calibration error and his actual height versus the assumed $1.75\,m$.

- *Perspective awareness*
  - Create different classifiers (or bias the training data) for different viewpoints. With a calibrated camera, one can automatically pick the right classification vector.
  - The largest problem with perspective distortion is skew induced by a camera whose optical axis is not parallel to the ground plane. Directly deskewing the image does not work well because it introduces very strong false edges (see Fig. 3-31). One solution is smooth the image after reprojection, but that reduces the effective image resolution. Another solution is to take the gradient vectors and use the camera calibration to undistort their location and direction before voting.
  - It may be beneficial to use a "Latent SVM" version of the HOG detector similar to that of Felzenszwalb, McAllester, and Ramanan [27]. After using a standard HOG window descriptor as a root detector, they allow for some

deformation by having a second collection of HOG features at a higher scale. These finer features can have their positions adjusted with respect to the root detection, allowing for local deformations with an associated classification penalty.

3.5 Summary

In this chapter, we have demonstrated the difficulties of using background subtraction for tracking pedestrians in large scenes with harsh lighting conditions and dense crowds. A naïve feature point detector does yield better results on the particular scene studied here, but its performance is still sub-par. We then showed how a consumer-grade graphics card can be used to accelerate a strong-model pedestrian detection algorithm. Our implementation achieves an overall 76× speedup on real-world tracking relative to an optimized CPU-only implementation. On a canned single-frame test, it is 58.8× faster. At approximately 1 frame per second, our implementation is nearing the realm of real-time processing and we estimate that it provides double the speedup of the fastest existing implementation. We have also provided suggestions on how to further improve the runtime performance.

Chapter 4

Scene Activity Modeling

This chapter contains previously unpublished joint work with Xiaogang Wang.

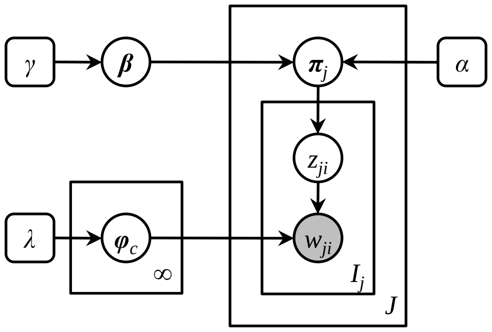
To help demonstrate the utility of our pedestrian detection approach, we now show that we can use it to generate an activity model that is more semantically meaningful than one produced by a previous method using sparse optical flow. The model we use is a Hierarchical Dirichlet Process (HDP) of the form used by Wang *et al.* [115][1].

We will first review HDP models and how Wang *et al.* use them to cluster trajectory modes. Using tracked feature points of the form used in §3.3, we are able to spatially segment the scene, but we are not able to model co-located activities with different moving directions (§4.2). We then show how tracks based on strong-model pedestrian detections yield much more meaningful clusters in §4.3.

4.1 Hierarchical Dirichlet Process Model

Here we briefly review the Hierarchical Dirichlet Process (HDP) model. For an in-depth discussion on the model, we refer the reader to the detailed tutorials by Teh *et al.* [105] and Sudderth [102]. Orbanz and Buhmann [79] have written an accessible paper on simpler Dirichlet Process mixture models coupled with Markov Random Fields.

[1]Most of Wang *et al.* [115] discusses a Dual-HDP model that adds an extra layer of clustering. We only compare against the standard HDP because it (a) has fewer parameters, (b) is empirically more robust to tracking errors, (c) is easier to analyze, and (d) can be learned more efficiently.



Variable	Intuitive Description	Formal Description
$\boldsymbol{\beta}$	Frequency of each activity cluster, across the whole dataset	$\boldsymbol{\beta} \sim \text{GEM}(\gamma)$
γ	Bias toward concentrating $\boldsymbol{\beta}$'s mass onto a few activity clusters	user supplied scalar
$\boldsymbol{\phi}_c$	Activity cluster: frequency of each quantized observation in cluster c	$\boldsymbol{\phi}_c \sim \text{H}(\lambda)$
$\text{H}(\lambda)$	The base Dirichlet distribution, a pseudo-count prior for multinomials. Typically, a uniform prior is used.	user supplied prior
λ	Bias for how much $\boldsymbol{\phi}_c$'s samples should resemble the base distribution. This is a multiplicative factor on the pseudo-counts.	user supplied scalar
$\boldsymbol{\pi}_j$	Frequency of each activity cluster in trajectory j	$\boldsymbol{\pi}_j \sim \text{DP}(\alpha, \boldsymbol{\beta})$
α	Bias for how much each $\boldsymbol{\pi}_j$ should resemble $\boldsymbol{\beta}$	user supplied scalar
z_{ji}	Index of the cluster chosen for observation i of trajectory j	$z_{ji} \sim \boldsymbol{\pi}_j$
w_{ji}	A single observed position and motion direction of trajectory j, quantized, indexed by i	$w_{ji} \sim \boldsymbol{\phi}_c \vert c = z_{ji}$
J	Number of observed trajectories	data dependent scalar
I_j	Number of quantized observations in trajectory j	data dependent scalar

Figure 4-1: *Hierarchical Dirichlet Process (HDP) Graphical Model:* Above we show a graphical model representation of a Hierarchical Dirichlet Process. This model is an infinite mixture of multinomials. The plates represent repeated structure of independent random variables. In the table below the figure, we summarize the role of each parameter and random variable.

The primary attraction of HDP models is that they provide an elegant and principled way of representing mixture models with countably infinite mixture components. Because they are posed in a Bayesian graphical model framework, they can readily be extended to model many more complex distributions. Inference can be done via Gibbs sampling or (sometimes) variational methods. The primary drawbacks are that Gibbs sampling can be very slow and that learning is only known to be tractable when the mixture component distributions are either Gaussians or multinomials. We will be using Gibbs sampling with base multinomial distributions.

Following the convention of Wang *et al.* [115], we model an individual person's trajectory as being composed of an unordered collection[2] of observations: their positions in the scene and direction of motion.

4.1.1 Our Definition of Observations

In Fig. 4-1, each of the J independently-observed trajectories is indexed by j and has I_j observations, $\{w_{ji}\}$, indexed by i. The observations are discrete tuples of the form

$$w_{ji} = (\breve{x}_{ji}, \breve{y}_{ji}, \breve{o}_{ji}) = \left(\left\lfloor \frac{W_w}{\|\mathbb{P}\|_x} \hat{x}_{ji} \right\rfloor, \left\lfloor \frac{H_w}{\|\mathbb{P}\|_y} \hat{y}_{ji} \right\rfloor, \left\lfloor \frac{O_w}{2\pi} \arctan \frac{d\hat{y}_{ji}/dt}{d\hat{x}_{ji}/dt} \right\rfloor \right) \tag{4.1}$$

where $(\hat{x}_{ji}, \hat{y}_{ji})$ and $(d\hat{x}_{ji}/dt, d\hat{y}_{ji}/dt)$ are the tracker's respective estimates of person j's ground-plane position and velocity for observation i, $\hat{x}_{ji} \in [0, \|\mathbb{P}\|_x)$, and $\hat{y}_{ji} \in \left[0, \|\mathbb{P}\|_y\right)$. Position is quantized into $W_w \times H_w$ cells, where the units are meters for rectified data or pixels for trajectories measured in image space. Motion direction is quantized into O_w direction bins, and optionally an addition bin encoding low velocity motion. Because w_{ji} is a tuple of discrete scalars, it can also be represented as a single scalar using any standard multidimensional array indexing technique [98].

[2]The data will always be conditioned on low-level tracking data, so the fact that the collection is modeled as being unordered is not especially important. We will only be using this model for inference, not for data generation. Also, any reasonable tracker can be tuned not produce tracks that randomly skip around the scene.

4.1.2 Activity Cluster Mixture Model

We define an activity cluster as a collection of observations that commonly co-occur within a trajectory. Cluster c is represented as a multinomial over observations, $\boldsymbol{\phi}_c$. There are (countably) infinitely many clusters in the model. The clusters themselves are drawn from a base Dirichlet distribution, H. In the information retrieval literature, the activity clusters are called topics. Teh *et al.* call them clusters.

A Dirichlet distribution is a multinomial's conjugate prior. Whereas a multinomial's parameter space is typically represented by a normalized histogram that gives the probabilities of each emitted symbol, a Dirichlet distribution's parameters can be represented as a normalized histogram with a factor proportional to λ multiplied onto each element. When λ is large, the emitted multinomials are concentrated: they look like the normalized histogram, with high probability. When λ is small, the emitted multinomials fill a larger volume. Given the same dataset, larger values of λ result in more compact models with fewer learned clusters.

In HDP and similar models, the two most common choices for the base distribution H are (a) the uniform distribution or (b) the sample marginal of observations. Empirically, Orbanz and Buhmann [79] found that a uniform distribution tends to avoid overfitting. We also use a uniform distribution.

The infinite collection of mixing weights is given by $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_c, ...)$, where β_c is the weight for cluster c. The elements of $\boldsymbol{\beta}$ are drawn from a "stick breaking" process described by Griffiths, Engen, and McCloskey (see Teh *et al.* [105]), hence the "GEM" shown in Fig. 4-1's table. We now describe the process. β_1 is sampled from a beta distribution such that $\beta_1 \sim \text{beta}(1, \gamma)$. β_2 is sampled from $\beta_2 \sim (1 - \beta_1) \cdot \text{beta}(1, \gamma)$. In general,

$$\beta_c \sim \left(1 - \sum_{c'=1}^{c-1} \beta_{c'}\right) \cdot \text{beta}(1, \gamma)\,, \tag{4.2}$$

where the probability density function for the restricted form of a beta distribution we use is given by

$$p(x;1,\beta) = \frac{(1-x)^{\beta-1}}{\int_0^1 (1-u)^{\beta-1} du}. \tag{4.3}$$

The analogy is that we start out with a stick of unit length. It is broken at a random location and the length of the left segment is β_1. The right segment is broken again, with the length of its left segment being β_2. The process of breaking the remainder of the stick is repeated infinitely.

4.1.3 Generating Trajectories and Observations

In a generative process, we start by sampling the infinite mixing weights, $\boldsymbol{\beta}$, and their corresponding activity cluster multinomial distributions, $\{\boldsymbol{\phi}_c\}_{c\in(1,2,\ldots)}$. We then create J trajectories, indexed by j. From a modeling perspective, our trajectories are equivalent to information retrieval's documents and Teh's groups. As implied by the plate notation in Fig. 4-1, each trajectory is sampled independently: we do not explicitly model any interactions between trajectories. For trajectory j, we first sample its distribution over activity clusters, $\boldsymbol{\pi}_j$. This distribution is sampled from a Dirichlet Process, *i.e.* $\boldsymbol{\pi}_j \sim \text{DP}(\alpha, \boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is the model's marginal distribution over clusters, as we have just discussed. α plays a similar role here to the one played by λ: larger values cause $\boldsymbol{\pi}_j$ samples to concentrate about $\boldsymbol{\beta}$ and thus encourage all trajectories to be sampled from similar distributions over clusters. Smaller values of α allow different trajectories to be more distinctive.

Once a distribution over activity clusters has been chosen for trajectory j, we independently sample I_j observations. To generate a single observation, we sample a cluster index, z_{ji} from the trajectory's infinite multinomial over clusters, $\boldsymbol{\pi}_j$. Given the index, we sample an observation from the corresponding activity cluster, $w_{ji} \sim \boldsymbol{\phi}_{z_{ji}}$.

Given a set of trajectories and their observations from data, Teh *et al.* present three Gibbs sampling algorithms for inferring the cluster assignments, $\{z_{ji}\}_{\forall i,j}$, and the model, $(\boldsymbol{\beta}, \{\boldsymbol{\phi}_c\}_c)$. Given finite data, all of these methods will learn an explicitly-represented finite mixture model plus an implicit infinite model. We use his auxiliary variables method. Refer to [105] for details.

4.2 Results Using Naïve Feature Points

As discussed in §3.2, background subtraction and blob tracking is impractical on the Grand Central scenes discussed in §3 and this chapter. One solution is to detect feature points using the method of Shi and Tomasi [94] (see §3.3), then track those using a sparse optical flow algorithm provided by the OpenCV library [78]. Substantial post-processing is used to smooth the trajectories and reject those with faulty data associations[3].

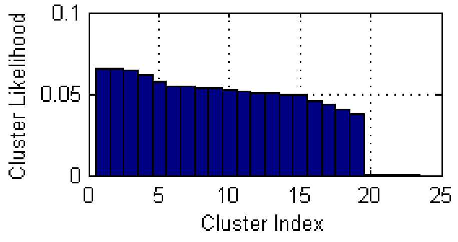
We then quantize the tracked feature point observations with disjoint spatial bins of size 10×10 pixels and $O_w = 4$ orientation bins[4]. In Fig. 4-2, we show the best results we were able to obtain (judged qualitatively). Note that the learned clusters are spatially compact, but they fail to separate out different paths that cross over the same location, but using different moving directions.

To help demonstrate this, we have created Fig. 4-2(j). This plot shows the "effective number of clusters" at each location, or the conditional perplexity of cluster assignments given an observation at a specific location. We define the effective number $\bar{C}_{\breve{x},\breve{y}}$ as the exponentiated information entropy, *i.e.*

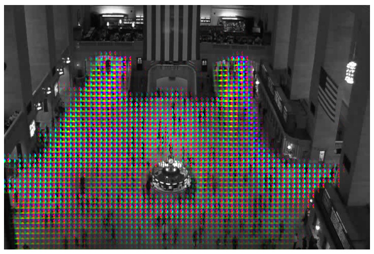
$$\log_2 \bar{C}_{\breve{x},\breve{y}} = -\sum_{z'=1}^{\infty} p(z'|\breve{x},\breve{y}) \log_2 p(z'|\breve{x},\breve{y}) \tag{4.4}$$

[3] *Data association reminder:* Recall that in multi-target tracking, one starts with a set of existing trajectories. Each trajectory has a density over detections. For example, a constant velocity Kalman filter has a Gaussian distribution over the location and velocity of the object at a future time. Given a collection of detections, the data association step matches observed detections with existing (or new) trajectories. The trajectories' densities are then updated.

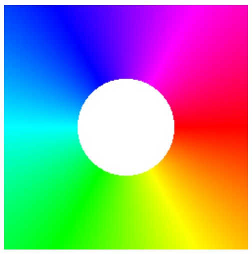
[4] The experiments in this section were performed before the high-resolution Grand Central videos were available. Instead of 1920×1080 input video, we use a very similar 720×480 video. The video used in this section was taken on a day with much milder lighting conditions, so these results are optimistic relative to the improved results we will see in §4.3.



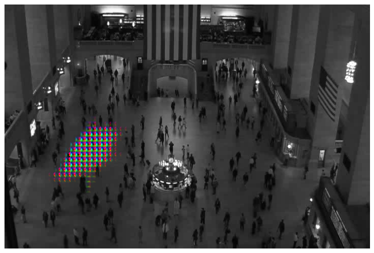
(a) Sorted Cluster Weights ($\boldsymbol{\beta}$))



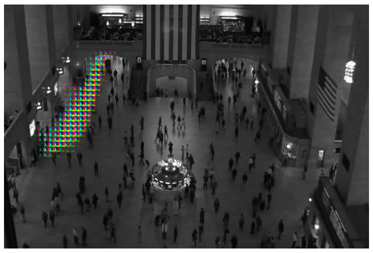
(b) Marginal of Observations



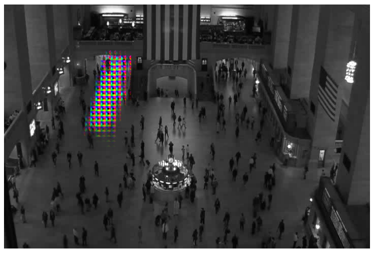
(c) Color Scheme



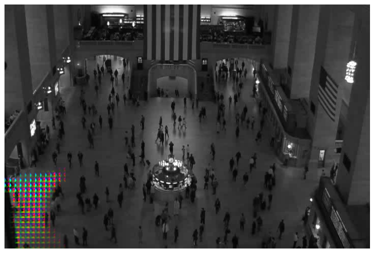
(d) Cluster 1



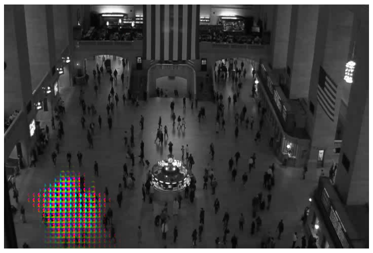
(e) Cluster 2



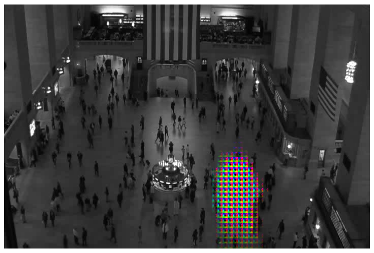
(f) Cluster 3



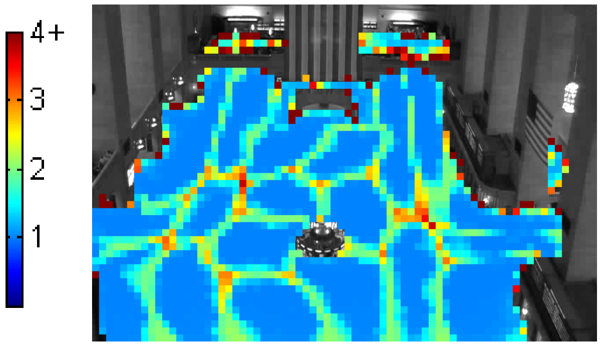
(g) Cluster 4



(h) Cluster 5



(i) Cluster 6



(j) Effective Number of Clusters per Cell

Figure 4-2: *Activity Modeling with Feature Points:* In Fig. 4-2(a), we show the learned activity cluster mixing weights, after sorting them. In Fig. 4-2(b), we show the marginal distribution of all observations (quantized location and moving direction of a tracked feature point) in the dataset. The hue mask in Fig. 4-2(c) encodes direction, and its alpha matting with respect to the background image encodes the observation frequency. Fig. 4-2(d)–4-2(i) show the six most highly weighted clusters. Note that although each cluster does have good spatial compactness, it captures motion in all directions. The model is unable to learn overlapping clusters where people are traveling on different paths. Except for a few of the low-weighted ones (which are not shown here), the remaining 17 clusters are qualitatively similar. In Fig. 4-2(j), we see that there is only a single cluster representing motion at most locations on the concourse floor.

where

$$p(z'|\breve{x}, \breve{y}) = \sum_{\breve{o}=0}^{O_w-1} \phi_{c,\breve{x},\breve{y},\breve{o}} \tag{4.5}$$

and $\boldsymbol{\phi}_c = (\phi_{c,0,0,0}, \ldots, \phi_{c,\breve{x},\breve{y},\breve{o}}, \ldots, \phi_{c,W_w-1,H_w-1,O_w-1})$. In Fig. 4-2(j), notice that only along cluster boundaries is the effective number greater than 1. We call this effect a "directional degeneracy" in the model.

We found that when they used smaller spatial bins or more orientation bins, the learned clusters became less spatially compact. Tuning other parameters did not yield improvements.

4.3 Results Using HOG Detections

Our primary motivation for developing an efficient pedestrian detector (see §3) was improving the input data for activity modeling. We now demonstrate how we are able to obtain more semantically meaningful clusters by using the higher-quality trajectories produced by our Dalal and Triggs detector, compared to the feature point tracking. In doing so, we will highlight difficulties we encountered as well as some initial strategies for mitigating them.

For our input data, we use the 1 hour high definition Grand Central clip we discussed throughout §3. Our GPU-accelerated detector is used to detect pedestrians. We use constant velocity Kalman filters and greedy data association to find trajectories, as described in §1.4. We then apply a number of post-processing and filtering steps to reject problematic observations and tracks. We first apply Gaussian smoothing to the trajectory to reduce the effects of spatial quantization in detector window placement. We then reject any trajectories with too few observations, ones that do not travel very far, and ones that are consistently close to the image edge. Observations with unrealistic speeds are rejected. If a trajectory has too large of a time gap between observations, only the longest sequence of observations with an allowable gap is retained. The first and last few observations are the most likely ones to have bad data associations, so they are removed too. These filtering steps are similar to the ones used for the feature point tracks and most are common heuristics used in other tracking applications.

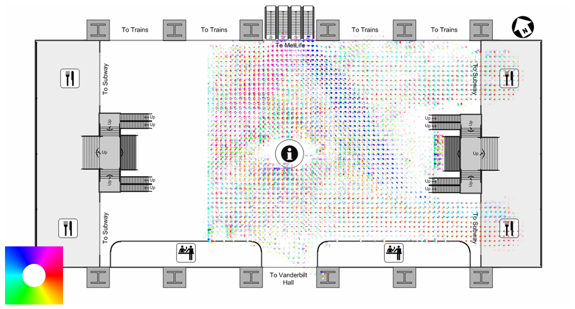
The surviving trajectories are then projected onto the ground plane for three reasons.

- The underlying pedestrian activities take place in the real world, so it makes sense to do our modeling in that space.
- This choice also allows future work to seamlessly combine data from multiple registered videos.
- We are able to apply more meaningful trajectory filtering. Specifically, we can specify tighter bounds on feasible pedestrian speeds because we do not have to allow for the effects of foreshortening on image speed.

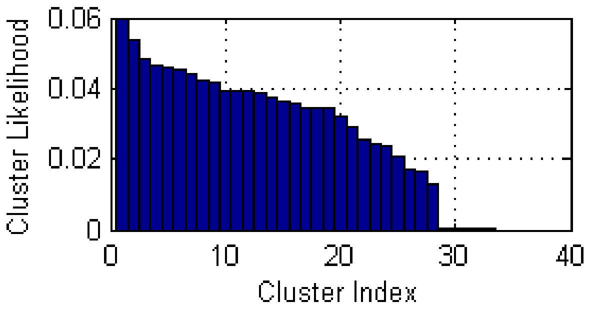
The re-filtered trajectories are then quantized into $12\,cm \times 12\,cm$ spatial bins and 16 orientation bins. For our trajectories, we found that the additional orientation bins help reduce the amount of directional degeneracy in the model. Using the quantized trajectories, we then the HDP model described in §4.1 to learn activity clusters. In Fig. 4-3, we show a summary of the input data, the learned cluster weight distribution (when λ is chosen to produce a similar number of clusters as seen in the feature point results), and the effective number of clusters per spatial quantization cell. Comparing Fig. 4-3(c) to Fig. 4-2(j), we see that with our quantization changes and higher-quality trajectories, the HDP model is able to learn clusters that overlap spatially but encode different moving directions.

In Fig. 4-4, we show the qualitatively best activity clusters. These capture medium- to long-range common paths with a consistent moving direction.

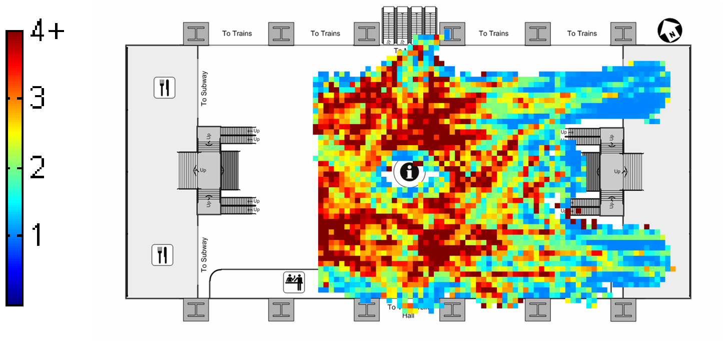
Unfortunately, not all activity modeling results are as ideal. In Fig. 4-5, we see that sometimes our user-chosen parameters cause multiple semantic path segments to be represented by a single cluster. By adjusting our priors, we can break up the cluster into its more meaningful constituents, as seen in Fig. 4-6.



(a) Marginal of Observations

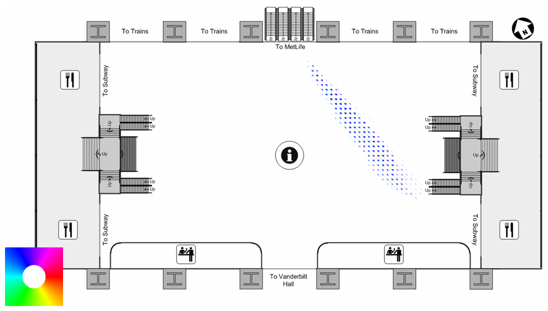


(b) Sorted Cluster Weights ($\boldsymbol{\beta}$ when $\lambda = 0.01$)

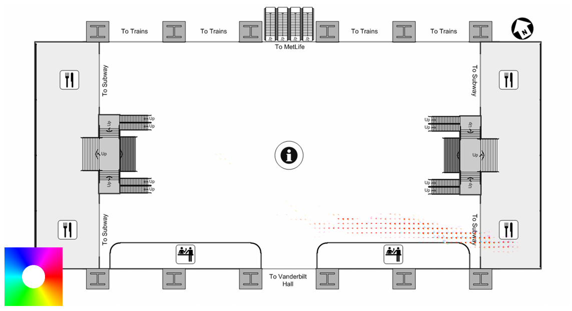


(c) Effective Number of Clusters per Cell

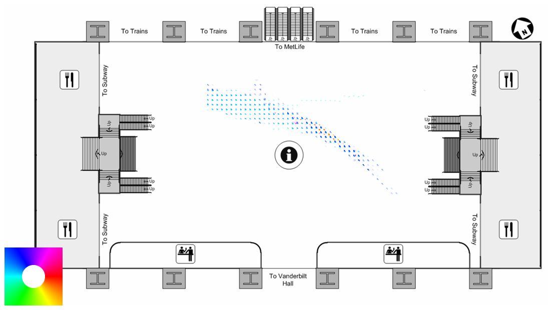
Figure 4-3: *Activity Modeling with Pedestrian Detection:* In Fig.4-3(a), we show the marginal distribution of all observations for the activity modeling experiments using pedestrian detection, rectified to the ground plane. The video used for this experiment is similar to the one used in Fig. 4-2, but this one is calibrated with respect to the ground plane, it uses 1920×1080 video as opposed to 720×480, and it was taken on a day with harsher lighting. We have empirically chosen $\lambda = 0.010$ so that a similar number of clusters is learned in our model compared to the one depicted in Fig. 4-2. In Fig. 4-3(c) we see that we are not just clustering on spatial location (as was the case in Fig. 4-2(j)), but we also are able to discriminate by motion direction: from inspection we know that in the ring around the central information booth, people walk in many different directions. In those parts of the scene, we have learned a large number of effective clusters. In the next several figures, we will examine various individual clusters.



(a) Cluster 17

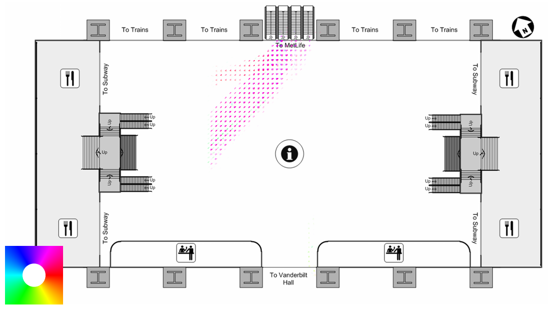


(b) Cluster 20

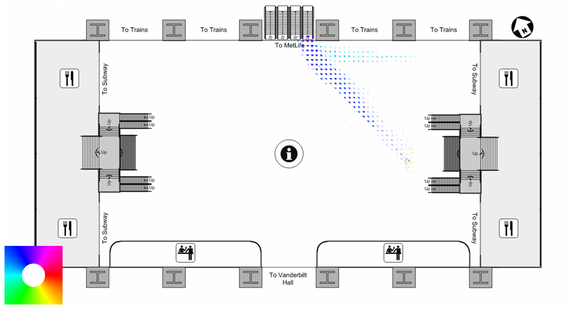


(c) Cluster 26

Figure 4-4: *High Quality Paths:* A number of the learned clusters are near our ideal: they represent long and narrow common paths through the scene, with little confusion in the direction of motion. When using feature point tracking (see Fig. 4-2), few clusters are as long and narrow as these, and all of those caption motion in both directions for a given location.

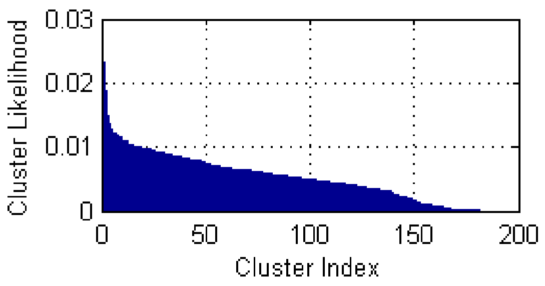


(a) Cluster 4

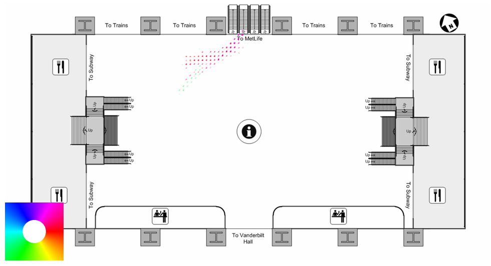


(b) Cluster 7

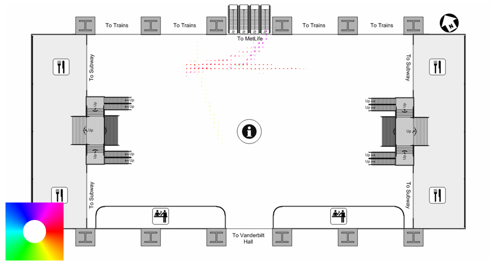
Figure 4-5: *Merged Paths:* Several other learned activity clusters group together observations from trajectories that converge on the same destination. In Fig. 4-6, we will see that the cluster depicted in Fig. 4-5(a) can be broken up into more semantically meaningful paths by lowering the pseudo-count prior λ.



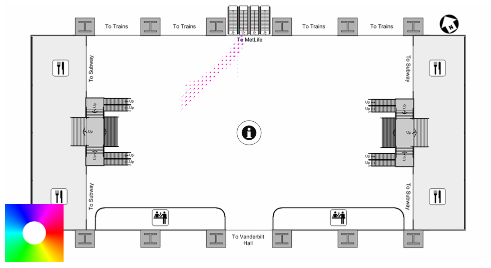
(a) Sorted Cluster Weights ($\boldsymbol{\beta}$ when $\lambda = 0.001$)



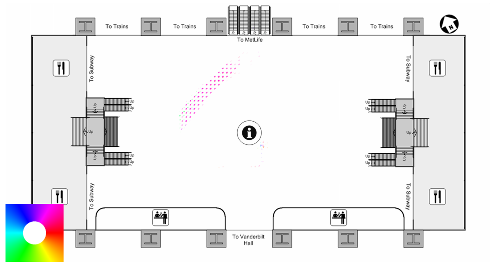
(b) Cluster 71'



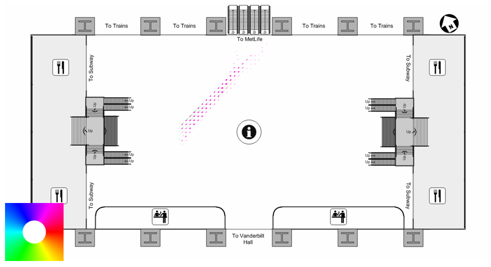
(c) Cluster 7'



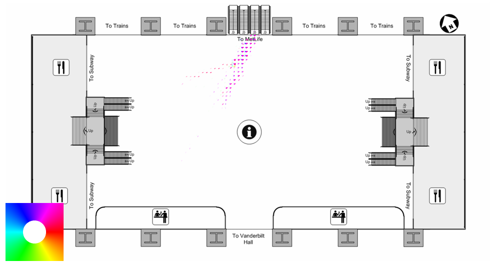
(d) Cluster 5'



(e) Cluster 6'

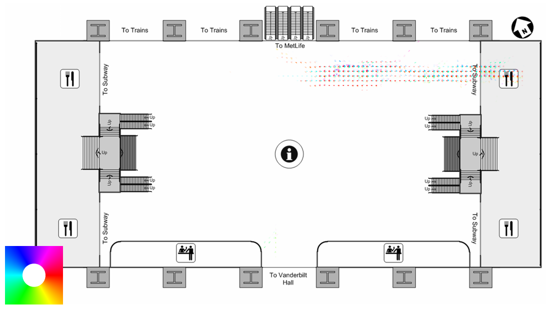


(f) Cluster 15'

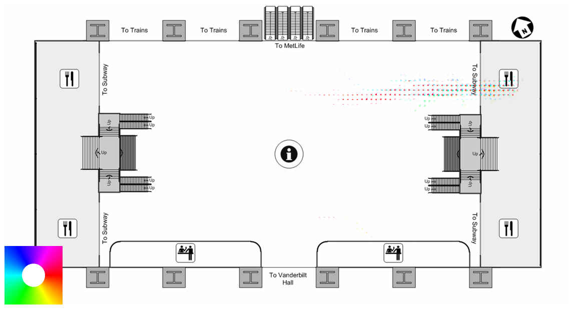


(g) Cluster 66'

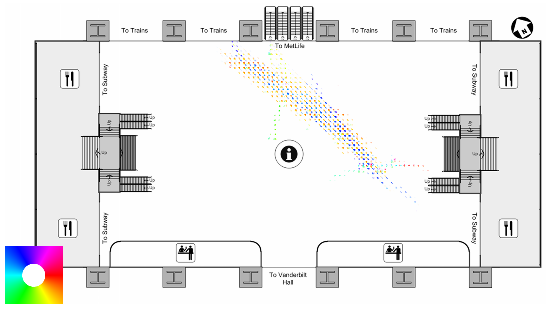
Figure 4-6: *Weaker Prior Benefits:* When we use a strong uniform prior on the distribution of observations within each activity cluster, it is easy to encounter problems where we undersegment the clusters, from a semantic perspective. In Fig. 4-5, we saw that converging paths may be grouped together when λ is large. Even worse, in Fig. 4-8 we see that semantically meaningful sub-paths are sometimes randomly clustered together by the Gibbs sampler. A common remedy is to weaken the prior. Here we show a collection of clusters that are learned when we reduce λ from 0.01 to 0.001. We have selected the clusters that correspond most closely with the original cluster 4 shown in Fig. 4-5(a). For reference purposes, we also show the sorted cluster weight distribution in Fig. 4-6(a). Fig. 4-6(b) and Fig. 4-6(c) correspond to people traveling from different parts of the northwest subway archway to the MetLife escalators. Other subfigures show that the unsupervised HDP algorithm automatically learned clusters corresponding to each of the possible concourse entrances on the west side.



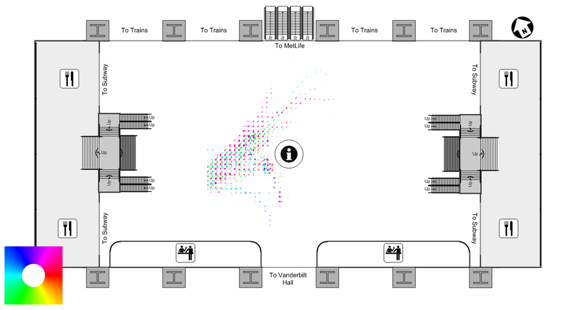
(a) Cluster 8



(b) Cluster 15

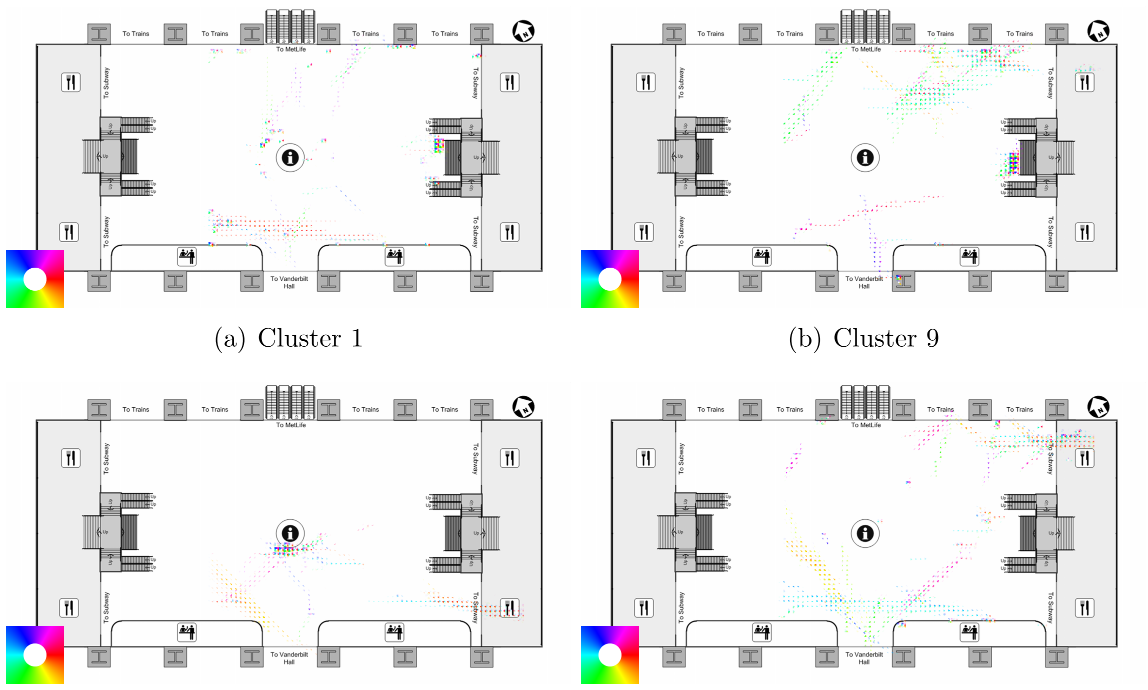


(c) Cluster 22



(d) Cluster 18

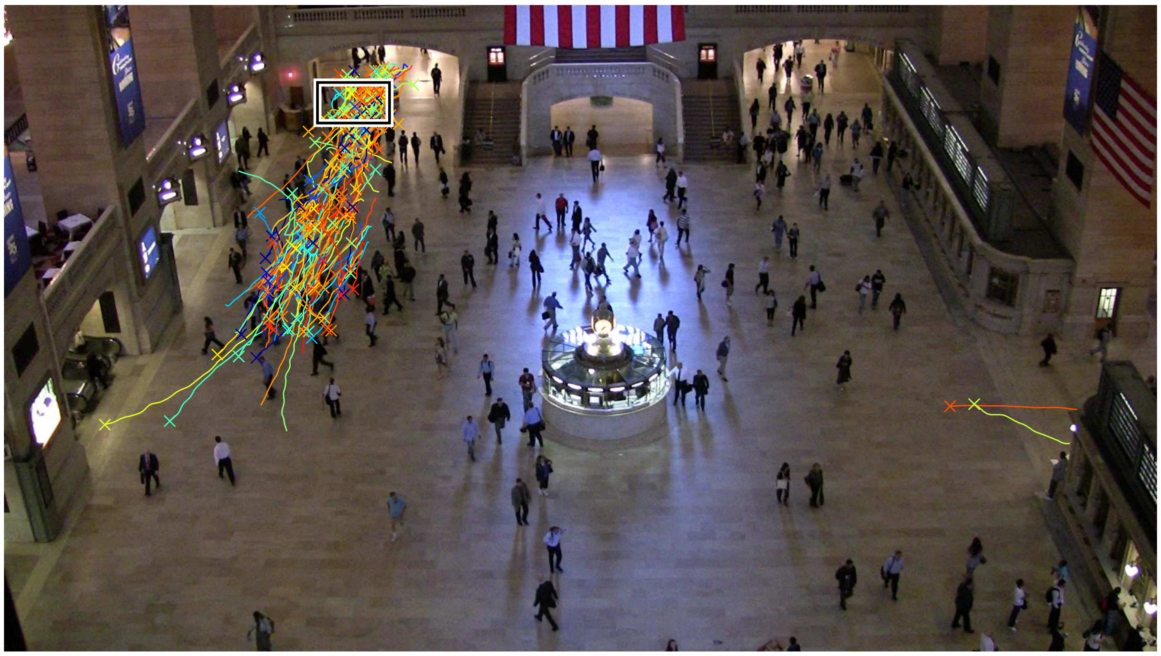
Figure 4-7: *Directional Degeneracies:* The most difficult problem we encountered was the clustering together of trajectories going in different directions at the same location. To see this problem manifested, use the color key in the corner of each subplot and notice how the rendered wedges tend to be paired representing movement in opposing directions, but at the same location. If we refer back to feature point results in Fig. 4-2, notice that the problem of not being able to separate out distinct directions into separate clusters is much worse for them: every single cluster with non-trivial weight experienced this problem. In Fig. 4-9 and Fig. 4-10, we will show the two most common reasons for having clusters that are degenerate in the motion direction dimension.



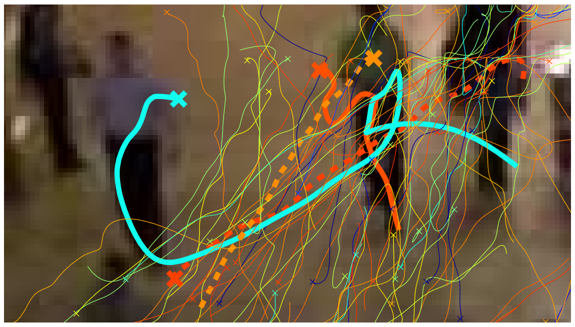
(a) Cluster 1 (b) Cluster 9

(c) Cluster 10 (d) Cluster 11

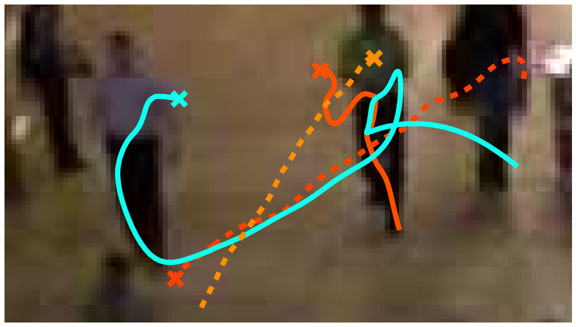
Figure 4-8: *Clustering of Disjoint Regions:* We use a uniform prior for the base distribution of our HDP model. When the prior is strong (*i.e.* when the pseudo-count prior λ introduced in Fig. 4-1 is large), all clusters are more similar to each other. As a result, there is a higher likelihood of the Gibbs sampler randomly assigning observations from a trajectory to a cluster that does not represent it well. Those spurious assignments then cause the model to encourage other similar associations. Given insufficiently many Gibbs sampling iterations, it can be difficult to escape these local minima. In this figure we show several clusters that include observations spread across the scene that are a result of insufficient learning, even after 15,000 Gibbs sampling iterations. As with the case of merged paths (see Fig. 4-5), one solution is to allow the clusters to become more specialized by lowering the pseudo-count prior, λ. See Fig. 4-6 for an example.



(a) Most Likely Trajectories under Cluster 8 (see Fig. 4-7(a))

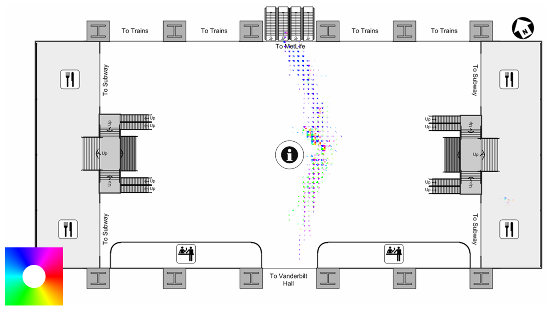


(b) Closeup of Some Meandering Trajectories

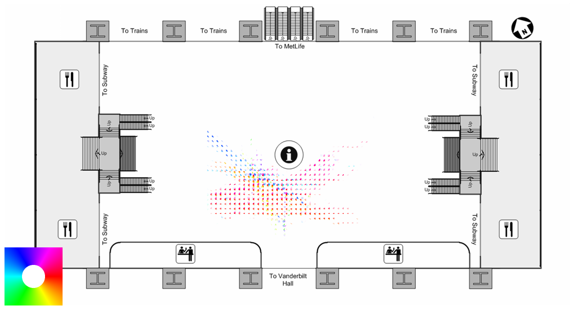


(c) Just the Discussed Trajectories

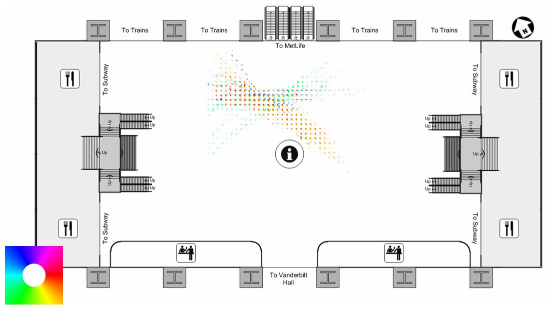
Figure 4-9: *Directional Degeneracies: Tracking Errors:* In Fig. 4-7, we presented the problem of learning clusters that are degenerate in moving direction: some clusters tend to include observations in the same location but opposing or different directions. In this figure, we explore a common reason: tracking errors. In Fig. 4-9(a), we show the set of trajectories for which cluster 8 (see Fig. 4-7(a)) is the single most likely cluster to have generated its observations. The first observation of each trajectory is shown with an $\times$. Notice that there are trajectories going from the top of the image down and from the lower parts of the image going up. For clarity, we have zoomed in on the black and white box, creating Fig. 4-9(b) and Fig. 4-9(c). In those, we have highlighted four trajectories of interest. The dashed bold ones are typical "good" trajectories, but they have roughly opposite directions of motion. They have been grouped together in the same cluster because there are enough tracks like the pair of solid bold trajectories. Because of bad data associations, these tracks meander through the scene. Because the two solid bold lines do briefly share a common location and direction, they are able to be clustered together. The cyan trajectory then shares a long portion of its path with the orange dashed trajectory. The solid orange trajectory shares a portion of its path with the dashed yellow trajectory. Because of the transitive nature of HDP clustering, all four trajectories are able to be grouped together. Our primary mechanism for combatting this problem is to have a conservative tracker that encourages long and straight trajectories traveling at feasible human speeds. More sophisticated data association would likely help.



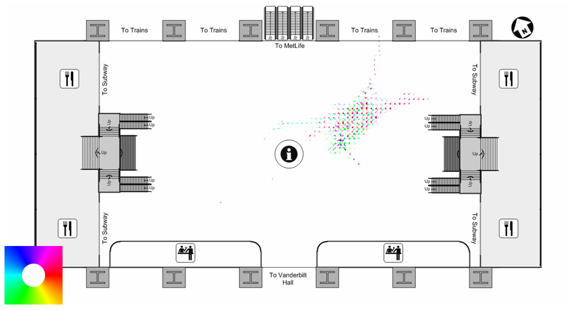
(a) Cluster 25



(b) Cluster 13



(c) Cluster 14



(d) Cluster 28

Figure 4-10: *Directional Degeneracies: Mingling:* In Fig. 4-9, we explained how tracking errors are one of the most common reasons for directional degeneracy within learned activity clusters. The second most common cause is the existence of "mingling regions" in the image: places where people tend to stop or turn around. Based on our own observation and conversations with security officials, people tend to use the information booth in the center of the scene as a meeting and waiting place. They will also tend to remain in pockets of low traffic when loitering. These pockets tend to form within a large circular region about the information booth. As people loiter, they tend to change moving directions. This creates the opportunity for spurious clustering for the same reasons that bad tracking can, as we discussed in Fig. 4-9. In Fig. 4-10(a), we show a mild case: cluster 25 is mostly composed of people walking north from Vanderbilt Hall to the MetLife escalators (bluish and purplish wedges). As it passes by the east side of the information booth, the HDP model tends to co-cluster with minglers there (all hues). Since some of those minglers head back to Vanderbilt Hall or to the southwest, a collection of extra observations are also associated with the cluster (greenish wedges below the information booth). The remaining subfigures show even more mixing of moving directions due to loitering, dodging, and crowd weaving that tend to happen in their respective regions. New ways of biasing the training procedure and/or a more sophisticated model are needed to overcome this issue.

As in the feature point results, we do still experience directional degeneracy issues, but less frequently and less severely. In Fig. 4-7, we show some examples of this problem. Some of the degeneracies are caused by tracking errors (Fig. 4-8 and Fig. 4-9), while others are due to legitimate meandering of trajectories (Fig. 4-10).

4.4 Summary and Lessons Learned

In this chapter, we showed how we are able to obtain more semantically meaningful clusters using tracks based on strong model pedestrian detections, compared to tracks based on feature point detections. In performing our HDP activity modeling experiments, we learned several several common-sense lessons that may be helpful to others using similar models.

- Lowering λ avoids random grouping of common trajectory segments, allowing for greater spatial compactness. In a scene such as the Grand Central one discussed here, there are many underlying ground truth paths taken, so it makes sense to try to learn them as best as possible.

- Lowering the spatial resolution reduces the random grouping and thus allows for greater spatial compactness, but at the expense of more direction grouping. The spatial compactness improvement is because we have a smaller vocabulary (a smaller space of quantized positions and moving directions) so it is easier to avoid overfitting and local minima problems with the Gibbs sampling. The direction grouping is worse because with larger cells, we have a higher likelihood of seeing a people spanning multiple direction bins within the cell.

- Lowering the angular resolution tends to encourage false direction grouping for the same reason that lower the spatial resolution does.

Chapter 5

Silhouette Refinement for Gait Recognition

This chapter contains joint work with Lily Lee and Kinh Tieu [62].

5.1 Introduction

In §4, we showed promising results in activity modeling for large scenes. A commonly desired feature of a site monitoring application is to detect anomalous activities and then identify the individual participants. For example, if a person is loitering near a sensitive area, a security official may wish to know if that person has loitered there in the past. If so, they may be casing the area for nefarious purposes. To answer this and similar questions, we need a way of identifying and/or matching people given tracking data. An advantage of building an activity model based on tracked pedestrian detections (as opposed to weaker detections like corner points) is that when an anomaly is detected, we can more easily analyze the image data corresponding to the trajectory in question: we are given the bounding boxes containing the person in question.

In this chapter, we explore a collection of methods for identifying individual pedestrians based on shape features. For privacy reasons, we were unable to perform these experiments on the Grand Central data discussed in Chapters 3 and 4: for legal and ethical reasons, consent of each test subject is required for research involving the long-term tracking of people. It is infeasible at this time to either obtain consent from the estimated 750,000 people who pass through Grand Central each day [39], to obtain ground truth data for a significant fraction of the population, or to do the recognition experiments in a way that is guaranteed to preserve privacy.

Instead of doing recognition experiments in the Grand Central setting, we will use a standardized dataset from the NIST Gait Challenge project[1]. Because its video clips were taken under constrained conditions, we return to a background subtraction-based tracking pipeline and we concentrate on the effects of foreground silhouette quality on recognition results. Compared to contemporaneously-developed algorithms, our system achieves competitive results. More recent results are summarized by Sarkar and Zongyi [89].

To further motivate the extraction of pedestrian silhouettes (as opposed to just detecting bounding boxes), we note that the ability to accurately segment pedestrians from a video stream is important for applications such as gait recognition, person height/girth estimation [51], articulated body tracking, pedestrian activity description [44], and 3D reconstruction of people from silhouettes [69]. We use a model-based approach to pedestrian segmentation that incorporates information from background subtraction, pedestrian shape models, and an individual shape model sampled at discrete phases of the walking cycle. Our approach reduces noise introduced by background subtraction, and fills in missing parts of the pedestrian silhouette, which often result from camera noise or lack of color/intensity difference between the pedestrian and the background. In addition, our pedestrian models are learned from a noisy background subtraction process, hence making the entire process completely automatic.

Traditional approaches to pedestrian segmentation from video generally involve using a background subtraction algorithm to arrive at foreground silhouettes, then post-processing to refine the silhouettes. Because background subtraction inherently detects pixel value changes in the video/image, spurious foreground pixels are formed

[1]See http://www.gaitchallenge.org.

by noise in the video, and pedestrian silhouettes will have holes and missing parts if there is not enough contrast between the pedestrian and the background scene. The general solutions to these difficulties are to apply large numbers of morphological operations to fill in holes and remove noise in the silhouettes, or to apply a smoothing process at the background subtraction stage [81]. In either case, these operations tend to systematically distort the silhouettes and remove fine details which may be important for identification.

We investigate the particular case of pedestrians walking in a plane roughly parallel to the camera image plane. Under this scenario, it is easy to see that there are commonalities between all pedestrian shapes. Moreover, the cyclic nature of the walking action ensures that the silhouette appearance of each individual pedestrian is repeated at each stride at semi-regular intervals. These observations make it possible to improve the estimation of silhouette appearance over time and over a population of pedestrians. We take advantage of these characteristics to learn two types of pedestrian models, one that represents all pedestrians, and one that represents each individual walking video sequence. Using the pedestrian population model and individual sequence models, we are able to remove noise from each frame of a silhouette sequence and fill in missing parts of each silhouette. To show that these silhouettes are an improvement over the traditional methods of silhouette smoothing, we apply our approach to the NIST gait data to produce a set of silhouettes and use these silhouettes in a set of baseline gait recognition tests introduced in [81]. The dataset contains video clips of 71 different subjects, with up to 8 distinct clips per subject. Our results show that recognition results are improved using our model-based silhouettes.

While we are only concerned with extracting pedestrian silhouettes in this thesis, the method we propose is generally applicable to any moving object that demonstrates the cyclic nature and common overall appearance that are observed in pedestrians. For example, models for joggers, or trotting dogs or horses may be built using the same technique.

5.2 Previous Work

There are many works related to the problem of pedestrian detection, tracking, and segmentation. We indicate several of relevance to our approach that were developed in the 1994–2003 time range, when work on the NIST dataset was most active. As previously mentioned, Sarkar and Zongyi [89] provide an overview containing a few more recent results. We categorize these works into two types: pedestrian detection and pedestrian shape representation.

Oren *et al.* [80] trained a set of wavelet template representations of the frontal view of pedestrians. These representations capture the shape gradient difference between the pedestrian and the surrounding background. The authors applied their pedestrian representation to images to detect roughly frontal (or back) views of pedestrians. Gavrila [36] used a set of edge models of pedestrian shapes to detect pedestrians from video sequences taken with a moving camera. While pedestrian detection is the goal of both algorithms, additional steps are needed to extract the silhouette of pedestrians.

Haritaoglu *et al.* [44] used background subtraction to detect, segment, and track pedestrians, but they did not eliminate the errors introduced by background subtraction. Baumberg and Hogg [6] represented the pedestrian shape by a chain of edge points. However, a clean segmentation of the pedestrian is assumed, and point selection requires human intervention.

Kale *et al.* [54] use a five state HMM for gait identification and reduce their observation distributions to a single Gaussian per state using noisy silhouettes. Zhou and Chellappa [128] use a time series continuous state space model to recognize people walking toward the camera.

The best results during the initiative used an HMM by Sunderesan, Chowdhury, and Chellapa [103]. They first recover the gait period for a sequence of silhouettes recovered from a person, then they cluster the silhouettes to find good exemplars at each of 6 discrete phases in the walking cycle.

While there are pedestrian model representations presented in these papers, they

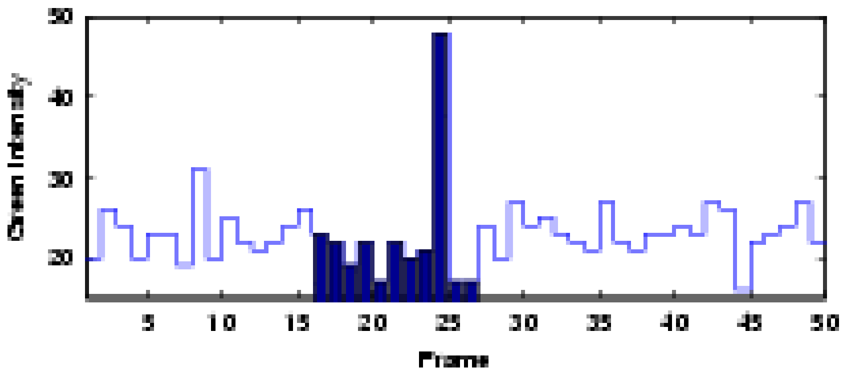


Figure 5-1: *Intensity of a Pixel through Time:* Within the foreground segment, indicated by the dark bars, there are times when the intensity is indistinguishable from the background values. Similar situations occur even when examining all color channels simultaneously, as we will explore in more detail in §6.

do not address problems inherent to background subtraction that make accurate extraction of pedestrian silhouettes difficult.

Without the full silhouette, questions such as "what color of clothing is the pedestrian wearing" can be hard to answer. Our approach seeks to overcome these difficulties by learning a probability distribution of pedestrian foreground models at different phases of a walking cycle over time and then using these models to provide better shape definitions and to recover from errors in the background subtraction process.

5.3 The Need for Model-based Segmentation

If pedestrians always appeared in colors that are drastically different from the surrounding background, and there were no cast shadows, then pedestrian segmentation from any image would be a simple task. However, in any realistic video monitoring situation people may have colors on the body that are close to the background, and shadows will appear. For example, Fig. 5-1 shows the intensity of one color channel of one pixel location in a video sequence. We manually found the frames for which the pedestrian is the foreground at that location. Clearly, there are some frames for which the foreground process is indistinguishable from the background process.

The pedestrian in this case is wearing a black shirt and walking past a black background. As a result there are large holes in the torso of the silhouette. These are difficulties that no local background subtraction algorithm can solve, because background subtraction only detects changes in pixel intensities. A non-local approach such as a Markov Random Field classifier (see §1.3) can bridge the gap between the pedestrian's head and legs, but only at great risk of also including non-pedestrian pixels in the silhouette. A model-based pedestrian representation imparts expectations on the structures of pedestrians, and the confidence level associated with the expectations will allow us to ignore the noise in the video data and fill in the expected structure where data is missing.

5.4 Learning Pedestrian Models

We consider the case where the pedestrian is walking in a plane that is roughly parallel to the image plane and always in the same direction. Under this scenario, the cyclic nature of pedestrian silhouette appearance is readily apparent. The same phase of a walking cycle will appear repeatedly in a sequence. Hence we can obtain a better estimate of a silhouette by using all silhouettes that correspond to that same phase. To further simplify the problem, we assume that the walking direction is known. Hence we only need to represent the silhouettes in one direction while the silhouette appearance from the opposite direction is a mirror image of the standard direction.

The above observations lead to a straightforward method for obtaining a pedestrian model within a silhouette sequence using a number of discrete phase representations:

1. Detect the period of the silhouette sequence using periodic features, such as the silhouette aspect ratio.

2. Align all silhouettes by the phase of the walking cycle assuming a constant walking period.

3. Average all silhouettes assigned to the same phase.

Assuming that there is no systematic error in the backgrounding process or in the environment, and that pedestrians walk at roughly constant speed, this method will generate a good representation of silhouettes over different phases of a walking cycle that captures the shape of the pedestrian in the walking sequence. However, both of these assumptions are occasionally violated. If the pedestrian has consistent patches of clothing that match the background environment, the raw silhouette sequence will have many frames with large holes in the body. If the walking speed of the pedestrian changes in a sequence, assigning a silhouette to its correct phase may be difficult.

5.4.1 Pedestrian Population Model

To address the issue of systematic noise in a gait video sequence, we devise a separate model that represents the appearance of all pedestrians, which we name the pedestrian population model. We assume that while systematic errors in background subtraction may occur for one walking sequence, they are unlikely to occur at the same location for a population of pedestrians. Hence, a silhouette model constructed using a sampling of silhouette sequences from a general population of pedestrians will not suffer from systematic background errors. However, because different individuals have different stride lengths, aligning and averaging silhouettes from different pedestrians by phase results in blurred legs, especially for the phase with the widest stance. This reduces the benefits of conditioning the model on the walking phase of a silhouette. As a consequence, we choose to represent the silhouette of all pedestrians with the mean silhouette of a training set that is representative of the population, ignoring phase information.

There are some postural differences between the silhouette appearances of male and female pedestrians (see Fig. 5-2a and 5-2b), thus the training sequences need to contain an equal number of male and females. Fig. 5-2c shows the average pedestrian model computed from 5 males and 5 females randomly chosen from our gait data set. This population model is generated using 100 random silhouette frames from each of the 10 training subjects. The amount of data used represents 1% of all the data frames, and 8% of the total number of frames of the training subjects.

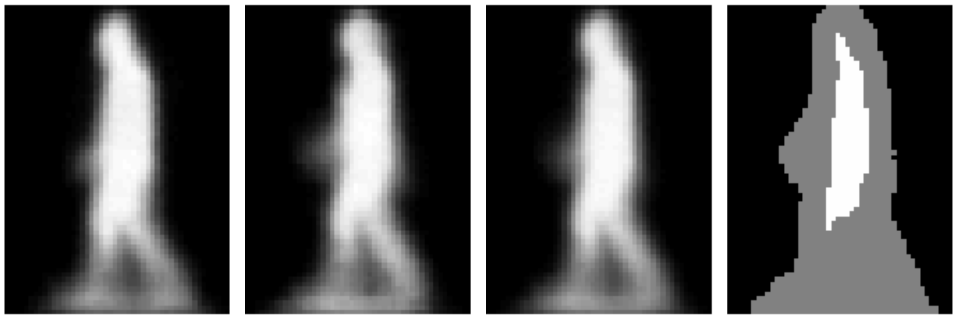


Figure 5-2: *Pedestrian Population Models:* (a) male, (b) female, (c) average model, and, (d) mask for pedestrian shape–black for turning off a pixel, white for turning on a pixel, and gray for unchanged pixel).

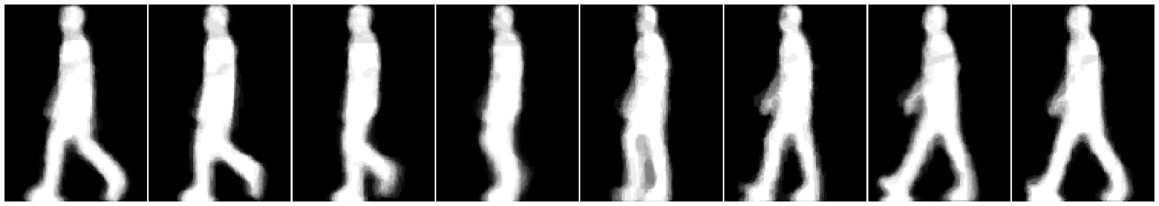


Figure 5-3: *Example Emission Model:* Sample model of 8 phases of the walking cycle for one of our sequences after HMM training.

5.4.2 Pedestrian Sequence Model

To overcome the constraint on constant walking period, we construct a hidden Markov model (HMM) of the silhouette appearances where each state represents the silhouette at different stages of walk for each pedestrian silhouette sequence. The transitions between the states in an HMM contain information about the relative amount of time a pedestrian stays at each state and thus covertly constrains the period, but this is not a hard constraint and does allow for adaptation to changing walking speed in a walking sequence. In addition, because an HMM is trained on each sequence, the states of the HMM will represent the silhouette appearance of each sequence much better than a pedestrian model constructed using any generic silhouette sequence.

5.4.3 HMM Training

An HMM is a probabilistic model of a random process with discrete states, s_0, s_1, ..., s_t, ... In a first order Markov model, the state of the system at time $t+1$ can be predicted knowing only the state at time t, *i.e.* $p(s_{t+1}|s_t, s_{t-1}, ..., s_0) = p(s_{t+1}|s_t)$. In our case, the states are 8 phases of the walking cycle, represented as images in Fig. 5-3. A Markov model is hidden when we are unable to directly observe the states. Instead we observe some output of the system, characterized by a probability distribution, $p(\boldsymbol{y}_t|s_t)$. For the pedestrians, we see images (our observations) of a person instead of having a perfect noise-free "phase detector" (our states).

An HMM is characterized by the probability of starting in some state, $p(s_0 = i)$, the transition probabilities, $p(s_{t+1} = i|s_t = j)$, and the observation probabilities, $p(\boldsymbol{y}|s)$. These probabilities are estimated using standard techniques [85], with the following caveats.

First, we model walking as a set of cyclical transitions between N discrete states, where we have selected $N = 8$.

Second, we assume that a person will start being filmed at a random time with respect to the phase, so $p(s_0 = i) = \frac{1}{N}$ for all i. Third, we set the transition probabilities to be:

$$p(s_{t+1} = i|s_t = j) = \begin{cases} 1 - \frac{1}{f/N} & \text{if } i = j, \\ \frac{1}{f/N} & \text{if } i = j + 1 \bmod N, \\ 0 & \text{otherwise,} \end{cases} \tag{5.1}$$

where f is the average number of frames in a walking cycle for the given sequence ($f > N$). $\frac{f}{N}$ is the average number of frames per phase transition, so $\frac{1}{f/N}$ is the probability of transitioning out of a state after one frame. In informal experiments, we found that allowing the non-zero entries to be learned from data had a negligible effect.

Finally, our observations are binary silhouette images. We model the probability of each individual pixel being turned on as an independent Bernoulli random variable. This model can be represented as an image where the intensity of a model pixel is the probability that that pixel will be on in an observed binary silhouette image. As previously mentioned, Fig. 5-3 is a rendering of this model for a particular HMM we trained.

To train the HMM, we must supply initial estimates of these probabilities. In our case, the transition and initial state probabilities are fixed as described. For the

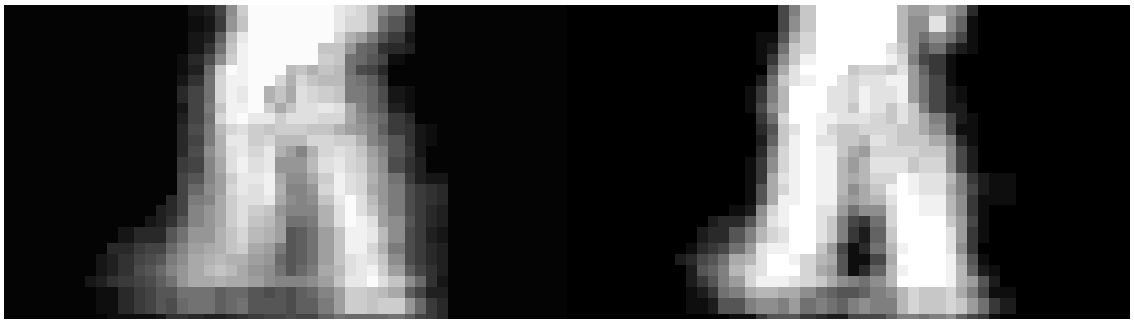


Figure 5-4: *Closeup of the Legs for the Sixth State of a Sequence:* Left: original estimate based on averaged frames. Right: refined estimate after HMM training.

observation probabilities, we start by assuming a near-constant walking speed and assigning the widest stance to be state 0. We then estimate the state of the frames:

$$s_t = \left(s_0 + \left\lfloor t \frac{1}{f/N} \right\rfloor \right) \bmod N \tag{5.2}$$

where s_0 is the state index for frame 0 and t is a non-negative integer. For our initial observation probabilities estimates, we then average all of the frames assigned to each state. For the frames in the NIST gait data we are using, the assumption of near-constant walking speed is valid, and these initial estimates work well. A more robust method would be necessary if there were significant changes or drift in the walking speed.

Once we have the initial estimate of the observation probabilities, we train an HMM on the sequence silhouettes to refine the probabilities. The HMM is able to adapt to smaller fluctuations in walking speed and make the observation model sharper, as seen in Fig. 5-4.

5.5 Raw Silhouette Extraction

We have assumed to this point that the raw pedestrian silhouettes used as input for our model-based pedestrian silhouette extraction method had been obtained and that the tracking of the silhouette is accurate. Below we describe the process by which we obtain such a set of silhouettes.

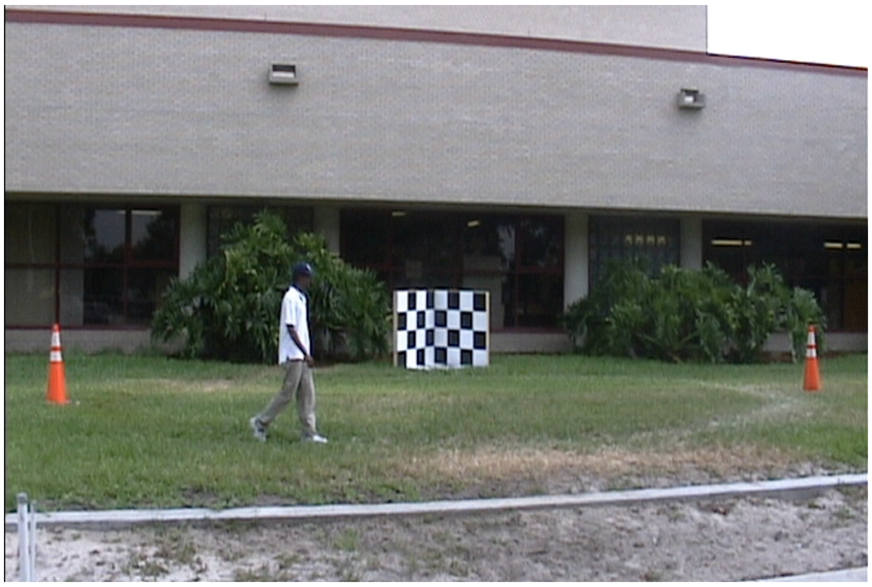


Figure 5-5: *Typical Frame for the NIST Gait Dataset:* This is a typical video frame from the NIST gait dataset. Test subjects were asked to walk an elliptical path between two cones. Different clips were recorded of the same person, varying parameters such as the camera angle, walking surface (concrete versus grass), shoes worn, with and without a carried briefcase, etc.

5.5.1 The Gait Data

The data set we are using is the standard NIST gait data set; the details of the data collection method are described in [81]. Subjects were asked to walk along a smoothly curving path under differing environmental and imaging conditions. In Fig. 5-5, we show a typical video frame from the dataset.

The difficulties posed by automatically extracting good silhouettes from this data set include: shadows on the ground, grass covering feet, moving objects (including people, palm trees, fluttering construction tape, etc.) in the background, subjects wearing clothing that is largely indistinguishable from the background. All of these make the tracking and background subtraction problem difficult. However, the predefined pedestrian path allows us to apply global constraints to simplify the tracking problem. Because all frames of a gait video sequence are available at processing time, we are able to use a batch background subtraction algorithm to extract the foreground.

5.5.2 Tracking and Background Subtraction

Because of the moving objects in the scene and the amount of harsh shadows, tracking the pedestrian accurately becomes a challenging problem if we make no assumption about the gait data. To simplify the tracking problem, we begin with frame differencing (i.e. we subtract color values at each pixel between successive frames) to initially locate the pedestrian in the image. Frame differencing has the advantage that it is robust to gradual lighting change, large shadows, and even waving trees, thus allowing us to localized the pedestrian accurately. However, it does suffer from missing pixels from the upper portion of the body at times, because the torso generates less motion than the legs. Hence we have to choose a large bounding box to outline the pedestrian. A pedestrian detector such as the one discussed in §3 could be used instead.

After using the frame difference image to localize the pedestrian, we impose a constraint that the path of the silhouette centroid must be smooth to a 2^{nd} degree polynomial. We use an iterative robust estimation process to generate a path and a set of bounding boxes containing the pedestrian.

Given the bounding boxes for the tracked pedestrian in each frame, we extract the initial silhouettes by performing background subtraction only within the moving bounding box (see §1.3 for a general description of background subtraction). In these videos, we learn the per-pixel background model in batch, fitting a single Gaussian model to the observed color data, but only using observations when no bounding box overlaps the pixel in question.

5.6 Model-based Silhouette Refinement

Given the raw pedestrian silhouettes generated in the process described in §5.5, and the pedestrian models described in §5.4, we can post-process the raw silhouettes by scale normalizing the silhouettes and then using the silhouette models to remove noise and fill in holes at each frame. Our pedestrian silhouette model involves two levels of representations: the pedestrian population representation and the pedestrian

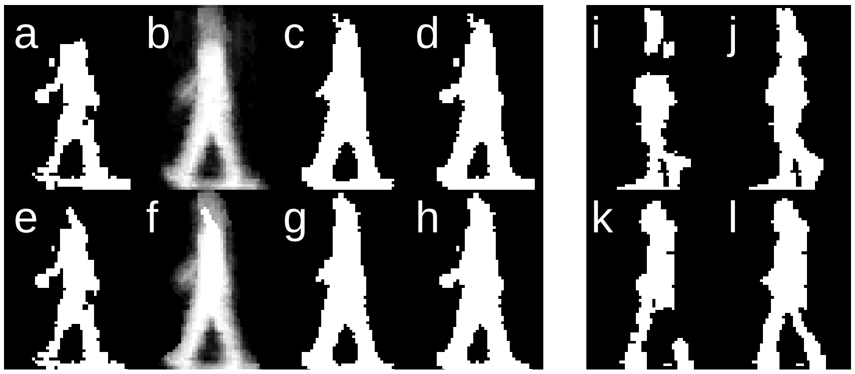


Figure 5-6: *Silhouette Filling Examples:* (a) raw silhouette (b) HMM model for the state most likely to have generated the silhouette (c) mask made by thresholding b (d) logical-OR of a and c. (e)-(h) are the same as a-d, except e is the population-filled silhouette and the HMM in f was trained on the population-filled sequence. (i) another raw silhouette from a different person (j) i after population- and HMM-filling. (k)-(l) same as i and j for a third silhouette.

sequence representation, each requiring a different treatment.

The pedestrian population model, generated by averaging a set of training silhouettes equally representing men and women, is used to refine the raw silhouettes. We can interpret the average as the maximum likelihood estimate of the parameters of a population silhouette generative process. Each pixel location p is an independent Bernoulli process with parameter $\theta_p = p(l_p = 1)$. Given a sequence of silhouettes from a pedestrian, we want to choose a binary value for each pixel location in every frame. We can obtain the posterior distribution of θ_p given the sequence and a prior based on the population parameters. In principle, we could threshold the maximum *a posteriori* value of θ_p. However because the population model prior is only valid for static binary shapes, we can only confidently threshold at pixel locations for which the shape is static across time (i.e., low variance Bernoulli processes). Empirically, we found that restricting the prior to be valid only in the range $\theta_p \geq 0.9$ and $\theta_p \leq 0.05$ worked well. All other pixel locations in the pedestrian silhouette sequence are left unchanged. This set of thresholds gives us the mask shown in Fig. 5-2d. Note that the pixels that are consistently turned on are the ones interior to the pedestrian torso and head region, and the ones that are turned off are far from the edge of the silhouettes, whereas the unchanged pixels are the edge of the silhouettes and the legs.

The pedestrian sequence model, a cyclic silhouette model representing discrete phases of a walking cycle, is used to produce silhouettes that preserve the fine details of an individual pedestrian. This model is trained on each sequence and hence is able to preserve the detailed shape of the silhouette in the sequence.

We begin by training an HMM on the sequence, as described in §5.4.2. Using that HMM, we determine the most likely state assignments for each of the silhouettes using the Viterbi algorithm [85]. To do the filling, we turn on any pixel in a silhouette that has a likelihood of greater than 0.5 in the HMM.

In Fig. 5-6, we see an example of the two filling methods: (a) has no filling, (d) is HMM-filled, (e) is population-filled, and (h) is both population and HMM-filled. In this example, the population-filling recovers part of the head and removes a few spurious pixels. The HMM-filling is able to fill in more of the head and parts of the lower torso. In (i) and (j), we see an example of filling in the entire upper torso and part of the hair for a different person. The legs are filled in for a frame of a third person in (k) and (l).

5.7 Evaluation Methods

To evaluate the quality of our model-based silhouettes, we apply these silhouettes in a gait recognition task. We use two gait recognition algorithms—an existing algorithm described in [61] briefly summarized below, and a distance metric based on the silhouette HMM states.

5.7.1 Ellipse Representation

Our gait dynamics feature vector consists of smoothed versions of moment features in image regions containing the walking person. For each silhouette of a gait sequence, we find the centroid and divide the silhouette into 7 parts roughly corresponding to head/shoulder, arms/torso (front and back), thighs(left/right), and calves/feet(left/right) (see Fig. 5-7(a)). For each of the regions, we fit an ellipse to describe the centroid, the aspect ratio and the angle of the portion of foreground

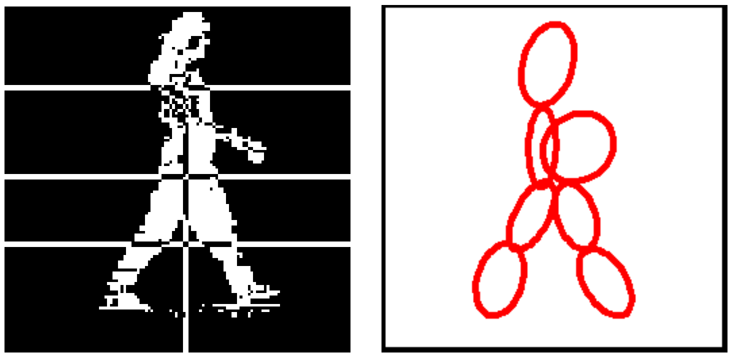

(a) Partition of a silhouette (b) Ellipse fit to each region

Figure 5-7: Computing the Feature Vector for Gait Recognition

object visible in that region(Fig. 5-7(b)).

We assume that all of these features–the centroid, aspect ratio, and angle of each region–are sampled from a Gaussian distribution and compute the mean and standard deviation for each of these parameters across each walking sequence. The feature vector of mean and standard deviation of each region is used in a nearest neighbor classifier to retrieve the identity whose walking dynamics feature vector is closest to the query feature vector.

5.7.2 HMM Representation

In addition to the region-based features, we also use the states of our HMM silhouette model as a gait representation. We use the Euclidean distances between the 8 HMM state observation models as comparison between two gait silhouette sequences.

5.8 Results

Given a set of gait data, we perform the following steps,

1. For each sequence, track the pedestrian and extract a set of raw silhouettes using the algorithm described in §5.5.

2. Build the following pedestrian models:

Silhouette Set Name	Description
S_N	The silhouettes provided with the NIST gait data, generated by USF using a semi-automatic method
S_r	Our fully-automatic raw silhouettes (see §5.5)
S_{d3}	S_r dilated with a neighborhood size of 3
S_{d6}	S_r dilated with a neighborhood size of 6
S_p	S_r cleaned and filled using the population model
S_{Hr}	S_r filled using an HMM trained using S_r
S_{Hp}	S_p filled using an HMM trained using S_p

Table 5.1: *Silhouette Sets:* These are the sets of silhouettes used in the recognition experiments.

- A population model that represents the appearance of all pedestrians. This model is constructed using 100 random raw silhouettes from each of 5 male and 5 female subjects (§5.4).
- A per-sequence HMM that models the silhouette at discrete phases of a walking cycle.

Note that these two models can be constructed independently of each other, or with the HMM following the pedestrian population model.

3. For each sequence, refine the silhouettes using the pedestrian population model and/or the state models of the HMM.
4. Generate a set of region-based gait features for recognition, or use the HMM states directly for recognition.

We applied the above steps to the NIST gait challenge data set, which resulted in a suite of silhouettes and gait features. These silhouettes and gait features were then used in a set of gait recognition tasks.

5.8.1 Silhouette Comparisons

For each gait sequence, we used seven different sets of silhouettes, as described in Tab. 5.1.

The silhouettes that are provided with the NIST gait data are semi-automatically generated in the following process:

1. Manually track the pedestrian in the video sequences.
2. Compute the Mahalanobis distance between the image containing the pedestrian and a background model.
3. Smooth the Mahalanobis distance image with a 9×9 filter.
4. Threshold the smoothed image to obtain the silhouette.

The smoothing process in step 3 has a side effect of smearing out the fine features of the silhouette and possibly removing some features that may be important to the identification of individuals.

Excluding the raw silhouettes, the set of silhouettes that we have chosen fall into two classes, those that reduce noise by a non-model-based process, such as smoothing or morphological operation, which are S_{d3}, S_{d6}, and S_N, and those that reduce noise by a model-based method, as in S_p, S_{Hr}, and S_{Hp}. We will show through gait recognition experiments that the silhouettes generated using a model-based method are consistently better.

For each set of silhouettes, we generate the region-based gait features described in §5.7. In addition, the two types of HMM, generated using S_r and S_p, are also used for gait recognition.

5.8.2 The Recognition Task

The NIST gait challenge data is comprised of gait video of individuals taken under different conditions. A standard set of tests, described in [81], examines the gait recognition rate across different conditions.

The NIST gait data set contains pedestrians walking on different surfaces (concrete and grass), with camera view change (left and right views), and shoe type change. The data set is divided into a gallery set and a number of probe sets. The gallery set contains sequences of pedestrian walking on grass wearing one particular type of

Probe Set	Difference
A (1)	view
B (2)	shoe
C (3)	shoe, view
D (4)	surface
E (5)	surface, shoe
F (6)	surface, view
G (7)	surface, shoe, view

Table 5.2: *Gallery versus Probe Differences:* One gallery (training) sequence is provided for each test subject. Additional probe (test) sequences are also provided where the viewpoint, shoes worn, and/or walking surface are varied. The probes are ordered from easiest to hardest.

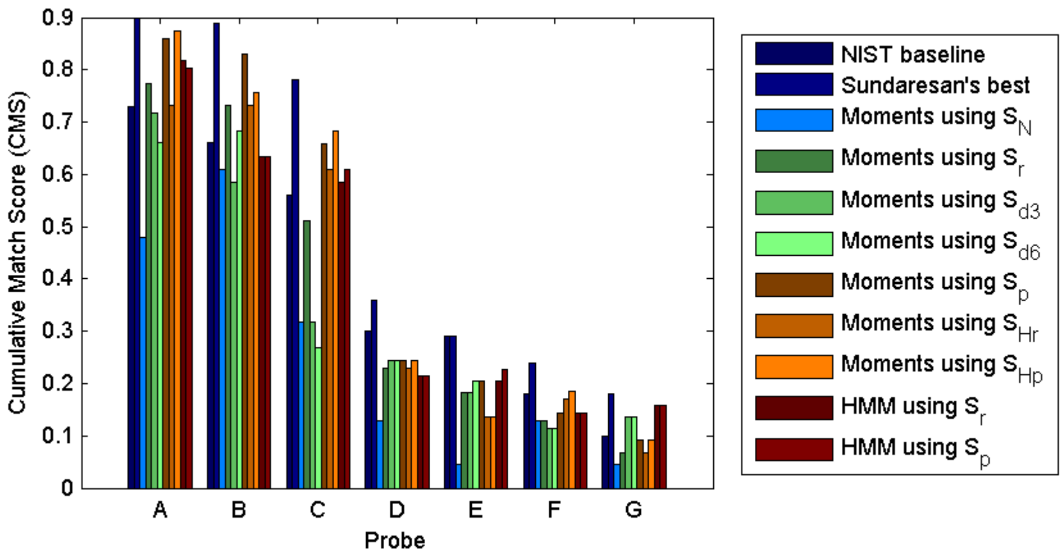
shoes and viewed from one of two cameras. The probe sets differs from the gallery in the ways described in Tab. 5.2.

There are seven corresponding recognition experiments labeled A through G, each testing a probe set against the gallery set. The task of a recognition algorithm is to rank the sequences in the gallery by their distances to the probe sequences. The recognition performance is evaluated using a cumulative match score (CMS), which measures the percentage of probes correctly identified at each ranking.

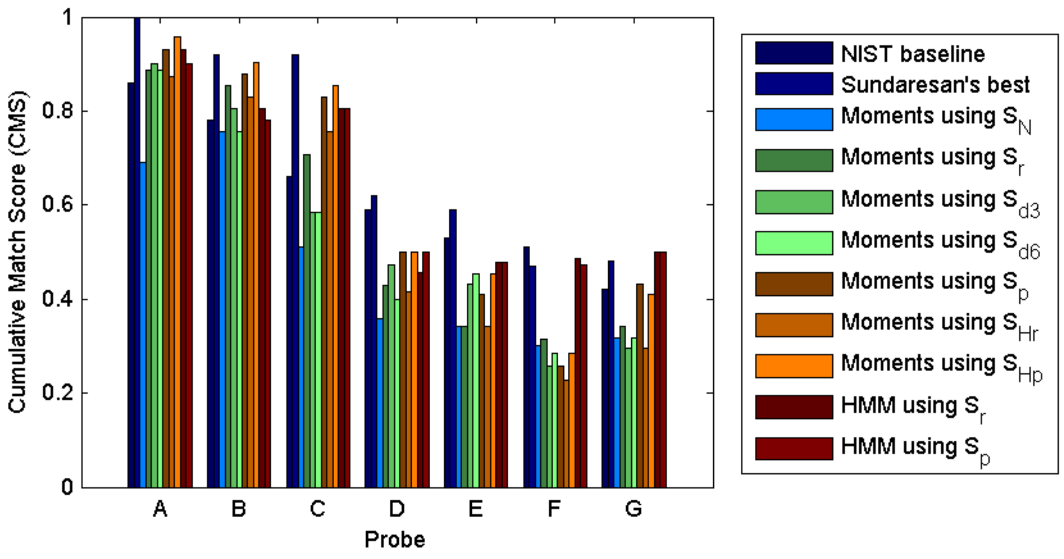
5.8.3 Recognition Results

As in [81], we report the gait recognition rate using the cumulative match score at ranks 1 and 5, as shown in Fig. 5-8. We observe that the experiments D, E, F, and G present the most challenging recognition problems because they all involve a surface change. For them, the feature representation (HMM versus moments) is more important than the silhouette set choice.

We are interested in the question of how well each silhouette type and gait feature type perform in all recognition experiments. To present a clearer picture, we average the CMS for each silhouette type across all recognition experiments A through G, across experiments A through C (the same surface condition), and across experiments D through G (the change-of-surface condition). The surface condition demands further investigation because it is the most challenging test. The averaged CMS are

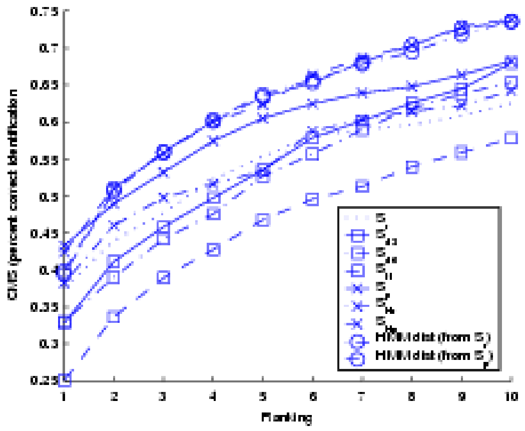


(a) Cumulative match score at rank 1

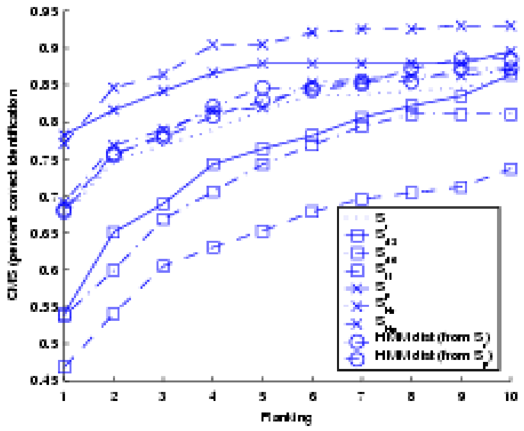


(b) Cumulative match score at rank 5

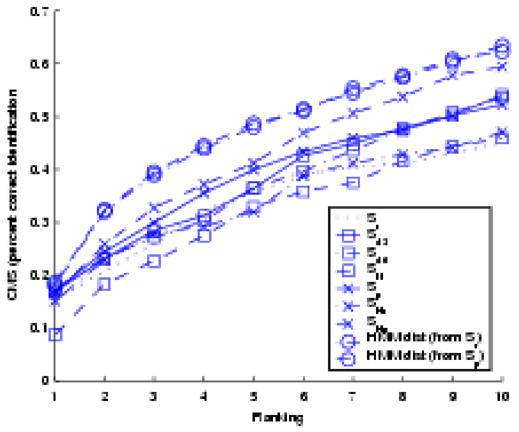
Figure 5-8: *Recognition Rates:* Comparison of recognition rates using different silhouettes using CMS at rank 1 and 5.



(a) Average CMS for all probes



(b) Average CMS for probes A, B, C



(c) Average CMS for probes D, E, F, G

Figure 5-9: *CMS Comparisons:* Comparison of recognition rate using different silhouettes using average CMS over all probes, grass probes, and concrete probes.

shown in Fig. 5-9. The general trends presented in the recognition results are:

- Recognition rates using the region based features on the NIST silhouette set, S_N, are consistently worse.
- Using the region based features, raw silhouettes performed better than their dilated cousins in the same surface condition, but are comparable or slightly worse in the change of surface experiments and the average of all experiments.
- Using the region based features on the S_p and S_{Hp} silhouettes (those that used the population model) resulted in better recognition rates than raw or dilated silhouettes.
- Using the region based features on the S_{Hr} silhouettes (those filled using an HMM trained on raw silhouette sequence) resulted in only marginally better performance than using the raw silhouettes.
- Using the distance of the HMM states, the recognition performances are comparable between the HMMs trained using the raw silhouettes and the HMMs trained on preprocessed silhouettes, S_p. They also performed much better than all other features in the change of surface condition.

5.8.4 Discussion

Our gait recognition experiments above show that incorporating a pedestrian model component, be it using HMM states for recognition or the region features on silhouettes filled with a pedestrian population model, resulted in better recognition rates than the non-model based silhouettes and the raw silhouettes.

Based on the gait recognition performance using region based features on the various silhouette types, we rank, in increasing recognition rate, the quality of the silhouettes as follows: S_{Hr}, S_p, S_{Hp}. Simply using a silhouette model based on one sequence is not adequate because there may be persistent silhouette errors through a large number of frames. These systematic errors in the raw silhouette tend to be caused by lack of contrast between the foreground object and the background environment. The pedestrian population model is able to recover from this type of error because the persistent errors for one sequence are unlikely to persist through a population of pedestrians.

Using the HMM silhouette model is an improvement over using just the pedestrian population model because it is able to improve the estimate of the individual shape over time and capture the appearance of the legs at discrete walking phase.

The recognition rates using the state observation models of the HMM trained on raw silhouettes and the HMM trained on S_p were among the best three algorithms/silhouette data. This indicates that for recognition purposes, HMM silhouette models are robust to some systematic silhouette errors.

5.8.5 Relationship to Other Chapters

In §2, we proposed a method for improving background subtraction results in the face of temporally irregular dynamic textures. In this chapter, we have investigated ways of improving the results of background subtraction by building individual and population appearance models. These models have been primarily used to mitigate camouflaging errors and these techniques are complementary to those in §2. For continuous video sequences under less-controlled circumstances, we could easily envision a system using both set of algorithms.

In §3.2, we saw that background subtraction could detect some people in large crowded environments with adverse lighting conditions like the Grand Central Terminal. Unfortunately, it was not reliable enough to even consider it for tracking a large fraction of the people traveling through the scene. We then saw in §3.4 how a high-quality strong-model pedestrian detector could be implemented efficiently using commodity graphics hardware. Using those tracks, we developed a scene-level activity model in §4. One of the motivations for developing activity models is to be able to identify anomalous trajectories. Given an unlikely trajectory, a variety of security scenarios call for discovering the identity of the person generating the trajectory. One could combine multiple techniques to help make this identification: (a) trace the trajectory using pedestrian detector outputs, (b) determine when the trajectory is isolated from other people and in a region where background subtraction is known to work well, (c) extract silhouettes during the periods when conditions are most amenable to it, then (d) feed the silhouettes into a pedestrian recognition system such as the one described in this chapter.

In §6, we will discuss event detection tasks such as discovering loiterers, theft, and abandoned luggage. In cases where events of interest occur, we could imagine a full site monitoring system integrating pedestrian recognition in a manner similar to the one described above for activity modeling scenarios.

5.9 Summary

We have proposed a method to automatically construct models of pedestrian silhouettes in a walking cycle. Our model contains two components, a pedestrian population based model, and an individual gait silhouette sequence model that is comprised of discrete phase states of walking cycles.

The population model is used to recover from systematic noise of a particular gait sequence.

The sequence model is used to correct for sporadic noise that occur from time to time within a video sequence. This model construction process can be applied to any moving object that exhibits cyclic properties and/or overall shape commonalities that allows one to improve the estimate of shape over time.

Our silhouette models can be used in two ways: to fill in silhouettes for any algorithm that needs accurate silhouette sequences, and to be used directly for gait recognition. In both cases, we have shown that using a model based silhouette extraction is superior to using a non-model based silhouette smoothing algorithm, such as morphological operations, or a smoothing process in the background subtraction phase.

Chapter 6

Robust Modeling for Event Detection in Short Videos

This chapter contains joint work with Xiaogang Wang [23].

6.1 Introduction

Beginning in the late 1990s, some key advances were made by researchers such as Stauffer and Grimson [101] and Haritaoglu *et al.* [44] that allowed the construction of full realtime farfield visual tracking systems on commodity hardware. Work has continued to progress in handling more challenging low-level situations such as variable lighting and dynamic backgrounds. With those improvements, researchers have been able to also make progress on higher-level tasks such as directly detecting events of interest[1]. In this chapter, we will focus on situations where the events of interest are well-defined, there is limited data available, and algorithm customization and tuning time is restricted. This mirrors situations in the real world where ad-hoc monitoring systems are deployed in response to immediate needs.

In the introductory chapter (§1.5.1) we discussed a body of existing work for

[1]In §4, we examined activities defined as clustered path segments. In this chapter, when we speak of event detection, we refer to identifying when a collection of objects jointly satisfy a collection of spatial, temporal, identity, and relational constraints. For example, an abandoned luggage event may be defined as a human being separated from a non-moving bag for enough time and distance.

event detection that can be divided into two classes. In one class is research focused on representing and modeling events themselves, with a focus on interpreting long continuous video feeds. We also mentioned work done on datasets produced for the Performance Evaluation of Tracking and Surveillance (PETS) workshops. The challenge problems associated with the most recent datasets emphasize the vision components of event detection systems by coupling challenging real-world data with strict evaluation guidelines. The supplied videos tend to be relatively short, which penalizes approaches that require a significant amount of training data or long bootstrapping periods. Researchers are given a single month from the release of the challenge problem to the submission of results and a paper for peer review.

For the PETS 2006 workshop [29], a standardized dataset was created to evaluate various automatic visual event detection systems. Seven videos were taken from each of four calibrated cameras overlooking a train station platform. Each video set recorded a left-luggage event that the automatic visual surveillance systems were expected to detect. Specularities, shadows, a partially-reflective glass surface, and mutual occlusions from multiple actors provided varying levels of difficulty depending on the camera views used and the individual staged scenarios.

Nearly all of the accepted PETS 2006 papers used background subtraction and/or motion detection to first identify foreground blobs. del-Rincón *et al.* [68] used multiple time scales and feedback loops to improve robustness. Lv *et al.* [67] and Grabner *et al.* [41] used the background subtraction results for static object detection and then used and/or learned a classifier for humans.

Auvinet *et al.* [5] took the foreground blobs and projected them onto the ground-plane, looking for multiple silhouette intersections from the four cameras. Li *et al.* [65] used a layered model to track objects across time. Lv *et al.* used Kalman filters on the human classifier output, falling back to meanshift [14] when necessary. del-Rincón *et al.* used an unscented Kalman filter to track the owners of discovered static objects while Smith *et al.* [97] used a full MCMC sampler. Krahnstoever *et al.* [57] and Auvinet *et al.* [5] showed results using nearest neighbors for their data association.

The PETS 2007 workshop's dataset was staged similarly to the 2006 one, adding

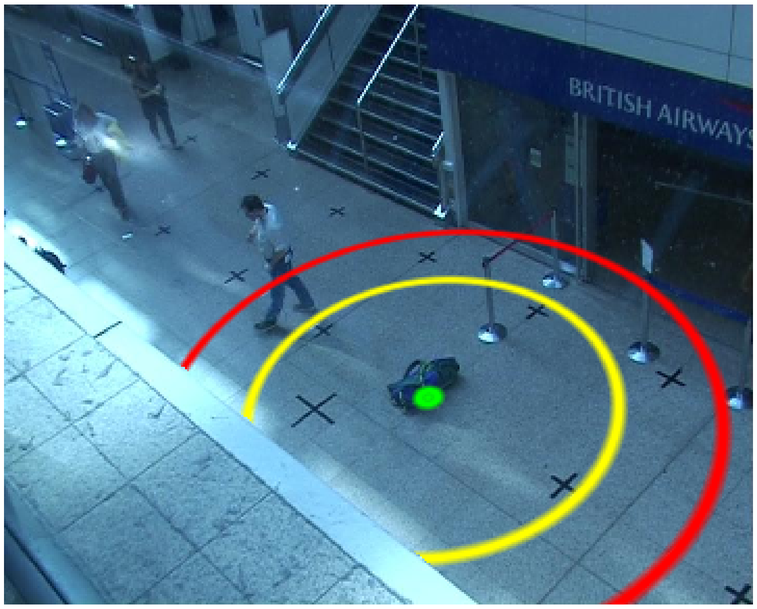


Figure 6-1: *Actor Entering the Abandoned-luggage Warning Zone:* In this image (frame 1120, camera 3, clip *S08*), we mark the location of a dropped piece of luggage with a green dot. The owner has just left the $2m$-radius area about the bag defining a warning zone indicated by the yellow circle. After remaining outside the red circle ($3m$) for more than 15 seconds, an abandoned luggage alarm should be generated.

several interesting real-world challenges:

- more inter- and intra-clip lighting changes,
- a changing mixture of harsh and soft shadows,
- camera movement between clips,
- denser pedestrian traffic,
- lower effective resolution in a key camera view,[2] and
- a broader set of events to detect.

In this chapter, we will be using the PETS 2007 dataset, summarized in Tab. 6.1 and below.

The dataset consists of 10 video clips from each of four calibrated PAL cameras (720×576, 25fps, interlaced). The *BACKGROUND* clip is a 1000-frame clip with

[2]In the PETS 2007 data, the most overhead view (camera 3) recorded interlaced video, whereas in PETS 2006, the camera for the best view was progressive scan.

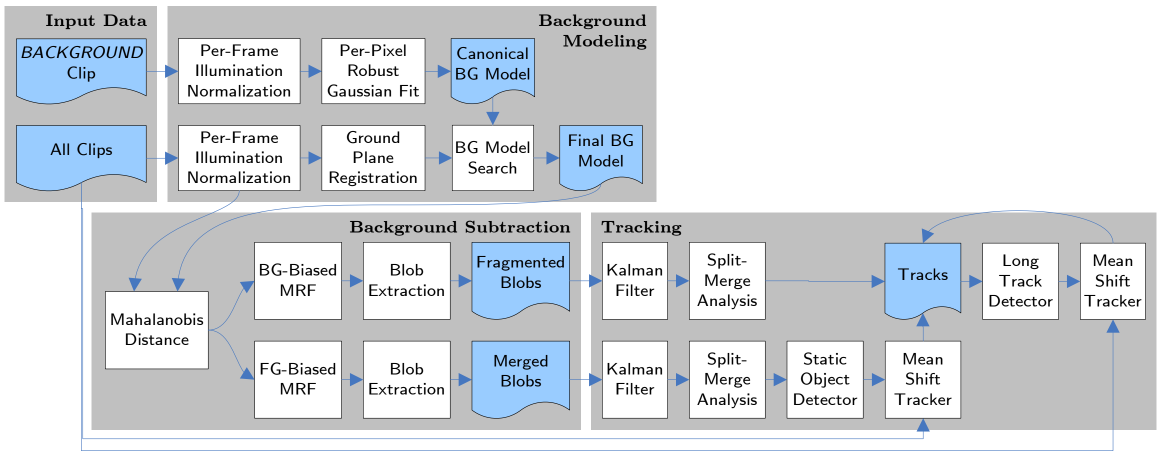


Figure 6-2: *Event Detection Pipeline:* This diagram outlines the primary data processing steps in our tracking pipeline. The motivation behind and implementation of each block is described in §6.2–§6.4.

Clip	Description	Subjective Difficulty	Notes
	BACKGROUND	N/A	contains sparse pedestrian traffic and no events of interest, meant for training
S00	No Defined Behavior	N/A	contains sparse pedestrian traffic and no events of interest
S01	General Loitering 1	**	contains a single actual loitering event
S02	General Loitering 2	***	contains a single actual loitering event, with heavier occlusion
S03	Swapping Bag 1	**	contains two scripted loitering events, one unscripted loiterer, and a luggage ownership transfer designed to overly simple systems
S04	Swapping Bag 2	****	contains three scripted loitering events, the luggage ownership transfer scenario, and heavier occlusion
S05	Theft 1	**	contains two scripted loitering events, and a luggage theft event (with an implicit luggage abandonment event)
S06	Theft 2	****	contains a single luggage theft event, with heavy occlusion
S07	Left Luggage 1	**	contains one luggage abandonment and one theft event
S08	Left Luggage 2	****	contains one luggage abandonment and one theft event, with heavy occlusion

Table 6.1: *PETS 2007 Clips:* This is a summary of the 1 training and 8 test clips included in the PETS 2007 dataset.

sparse pedestrian traffic, provided to allow for algorithm training, as needed. The remaining clips have durations of 2750 to 4500 frames. *S00* is a 4500-frame control sequence in which none of the defined events occur. *S01* and *S02* were designed to contain one staged loitering event each under easy and hard conditions, respectively. For this dataset, loitering is defined as remaining in the scene for more than 60 consecutive seconds. *S03* and *S04* contain easy and hard staged luggage retrieval events where a group of two people enter and a bag is placed on the ground. Both members of the couple stay near the bag and then later the second person picks up the bag and the couple leaves together. *S05* and S06 each contain an example of theft, where someone other than the owner picks up a bag that was placed on the ground by the original owner. In *S07* and *S08*, a person drops a bag and "abandons" it by moving more than 3m away for more than 15s (see Fig. 6-1)[3].

Before proceeding to describe our method, we note a few details. Our approach is purely monocular and we use camera 3 (see Fig. 6-1) exclusively because it offers the viewpoint with the fewest inter-human occlusions. Because this camera's video was interlaced, we subsample the input video without smoothing down to 360 × 288. Our algorithm is based on generic blob tracking techniques that are readily implemented and require little training and tuning relative to ones that use strong human appearance models. For all tunable parameters, we used the same settings for all clips.

We have implemented a tracking system to detect this workshop's events. Similar in spirit to the work of del-Rincón *et al.* in PETS 2006, our system is attention-based. Using background subtraction (our attention mechanism), we identify (a) likely dropped luggage and (b) long spatially isolated human tracks. When dropped luggage appears, we perform a local spatio-temporal search for the human owner. Humans identified by long tracks or by association with luggage then have their tracks temporally extended via meanshift. Our system is able to accurately detect nearly all of the events that occur in the dataset with no false positives, including

[3]Originally, the time period was 25s, but it was later changed to 15s because the actors did not stay away from their luggage for strictly more than 25s

some actual loitering events that were omitted from the official ground truth.

Our processing pipeline is illustrated in Fig. 6-2. In §6.2, we discuss our background modeling approach and why the traditional approach is insufficient for this dataset. We then detail our foreground/background segmentation algorithm in §6.3 and tracking in §6.4. Our event detection rules are described in §6.5 and its results are given in §6.6. We summarize our system in §6.7.

6.2 Background Modeling

The first step in our processing pipeline is background modeling. Our goal in this stage is to build, for each pixel in each clip, a model of the appearance of the static elements in the scene. The most common approaches to solve this type of problem are to adaptively model the background as a mixture of Gaussians (c.f. Stauffer and Grimson [101]) or using a kernel density estimate, as done by Mittal and Paragios [72], as we discussed in Chapters 1 and 2. Unfortunately these approaches cannot be used directly for datasets like the PETS 2007's because they make the fundamental assumption that the most common color modes for each pixel correspond to the background: the clips recorded for PETS are short and many have loitering events where one or more people remain in nearly the same location for almost the entire clip. In fact, in *S02*, some pixels view the background less than 10% of the time.

A natural approach would be to use the *BACKGROUND* clip for training a standard background model to be used directly and without adaptation in the test clips. *BACKGROUND* is indeed mostly background for all important pixels, but its lighting is significantly different from all of the other clips, as we will see when we discuss Fig. 6-3. For most of the clips, the overall illumination is lower, and the locations of bright highlights on the walls and floor move. Additionally, camera 3 was moved slightly for clips *S05* through *S08*, causing difficulties for background subtraction approaches that rely critically on known registration.

An alternative to starting with a foreground/background segmentation is to simultaneously learn the appearance and extent of all objects and the background. Unfor-

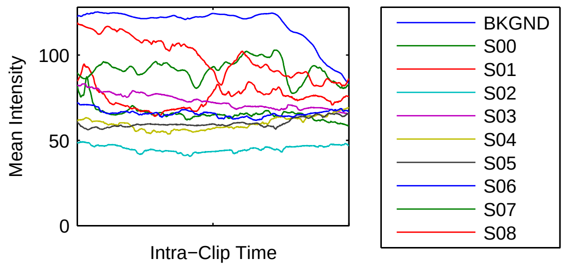


Figure 6-3: *Dramatic Inter- and Intra-clip Intensity Changes:* Here we plot the average intensity (on a scale of 0 to 255) of each frame over time, for each video clip. We note that there are significant inter-clip changes (e.g. *BACKGROUND* vs. S02) and intra-clip changes (e.g. *BACKGROUND* and *S08*).

tunately, full layered model alternatives either make modeling assumptions that are too strong for this dataset and are very computationally expensive [52, 119] or they bootstrap off background subtraction [65, 129]. Also, most existing implementations are inappropriate for very busy scenes with frequent occlusions and changes to layer depth ordering.

6.2.1 Implementation

We now give the details of our background modeling implementation as outlined in Fig. 6-2.

As already mentioned, one of the challenges of this dataset is non-constant illumination, as we now illustrate in Fig. 6-3. Even when just observing the mean intensity of the scene, we can see large differences over time within many clips as well as large differences between clips. The lighting effects we observe are consistent with natural lighting during a partially-overcast day with slowly passing cloud cover. To reduce these effects, we illumination-normalize each video frame, by using the color $\tilde{\boldsymbol{c}}_p^{(t)}$,

$$\tilde{\boldsymbol{c}}_p^{(t)} = \left(\Sigma_{img}^{(t)}\right)^{-\frac{1}{2}} \left(\boldsymbol{c}_p^{(t)} - \bar{\boldsymbol{c}}^{(t)}\right), \tag{6.1}$$

instead of the original RGB value $\boldsymbol{c}_p^{(t)}$, where p is the pixel location, t is the current frame number, $\bar{\boldsymbol{c}}^{(t)}$ is the mean RGB color vector for the frame, and $\Sigma_{img}^{(t)}$ is the frame's diagonal RGB covariance matrix. The remainder of the background modeling is done independently for each pixel.

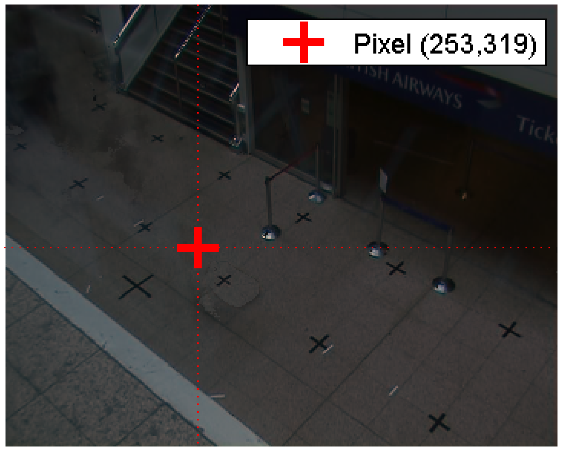
Since all pixels of interest in the *BACKGROUND* clip are sampled from the background distribution nearly all of the time, we fit a single joint Gaussian distribution to each pixel's illumination-normalized RGB value. To add a small measure of robustness to foreground pixels that are present, we discard any pixels with a Mahalanobis distance greater than 5 from the Gaussian and then refit the distribution to the inliers. To avoid overfitting, if any eigenvalue in the covariance matrix is less than 0.002, we round it up to that value. Minimal tuning was used to select these parameters. The output from this step is a canonical background model for each pixel location.

If the lighting changes were more mild, we could directly use our canonical model in all clips; however, our illumination normalization is insufficient to be used directly in these clips. To illustrate the issue, refer to Fig. 6-4, where we examine the observed colors when a given pixel views foreground objects nearly 90% of the time. For that pixel in clip *S02*, we have manually labeled all frames as foreground or background, and observed the normalized color distributions. The black set of axes represent the canonical background model. We can see that that model poorly represents the hand-labeled background distribution (red X's), but it is much closer to the true background mode than to any modes from the foreground distribution (blue dots).

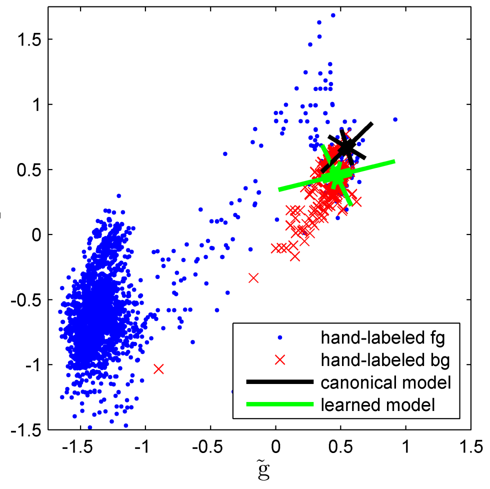
In essence we wish to track the background's color distribution between clips. We assume (a) that the background is well-modeled by a Gaussian, (b) that the Gaussian distribution's mean and covariance shift slowly over time, and (c) that the background distribution is distinctive from the modes in the foreground distribution. Given these assumptions, we wish to find the Gaussian mode in the new clip's distribution that is closest to the canonical model, as suggested in the previous paragraph. In doing so, we wish to be agnostic to the observed foreground distribution and only concern ourselves with finding the closest color mode.

We can simplify our search by warping our illumination-normalized pixel values so the canonical model is represented as an origin-centered unit normal distribution:

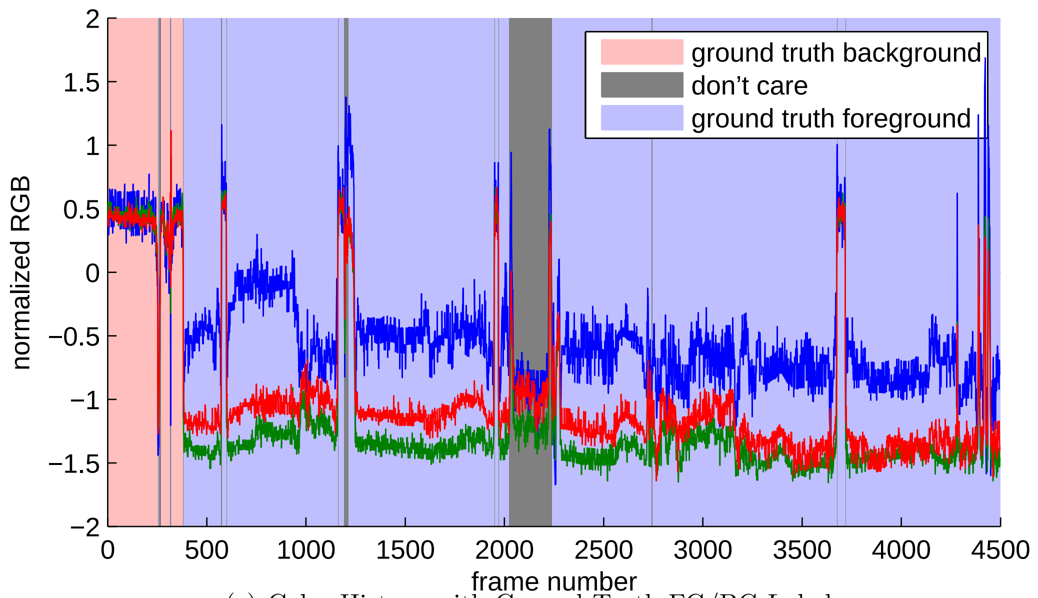
$$\tilde{\tilde{\boldsymbol{c}}}_p^{(t)} = \Sigma_{BG,p}^{-\frac{1}{2}}(\tilde{\boldsymbol{c}}_p^{(t)} - \bar{\boldsymbol{c}}_{BG,p}) \tag{6.2}$$



(a) Pixel at $(253, 319)$ in clip *S02*



(b) Adaptation



(c) Color History with Ground Truth FG/BG Labels

Figure 6-4: *Background Model Adaptation for a Challenging Case:* For pixel $(253, 319)$ in the *S02* clip, we labeled all frames in which the pixel was viewing the background versus any foreground object. In Fig. 6-4(a), we show the pixel location in question on a rendering of the background model. In Fig. 6-4(b), color values when the pixel was viewing background are shown as red X's in a scatter-plot (projected onto the normalized red and green axes). Blue dots represent observed foreground color samples. For this pixel, the Gaussian distribution learned for the *BACKGROUND* clip is shown as a black-colored set of principle axes. The green axes represent the learned model after it was adapted to this clip. If we construct a Gaussian classifier from the learned model for this pixel, the area under the ROC curve is 0.9926 (not shown). In Fig. 6-4(c), we show the illumination-normalized color history for the pixel with the ground truth annotations in the background. The pixel analyzed in this figure is a particularly challenging case as it is viewing foreground objects nearly 90% of the time.

where we search for a mode in the $\tilde{\bar{\boldsymbol{c}}}_p^{(t)}$ space, given the canonical model's center $\bar{\boldsymbol{c}}_{BG,p}$ and covariance $\Sigma_{BG,p}$.

We then iteratively slide a spherical template of radius 5 to the mean location of the sample points that fall within the template, until convergence[4]. We then fit a Gaussian distribution to the samples that fall within the final template boundaries and limit the covariance eigenvalues as we did for the canonical model. The output from this step is a single-Gaussian background model, $\left(\hat{\boldsymbol{c}}_p, \hat{\Sigma}_p\right)$, for each pixel, tuned for each video clip, and defined in the illumination-normalized RGB space.

For the PETS 2007 dataset, we have used the method just described to track the background distribution from the *BACKGROUND* clip to other clips. We have found this novel implementation to be effective and to require minimal tuning.

6.3 Background Subtraction

Given a statistical model of the background, we can perform likelihood tests to classify sampled pixels as foreground or background, as is standard practice. Markov Random Fields (MRFs) are an effective mechanism for applying spatial smoothing priors to a label field [38] instead of relying on a purely independent thresholding at each pixel. As described in §1.2, our MRF optimizes the following objective function:

$$E(\boldsymbol{l}) = \sum_{\{p,q\}\in\mathbb{N}} \mathrm{V}_{p,q}(l_p, l_q) + \sum_{p\in\mathbb{P}} \mathrm{T}_p(l_p) + \sum_{p\in\mathbb{P}} \mathrm{D}_p(l_p) \tag{6.3}$$

where $\boldsymbol{l} = (l_1, ..., l_{\|\mathbb{P}\|})$ is the field of foreground-background labels, $\mathbb{P}$ is the set of pixel sites, $\mathbb{N}$ is the 8-neighborhood graph, $\mathrm{V}_{p,q}(l_p, l_q)$ encourages spatially neighboring pixels to have the same label, $\mathrm{T}_p(l_p)$ encourages pixels to have the same foreground/background label they had in the previous frame, and $\mathrm{D}_p(l_p)$ encourages pixels to be labeled as foreground when they do not match the background model

[4]The template radius was tuned by selecting a single predominantly-foreground pixel location in each of two test videos and performing a quick ROC analysis. The results are not especially sensitive to this radius.

well. These energy terms are defined as

$$\mathrm{V}_{p,q}(l_p, l_q) = t_N \delta(l_p, l_q) \tag{6.4}$$

$$\mathrm{T}_p(l_p) = t_T \delta\big(l_p, l'_p\big) \tag{6.5}$$

$$\mathrm{D}_p(l_p) = \begin{cases} t_F, & \text{if } \Big(l_p = 1\Big) \vee \Big(t_F < \min_{i=1}^{b_p} \mathrm{d}^2(\boldsymbol{c}_p; \boldsymbol{\mu}_{qk}, \Sigma_{qk})\Big); \\ \min_{i=1}^{b_p} \mathrm{d}^2(\boldsymbol{c}_p; \boldsymbol{\mu}_{qk}, \Sigma_{qk}) & \text{otherwise,} \end{cases} \tag{6.6}$$

where $\delta(\cdot,\cdot)$ is the Kronecker delta function, t_N is the spatial mismatch potential, t_T is the temporal mismatch potential, l'_p is the label assigned to pixel p in the previous frame, and t_F is the foreground label potential. The background potential (the "otherwise" case) in $\mathrm{d}_p(\cdot;\cdot,\cdot)$ is the Mahalanobis distance from the learned background model for the given pixel in the given clip.

We have used an existing MRF implementation that uses the fast two-label *st*-cut implementation of Boykov, Veksler, and Zabih [9, 10, 56] with the temporal smoothness terms suggested by Migdal [70]. For other background subtraction work, we have found $t_T = t_N = 5$ and $t_F = \log(256^3) \approx 16.6$ to be good default values. We have used those t_T and t_N values without any tuning. We next discuss how we chose t_F.

Instead of choosing a single value for t_F for the PETS dataset, we construct a pair of MRFs to address two different detection tasks. One is biased to produce more foreground labels ($t_F = 8$) and the other is biased to produce more background labels ($t_F = 32$). These values were chosen empirically by starting with the baseline value of $t_F \approx 16.6$ and observing foreground/background classification results on a handful of frames with a few different settings.

As demonstrated in Fig. 6-5, the foreground-biased MRF (left) tends to capture all of the foreground into coherent blobs at the cost of mislabeling shadows as foreground and joining independent objects. On the right, we see the same frame segmented with the background-biased MRF. Its silhouettes are cleaner and distinct, but camouflaging

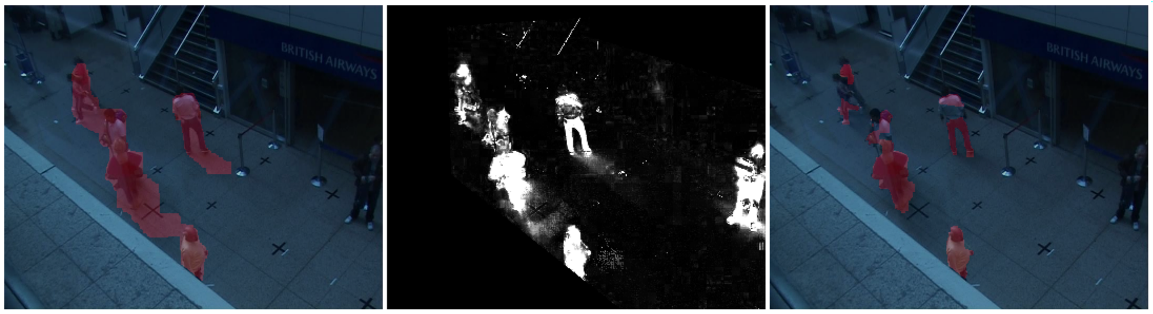


Figure 6-5: *An Extreme FG/BG Segmentation Example:* Here we show segmentation results for the foreground- (left) and background-biased (right) MRFs for frame 500 of clip *S08*. The frame shown here is a somewhat extreme example of the different solutions found by the two MRFs. Both MRFs use the Mahalanobis distance values visualized in the middle image as the background label potential. For display purposes, the intensity is saturated at a distance of 64, twice the foreground potential for the background-biased MRF (so a 50% gray pixel sits on the classification threshold, absent any neighborhood effects).

effects have caused some objects to be split into multiple blobs.

When attempting to detect dropped luggage, we wish to obtain robust detections of relatively small isolated objects. The foreground-biased MRF resists camouflaging effects of humans that result in fragmented blobs. Thus when a small foreground blob is present in its solution, there is a high likelihood that it was actually the result of a small object, not a blob fragment from a human.

For the foreground-biased MRF segmentations, we extract blobs using a standard 4-connected neighborhood connected components extractor. Any blobs that have fewer than 75 pixels are discarded. Before evaluating the MRF, we mask out regions of the image corresponding to the ledge on the bottom left of camera and corresponding to the "British Airways" sign and above on the wall. No humans can move here and these areas are especially susceptible to lighting changes in some clips. We also reject any blobs that are very near the edges of the image because they are unreliable for tracking purposes. The 75-pixel threshold was chosen with minimal tuning and could be made more robust by scaling it by the square of the estimated distance to the camera using its calibration. For this dataset, we did not find it necessary to take this extra step.

When attempting to detect individual humans who are present in the scene for a long time, tracking at a blob-level only works well when the blobs are disjoint. With the density of human traffic in the PETS 2007 dataset, our foreground-biased MRF tends to group multiple people into a single blob very frequently and it also mislabels shadows as foreground. By biasing a second MRF to prefer background labels, we obtain clean silhouettes which can be tracked more easily, at the cost of needing to be robust to some blob fragmentation.

For the background-biased segmentations, we extract blobs by grouping any 4-connected blobs that are within 10 pixels of another blob. The 10 pixel dilation diameter was chosen by observing differences in the MRF segmentations in a handful of frames from two clips.

The output of the background subtraction is a collection of segmented foreground blobs from each MRF. We keep both sets of blobs separately since they are redundant and optimized for different tracking tasks (dropped luggage versus human tracking).

6.4 Tracking

After extracting the foreground blobs, we need to do enough tracking to detect the desired events: loitering, luggage abandonment, and theft. These events rely primarily on (a) tracking and maintaining the identity of humans who remain in the scene for a long time, (b) detecting luggage placed on the ground, (c) identifying who owns a piece of dropped luggage, and (d) identifying those who pick up luggage. Because the clips all have many actors who occlude each other, simple blob tracking without appearance information cannot succeed. One approach is to explicitly build strong models and attempt to track all humans, as was done by Grabner *et al.* and Lv *et al.* for PETS 2006. We have chosen instead to build an attentional system that identifies (a) proactive opportunities to build robust models and (b) times when more extensive tracking are desired.

In this section, we provide a detailed description of our tracking algorithms along with chosen parameter values. These parameters are specific to this dataset, but the general approach we take is applicable to other event detection scenarios, especially in cases where there is (a) too much crowding in the scene to just use blob trackers, and (b) insufficient data (or manual annotations of data) to use a training-based approach. Due to the lack of independent training data, we are forced to use some of the test data for tuning our algorithms if we wish to evaluate it on all of the test sequences using the same parameter settings. We include the specific parameter settings and how they were chosen in an effort to demonstrate that we have chosen them conservatively.

Our first tracking submodule takes the background-biased foreground blobs and performs standard Kalman tracking on them, using a constant velocity model on the blob's centroid in the image plane. During data association, we independently associate each track with the blobs in the current frame. If multiple tracks are associated with a single blob, we initialize a new track to follow the merged group as long as it remains coherent. This track is tagged as containing a group of actors. We similarly detect tracks that split into multiple blobs. Although we did not find it necessary for this dataset, it is possible to use trajectory and/or appearance information to disambiguate mild to moderate split-merge graphs using approaches such as that of Bose *et al.* [8]. At greater computational cost, an MCMC tracker such as the one used by Smith *et al.* [97] could be used as well. For this dataset, we used a simplified version of the implementation of Bose *et al.* In other datasets where the objects of interest appear as isolated blobs for non-trivial periods of time, a base level blob tracker such as this one can be useful. We note that this submodule is not expected to track through crowds or occlusions.

Our second tracking submodule identifies loitering candidates. When we observe a non-group track that lasts more than 16 seconds and whose mean blob area is between 1500 and 3000 pixels (in a 360×288 frame), we consider this track as a good candidate. These are very loose thresholds. We use the pixels in that person's tracked blob to learn a color histogram appearance model for that person. We then use meanshift tracking [14] to temporally extend the track both forward and backward through occlusions, dropouts, and merge-split events. We stop temporally extending meanshift tracker when the Bhattacharyya distance,

$$\mathrm{BC}(p,q) = \sqrt{1 - \sum_{\boldsymbol{c} \in [0,15]^3} \sqrt{p(\boldsymbol{c})q(\boldsymbol{c})}} \tag{6.7}$$

exceeds some threshold, $\tau_{meanshift} = 0.14$, where $\boldsymbol{c}$ is an RGB color value, p is the color histogram model, and q is the color histogram of the pixels in the putative object location. Our color histograms have 16^3 bins. We have used an existing meanshift tracker implementation. This second submodule works because the first blob tracker was able to collect enough data to build a stronger appearance model. In more general situations, it is common to have a weak model that can track some objects with very high confidence (and high precision) but low recall. One can then bootstrap a stronger model that increases the recall. In our situation, the stronger model does not increase the number of pedestrians detected, but it does increase the number of frames in which a given person is tracked.

If a tracked object disappears at a scene boundary, we terminate the tracking, but we retain the meanshift model. This allows us to reacquire targets that reenter the scene soon after leaving. This stronger model that was bootstrapped allows us to do this extra task of target reacquisition.

The third tracking submodule performs Kalman tracking on the foreground-biased blobs. Humans walking near each other are often grouped in the same blob with this tracker, so it is less effective at identifying individuals. It is however robust to long-term camouflage effects with humans. This means that a given person rarely produces a track fragment that represents only a small portion of their body for more than a few frames at a time. When people drop luggage and leave it on the ground for an extended period of time, they often are segmented separately from the dropped luggage for a second or more at a time. This presents us with an opportunity to detect these static objects. When we see a track with detections between 200 to 1000 pixels of area per observation, we hypothesize that it is a piece of dropped luggage. As was done with the long isolated human tracks, we initialize a meanshift tracker using appearance information from the initial track. We then extend the track temporally in both directions until significant movement ($\tau_{ds} = 50$ pixels) is observed. This gives us the drop-off and pickup times. In more generic site monitoring situations where multiple object classes need to be tracked, this kind of injection of top-down knowledge can improve results. Different low-level detection mechanisms can be tuned for the different classes.

Given a drop-off and pickup time and a location for a piece of luggage, we would like to identify the individual who has initiated each event. If a human-sized blob overlaps the bag's meanshift template area when it moves by more than τ_{ds} pixels from its starting position, we assume that that this person is moving the bag. If we were searching back in time with meanshift, this blob is the original owner (or victim in the case of theft), and when searching forward in time, the blob is the new owner (or thief).

To disambiguate theft, luggage swaps, luggage that always stays near the owner, and dropped luggage, we need to track the original and new owners. If either of these people has already been associated with a loitering track, no extra work is needed. Otherwise, we use a new meanshift tracker for each person to discover their long-term trajectory.

Any time we have two temporally-separated full tracks (those which we extended with meanshift trackers), we compare the color histograms that were used for the meanshift tracking. If the Bhattacharyya distance is less than $\tau_{reacquire} = 0.15$, we consider the two tracks to belong to the same person.

6.5 Event Detection

As described in the previous section, the attentional tracker is configured to output the primary pieces of information required for event detection. We are able to translate the events from image coordinates to real-world coordinates by assuming the middle of the bottom of the blob or meanshift tracker's bounding box is touching the ground. Using the supplied camera calibration, we are able to infer the world coordinates of this point.

	Loitering			Abandoned Luggage			Theft			Subjective
Clip	TP	FN	Error	TP	FN	Error	TP	FN	Error	Difficulty
S00	-	-	-	-	-	-	-	-	-	**
S01	1	-	5.08s	-	-	-	-	-	-	**
S02	1	-	1.44s	-	-	-	-	-	-	***
S03	2	1	8.22s	-	-	-	-	-	-	**
S04	1	2	0s	-	-	-	-	-	-	****
S05	1	1	19.2s	1	-	0.08s	1	-	0.08s	**
S06	-	-	-	-	-	-	-	1	∞	****
S07	-	-	-	1	-	0.12s	1	-	0.08s	**
S08	-	-	-	1	-	0.2s	1	-	0.12s	****

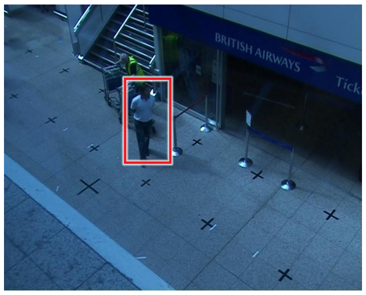
Table 6.2: *Event Detection Results (PETS 2007):* Key frames from each clip are shown in Fig. 6-6–6-11. TP is the number of true positive event alarms and FN is the number of missed alarms. In the PETS 2007 dataset, events of interest have a well-defined start time (e.g. a loitering alarm should sound the moment a person has been in the scene for 60 consecutive seconds). The temporal errors reported here are the absolute value of the time we raised an alarm minus the time we should have raised it, according to the supplied ground truth. Our system had no false positives for this dataset. The subjective difficulty was defined by the PETS 2007 organizers.

Any individual tracks that have a human-like size and aspect ratio and exist for more than 60 seconds trigger loitering events. If the owner of a piece of luggage travels more than 2 meters away from their luggage, a warning is triggered, and if they stay more than 3 meters away for more than 15 or 25 seconds (depending on the scenario, as indicated in the ground truth), an abandoned luggage alarm is triggered. If a new owner removes a piece of luggage beyond the 3 meter radius, a theft alarm is triggered after 15 or 25 seconds, as appropriate for that clip. If the new and original owners both exit the scene together, we have chosen to output a reattended alarm.

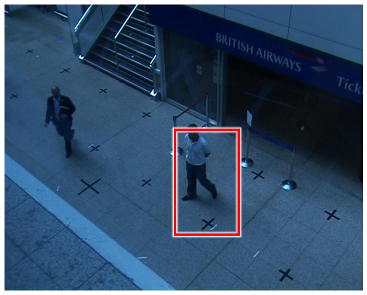
As the events of interest are precisely and deterministically defined for this dataset, our system attempts to directly detect the required conditions for alarms. If a more sophisticated probabilistic event model were desired, it could be substituted.

6.6 Results

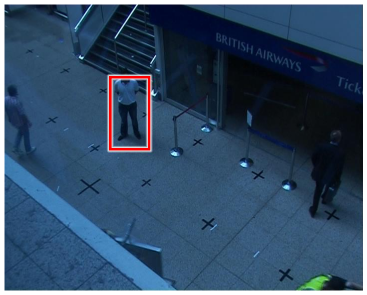
In this section, we briefly summarize our results on the PETS 2007 dataset. Before doing so, we highlight our strategies for parameter tuning. Because only 9 video clips



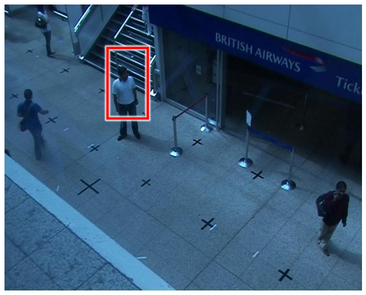
(a) 0400



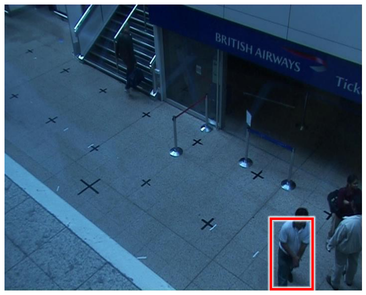
(b) 0800



(c) 1200

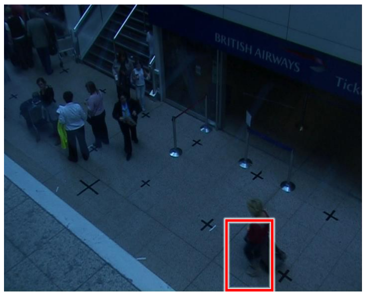


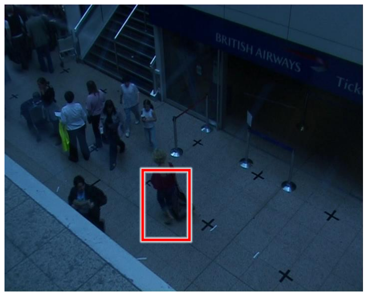
(d) 3000

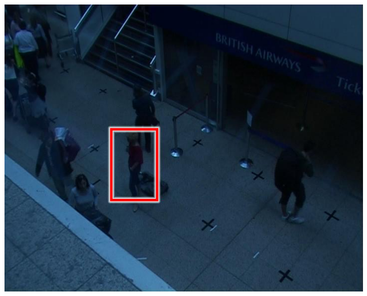


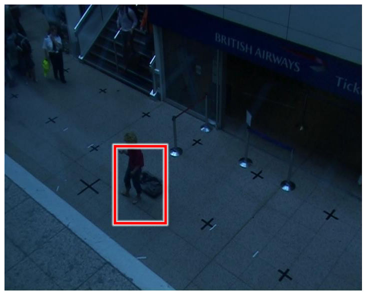
(e) 3900

Figure 6-6: S01 *Key Frames:* This clip contains a single loiterer. We show our tracker's estimate of the loiterer in several manually-selected key frames as a red bounding box. The subfigure labels indicate the frame number, for those wishing to compare and/or replicate our results.


(a) 0244


(b) 0296


(c) 1097


(d) 3143

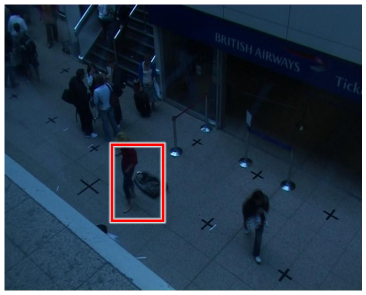
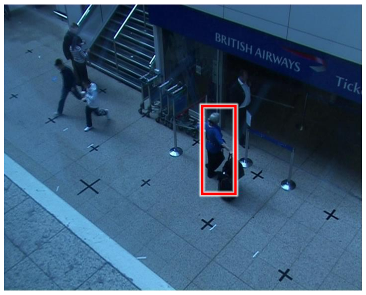
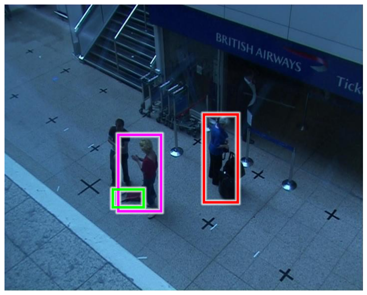

(e) 3984

Figure 6-7: S02 *Key Frames:* This clip contains a single loiterer, but heavier occlusions.


(a) 0000


(b) 0977

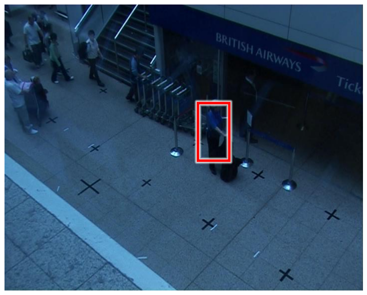
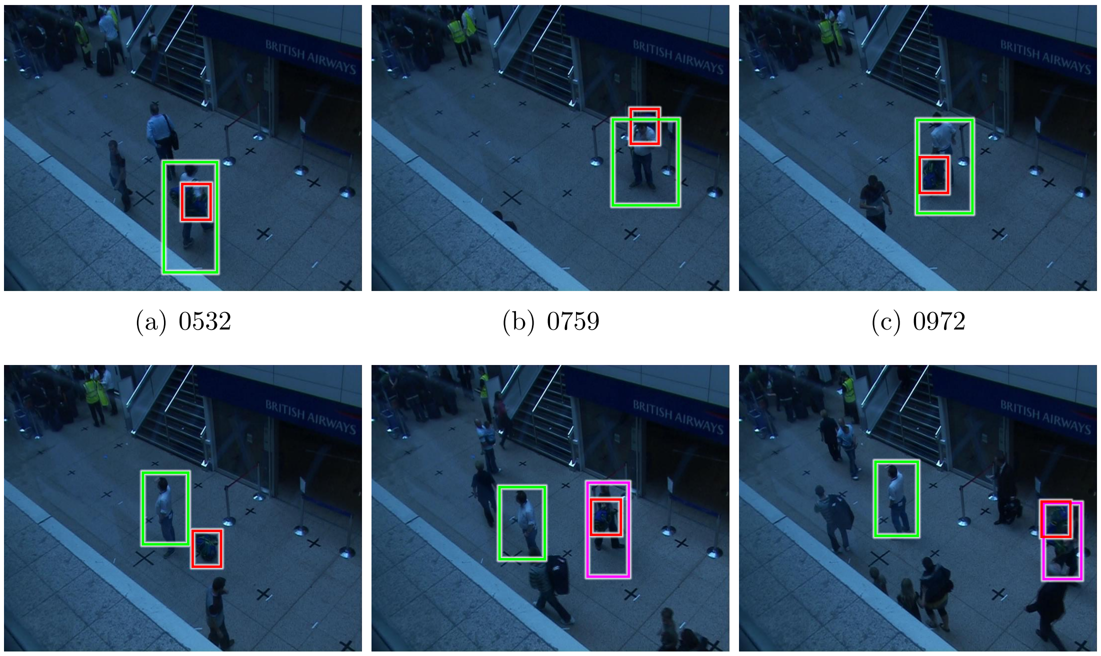

(c) 2967

Figure 6-8: S03 *Key Frames:* We successfully track two of the three loiterers (red and magenta bounding boxes). We track the ownership of the dropped luggage (green box) sufficiently well to avoid false alarms for stolen or abandoned luggage.

		Number of	Number of Missed Events				Max Temporal Error			
		Actual	Arsić	Ribeiro	Dalley	Ardö and	Arsić	Ribeiro	Dalley	Ardö and
Clip	Description	Events	*et al.* [3]	*et al.* [86]	*et al.* [23]	Åström [2]	*et al.* [3]	*et al.* [86]	*et al.* [23]	Åström [2]
S01	General Loitering 1	1	0	N/R	0	0	0.92	N/R	5.08 s	N/R
S02	General Loitering 2	1	0	N/R	0	0	3.56	N/R	1.44 s	N/R
S03	Swapping Bag 1	3	N/R	N/R	1	N/R	N/R	N/R	8.22 s	N/R
S04	Swapping Bag 2	3	N/R	N/R	2	N/R	N/R	N/R	0 s	N/R
S05	Theft 1	4	N/R	N/R	1	N/R	N/R	N/R	19.2 s	N/R
S06	Theft 2	1	N/R	N/R	1	N/R	N/R	N/R	∞	N/R
S07	Left Luggage 1	2	0	N/R	0	N/R	N/R	N/R	0.12 s	N/R
S08	Left Luggage 2	2	1	0	0	N/R	N/R	1	0.2 s	N/R

Table 6.3: *Comparative Summary of PETS 2007 Results:* Here we summarize the results from all PETS 2007 submissions, focusing on the detection rate and temporal errors. Ours was the only system for which results were reported on all clips. With the exception of *S01*, our temporal errors were also smaller than those of the others. In this table, "N/R" designates results that were "not reported."



(a) 0532 (b) 0759 (c) 0972

(d) 1136 (e) 2039 (f) 2107

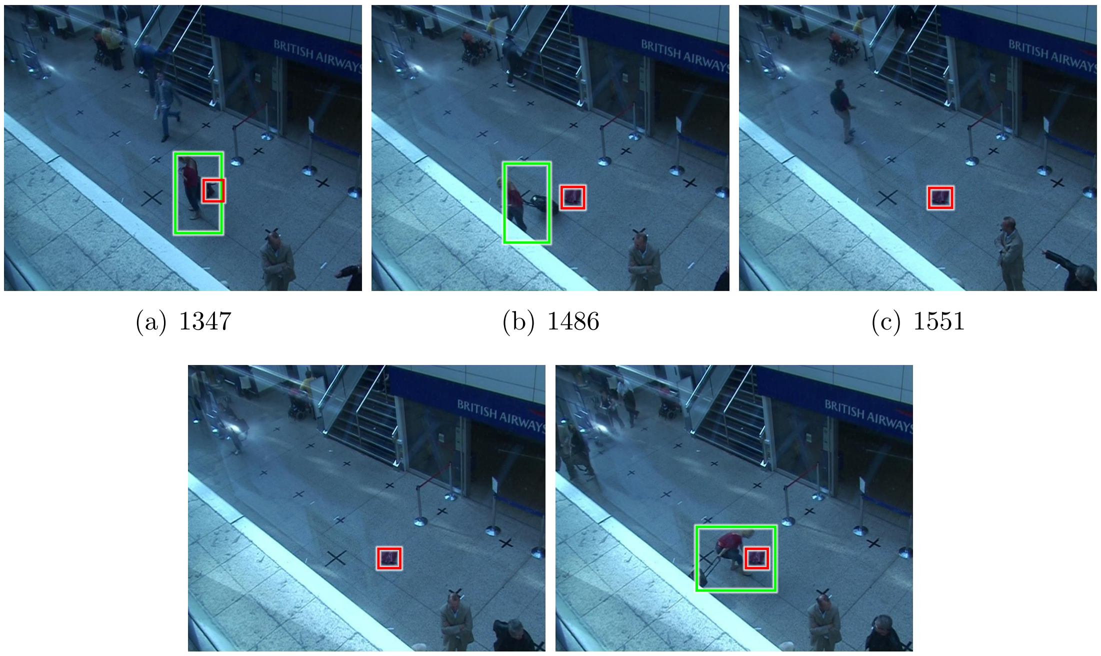
Figure 6-9: S05 *Key Frames:* We track the two scripted loiterers (green and magenta bounding boxes). We also track the ownership of the dropped luggage (red boxes) enabling us to detect the transfer of ownership to the thief.

were supplied with the dataset and the single "training" clip, *BACKGROUND*, had no events of interest, it was not possible to have a meaningful separation of training and test sets. Instead, we chose parameter settings as described in previous sections of this chapter. As noted in those sections, we used either standard parameter settings or when necessary, we used very loose thresholds based on minimal information. For example, at the end of §6.2, we describe an iterative mode finding algorithm. The large template radius was chosen by examining a single pixel value over time. After choosing this radius, we used it on all pixels and on all clips without further tuning. In all cases, we used the same parameter settings on all test videos.

We note that we had no false positive events for any of the clips.

For *S00*, there were no events that took place (and our system raised no alarms).

For *S01*, the first loitering clip, there was a single loitering event. Using our meanshift tracker, we were able to track the loiterer back to 5 seconds after the actual scene entry. Selected key frames are shown in Fig. 6-6. For *S02* (Fig. 6-7), the



(a) 1347 (b) 1486 (c) 1551

(d) 1922 (e) 2077

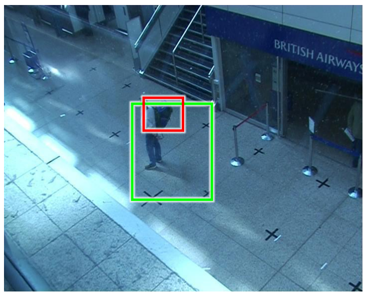
Figure 6-10: S07 *Key Frames:* We detect the loiterer (green bounding boxes), the fact that she drops and abandons a purse (red boxes), and that she is the person who later returns to pick up the purse.

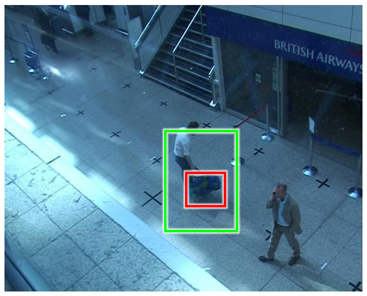
second loitering clip, we achieved excellent results.

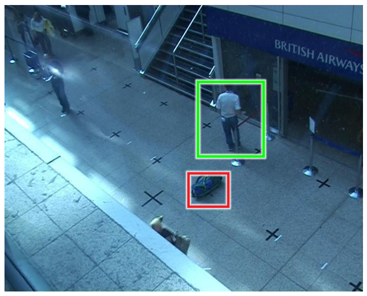
In each of *S03* (Fig. 6-8) and *S04*, a couple swapped a piece of luggage. Since the luggage was never unattended, according to the PETS definition, only loitering events occurred. For *S04*, we detected the time correctly, to the exact frame.

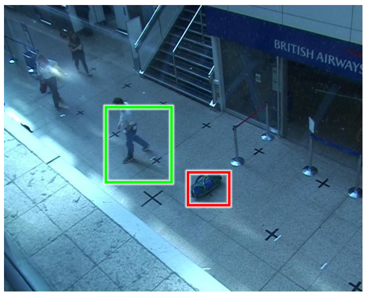
In each of *S05* and *S06*, a theft occurs. Because these are very busy scenes, especially while the event in question is occurring, we fail to track the second member of the victim couple in *S05* (Fig. 6-9) and we are not able to track the couple all the way back to their entrance time (yielding a high temporal error in the detected loitering event). For *S06* the scene is too busy for us to obtain a successful event detection.

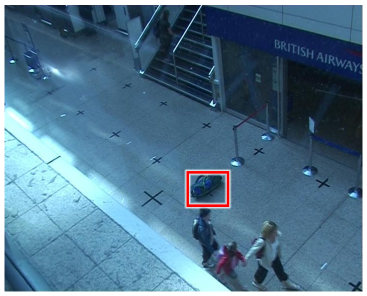
In *S07* (Fig. 6-10) and *S08* (Fig. 6-11), a person loiters in the scene, drops a bag, exits the scene, then later the same person reenters and picks up the luggage. Using the model trained for meanshift tracking, we are able to not only trigger the left-luggage alarms with high temporal accuracy, but we are also able to detect that

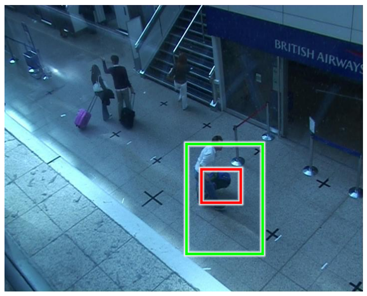

(a) 0543


(b) 0888


(c) 1050


(d) 1121


(e) 1214


(f) 1922

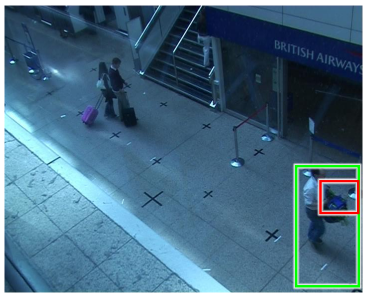

(g) 1972

Figure 6-11: S08 *Key Frames:* As in Fig. 6-10, we detect loitering, abandoned luggage, and reattended luggage events.

it is the same person picking up the luggage.

The loitering events we report in *S03*, *S04*, and *S05* were not in the official ground truth data, but our own manual verification indicates that they did occur.

In Tab. 6.3, we compare our results to others who worked on the PETS 2007 challenge problem. Ours was the only system that reported results on all test clips. With one exception, our temporal errors were significantly lower than those of the competition.

6.7 Summary

In this chapter, we have described a event detection system based on a bootstrapped background subtraction system. When presented with a novel event detection situation in a site monitoring application, it is valuable to build the tools and intuition such that new solutions can be deployed quickly. We use blob tracking as an attention mechanism and when we identify tracks of interest, we employ meanshift trackers to temporally extend tracks, find related tracks, and associate tracks that are temporally separated. Our system performs well on the PETS 2007 dataset and all experiments were performed in a short time period.

Chapter 7

Conclusions

In §1, we discussed a traditional visual tracking pipeline and applications of it such as activity modeling, event detection, and recognizing individual people. These are commonly desired components of automatic site monitoring systems. In order to achieve the high level goals of these applications, one needs low-level detectors and models that are robust enough to real world conditions like moving backgrounds, crowded scenes, and adverse lighting conditions to be useful. The low-level tools must also be computationally efficient enough that they can be used on large or even continuous datasets.

In §2, we showed a way of enhancing the traditional Mixture of Gaussians (MoG) background model to more effectively separate the moving and non-moving parts of a scene, even when there are small local motions arising from non-moving objects. That chapter takes advantage of the approach of enhancing an existing model (traditional MoG) when one can characterize how its modeling assumptions break down under real-world conditions (*i.e.* motions that are irregular in time do result in mixture components of sufficient strength in the traditional MoG).

The use of background subtraction as an object detection technique can fail under adverse lighting conditions and when a scene is densely filled with objects of interest. In §3, we described a novel implementation of a strong-model pedestrian detector that is efficient enough for use in large datasets with high-resolution video. It uses commodity graphics hardware to achieve up to a 76× speedup versus a CPU-only implementation running on a high-end workstation. This speed improvement was only possible by focusing the algorithm design on efficient memory access. As computational hardware becomes ever more parallel, the impact of memory bandwidth considerations on throughput is likely to become increasingly important to fast runtime.

In §4, we modeled scene-level activities with a Hierarchical Dirichlet Process (HDP) built on observed trajectories. Results are improved significantly when a strong model pedestrian detector is used instead of a weaker naïve feature point detector. In busy or cluttered scenes, the extra power of the strong model is needed to diminish the number of bad data associations.

Alternative strong appearance models were used in §5 to help improve the quality of pedestrian silhouettes for recognizing individual people. There we saw that we could overcome many systematic background subtraction errors. We did so by combining information from multiple models. More generic appearance information gathered from a large population of pedestrians can help remove artifacts that are persistent across a whole sequence of silhouettes. We also used a Hidden Markov Model (HMM) to combine information from multiple walking periods, reducing the effects of silhouette errors tied to particular parts of a scene through which a pedestrian travels.

In §6, we combine many different techniques to produce the most complete set known of results for a challenging event detection dataset.

If one were to build a new site monitoring system from scratch, we believe that a hybrid approach is necessary to obtain the best performance under real-world conditions. As in §6, a weak model like background subtraction can be used to obtain high-confidence information that can be used to bootstrap stronger models. With care, one can make these weaker models more robust by examining how they break down, as we saw in §2.

It would be interesting to further investigate the learning of stronger models from weak ones using an approach like that of Zhou and Tao [129]. Then given an efficient strong model (§3), a site monitoring system could learn the typical behavior of tracked actors (§4). As tracks or events of interest are detected (§6), stronger per-actor models could be built and used to further improve the system (§5, §6).

In the end, each layer of the system must be aware of the types of errors occurring in lower layers so they can be mitigated in the best possible way, either by switching lower-level layers or by incorporating the errors into the model. In §2, our enhanced background subtractor can be viewed as a thin additional layer built on top of a traditional MoG model that allows it to better handle temporally irregular dynamic textures. In §3 and §4, we saw that a strong pedestrian detector in place of background subtraction can be beneficial. In §6, we took a hybrid approach where we used background subtraction when it was robust and switched to a stronger color histogram model when background subtraction failed.

Appendix A

Notation and Conventions

In this thesis, we follow standard notation practices for mathematical derivations and analysis that are used in much of the machine learning and computer vision communities. In this chapter, we briefly outline these conventions.

The major semantic types of mathematical symbols are indicated by their font and various ancillary glyphs, as shown in Tab. A.1.

A.1 2×2 Confusion Matrices

We pose many of the problems in this thesis as binary decision problems: given some input $\boldsymbol{x}$, we wish to produce a binary labeling. For example, $\boldsymbol{x}$ might represent an pixel's observed color and we wish to determine whether the pixel should be classified as foreground or background. In other cases we may have a large set of feature vectors extracted from an image, $\mathbb{D} = \{\boldsymbol{x}_i\}_i$ and we wish to determine which subset corresponds to pedestrians shown in the image. A principled way to analyze the performance of an algorithm is to measure various statistics related to a 2×2 confusion matrix.

Consider an experiment involving N inputs. Each input has a binary ground

[1]Sometimes δ_{xy} or $\delta_{x,y}$ is used to denote the Kronecker delta function and the non-real function $\delta(x, y) = \lim_{a \to \infty} \frac{1}{a} \exp\bigl(-(x-y)^2/a^2\bigr)$ is reserved for the Dirac delta, its continuous counterpart. In this thesis, we will only be using the Kronecker delta. We use the $\delta(x, y)$ notation to improve the readability of subscripts and ancillary glyphs on the function arguments.

Example	Meaning
x	scalar value (standard font)
x	observed value of random variable X (lowercase)
X	random variable (uppercase)
X	constant, often indicating the cardinality of some set (uppercase)
$\hat{x}$	estimated value of x (often the maximum likely estimate; hat)
$\bar{x}$	mean value of x (bar)
$\breve{x}$	quantized value of x (breve)
$\tilde{x}$	normalized value of x (tilde)
$\mathrm{f}()$	function (Roman font)
$\boldsymbol{x}$	column vector of the form $(x_1, x_2, ..., x_N)^\top$ (bold)
$\mathbb{S}$	set of mathematical objects (typeface)
$\{x_i\}_i$	set of values enumerated by i
$\big\{x_i \mid x_i < 10\big\}$	the set of all x_i such that $x_i < 10$
$\lvert x \rvert$	absolute value, $\lvert x \rvert = \sqrt{x^2}$
$\lVert \boldsymbol{x} \rVert$	Euclidean or L_2 norm of vector $\boldsymbol{x}$, $\lVert \boldsymbol{x} \rVert = (x_1^2 + x_2^2 + ... + x_N^2)^{1/2}$
$\lVert \boldsymbol{x} \rVert_p$	Lebesgue p-norm of vector $\boldsymbol{x}$, $\lVert \boldsymbol{x} \rVert_p = (x_1^p + x_2^p + ... + x_N^p)^{1/p}$
$\lVert \Sigma \rVert$	determinant of the matrix Σ
$\delta(x, y)$	Kronecker[1]delta function, $\delta(x, y) = 1$ if and only if $x = y$, otherwise $\delta(x, y) = 0$. If the second argument is omitted, it is assumed to be 0: $\delta(x) = \delta(x, 0)$.

Table A.1: *Mathematical Notation:* We show our notational conventions by way of example. Note that we will often use the shorthand $p(x)$ instead of $p(X = x)$ when discussing probabilities of random variables.

truth label, y_i and the estimated label produced by our system, $\hat{y}_i$. Suppose that if $y_i = 1$, then input i corresponds to a pedestrian location in an image and $y_i = 0$ if it is not a pedestrian. $\hat{y}_i$ is then our system's estimate of whether location i contains a pedestrian or not. The successes and failures of the system can be characterized by the following 2×2 confusion matrix shown in Tab. A.2 where

- the true positive count is $TP = \sum_{i=1}^{N} \delta(y, 1)\,\delta(\hat{y}, 1)$,
- the false positive count is $FP = \sum_{i=1}^{N} \delta(y, 0)\,\delta(\hat{y}, 1)$,
- the false negative count is $FN = \sum_{i=1}^{N} \delta(y, 1)\,\delta(\hat{y}, 0)$,
- the true negative count is $TN = \sum_{i=1}^{N} \delta(y, 0)\,\delta(\hat{y}, 0)$,
- the positive count is $\hat{P} = \sum_{i=1}^{N} \delta(\hat{y}, 1)$, and
- the negative count is $N - \hat{P}$.

There are a number of commonly-used statistics that can be derived from such a matrix. The true positive rate

$$TPR = TP/(TP + FN) \tag{A.1}$$

measures the fraction of pedestrians that are detected by the system. The true positive rate is also known as the hit rate, sensitivity, or recall. The false positive rate

$$FPR = FP/(FP + TN) \tag{A.2}$$

measures the fraction of system detections that are not actually pedestrians. It is also known as the false rejection rate. The precision

$$PR = TP/(TP + FP) \tag{A.3}$$

measures the purity of the detections: what fraction of the system detections actually correspond to pedestrians. Precision is also known as the positive predictive value.

		actual value		
		$y_i = 1$	$y_i = 0$	total
predicted	$\hat{y}_i = 1$	True Positives	False Positives	$\hat{P}$
value	$\hat{y}_i = 0$	False Negatives	True Negatives	$N - \hat{P}$
	total	P	$N - P$	N

Table A.2: 2×2 Confusion Matrix

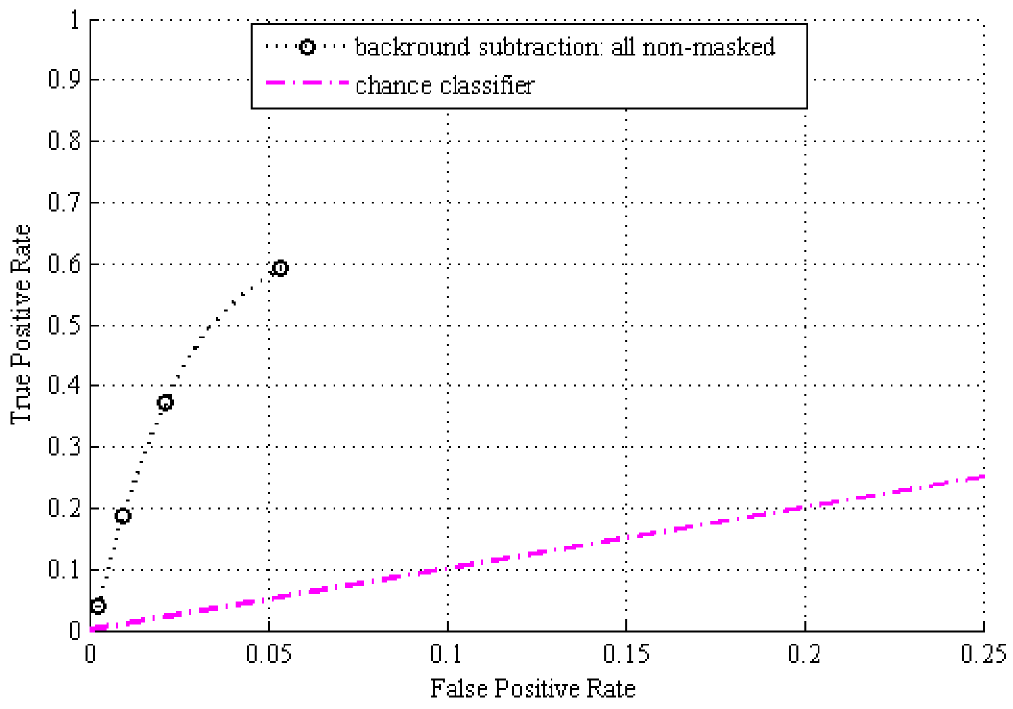
The miss rate

$$MR = 1 - TPR = FN/(TP + FN) \tag{A.4}$$

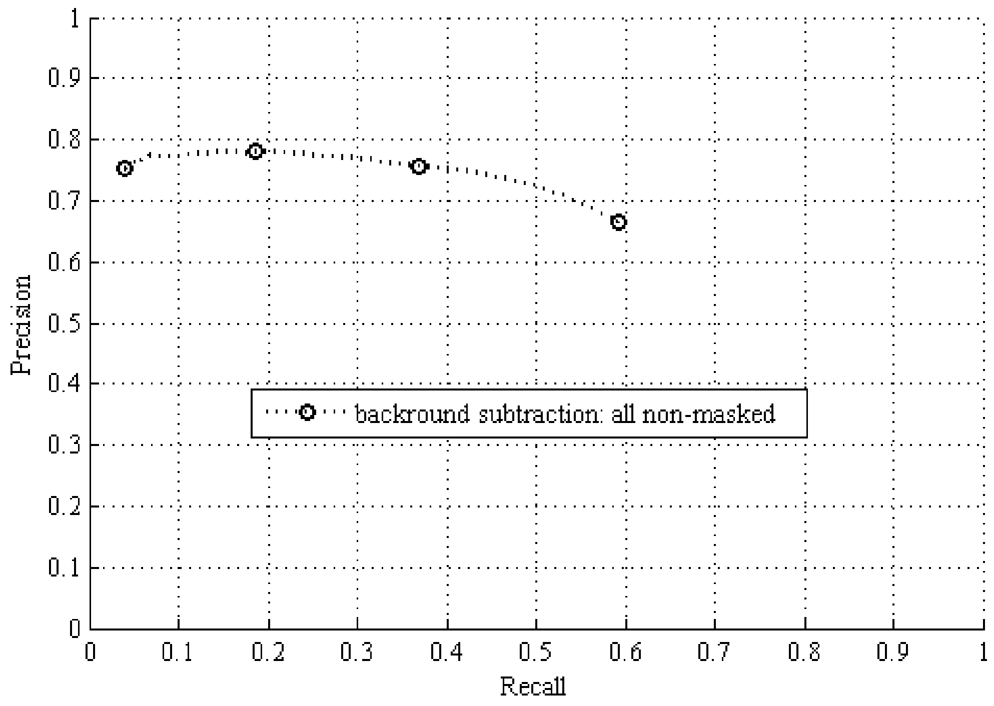
measures the fraction of pedestrians that are not detected by the system.

Most classifiers have tunable parameters that can be used to generate a variety of different confusion matrices. For example, a support vector machine [11] outputs real values which are then thresholded to produce a binary classification. By adjusting the threshold, the user can bias the classifier to produce more detections but with more false positives or to produce fewer detections but with more missed detections. One can produce a curve that characterizes the performance tradeoffs of a classifier by varying the threshold or other parameters and plotting the true positive versus false positive rates (Fig. A-1(a)). For historical reasons, this is known as a Receiver Operator Characteristic (ROC) curve. These curves are monotonic and a perfect classifier reaches the top-left corner of the plot.

There are situations where the negative class has infinite size and thus a ROC curve becomes degenerate. For example, suppose we wish to find all pedestrians in a given image by identifying a single bounding box for each person. If the bounding boxes can be positioned with arbitrary precision, there will be an infinite number of bounding boxes that do not correspond exactly to any of the fixed and finite number of pedestrians. In these cases, it is more appropriate to plot precision versus recall, as shown in Fig. A-1(b). Unlike ROC curves, precision-recall curves are not monotonic and the ideal classifier reaches the top-right corner. Davis and Goadrich [25] provide a more in-depth discussion of how to reason about ROC versus precision-recall curves and how to convert between the two of them under mild assumptions.



(a) ROC Curves



(b) Precision-Recall Curves

Figure A-1: *Sample ROC and Precision-Recall Curves:* See Fig. 3-8 in §3.2 for a fuller explanation of this plot.

A.2 Mathematical Objects

Here we list most of the mathematical objects used in this thesis.

For the sake of conciseness, superscripts and subscripts are often omitted from variable names when their values are unambiguous. For example, when discussing the Mixture of Gaussians (MoG) model where each pixel is modeled independently of all other pixels, we may abbreviate a Gaussian's mean as $\boldsymbol{\mu}_k$, where one should assume that this refers to $\boldsymbol{\mu}_{qk}^{(t)}$ where t is the current frame's time and q is the location of the pixel being discussed.

Name	Description
$\mathrm{BC}(p,q)$	Bhattacharyya distance between two probability mass functions, p and q
$b_p^{(t)}$	Number of MoG mixture components that are considered to belong to static portions of the scene for pixel location p at time t
(b_x, b_y)	(x,y) center position of descriptor block to which a HOG cell belongs
$\mathrm{C}(\cdot,\cdot)$	The cost of associating two detections with each other
$\mathbb{C}$	The set of all (x,y) pixel locations within a HOG cell
$\boldsymbol{c}_p^{(t)}$	Color value observed at time t for pixel location p
$\bar{\boldsymbol{c}}^{(t)}$	Mean color of the whole image at time t
$\tilde{\boldsymbol{c}}_p^{(t)}$	Illumination-normalized color at time t for pixel location p
$\tilde{\tilde{\boldsymbol{c}}}_p^{(t)}$	Color of pixel p at time t after illumination normalization and matching to the background model
$corr_{qk}^{(t)}$	Un-normalized Color correlation statistics for MoG mixture component k at location q and time t, modulated by an exponential decay
$\mathbb{D}_{gt}$	The set of ground truth pedestrian windows, $\mathbb{D}_{gt} = \left\{\boldsymbol{x}_j = (i_j, x_j, y_j, w_j, h_j)^\top\right\}_{j=1}^{P}$
$\mathbb{D}^{(t)}$	The set of object detections at time t

Name	Description
$\mathrm{D}_p(l_p; \boldsymbol{c}_p)$	MRF data energy for measuring the incompatibility between label l_p at pixel location p and the observed color, $\boldsymbol{c}_p$
$\mathrm{d}(\boldsymbol{c}; \boldsymbol{\mu}, \Sigma)$	Mahalanobis distance function
d_{pqk}	Shorthand for the Mahalanobis distance $\mathrm{d}(\boldsymbol{c}_p; \boldsymbol{\mu}_{qk}, \Sigma_{qk})$
$(d_x, d_y)^\top$	Maximum (x, y) displacement of a pixel within a dynamic texture
$d_i^{(t)}$	Object detection i at time t
$e_{qk}^{(t)}$	Total evidentiary weight assigned to MoG mixture component k at location q and time t, modulated by an exponential decay
f	Average number of frames in a given pedestrian's walking cycle
$\mathrm{f}(\boldsymbol{x})$	Support vector machine (SVM) classification function for feature vector $\boldsymbol{x}$
$\mathrm{H}(\lambda)$	The base Dirichlet distribution, a pseudo-count prior for multinomials (user supplied prior)
$\mathrm{h}(o; b_x, b_y, \mathbb{C})$	A histogram of oriented gradients (HOG) for a single cell
h_j	height pedestrian detection j's bounding box
I	Number of input test images
I_j	Number of quantized observations in trajectory j (data dependent scalar)
$I(x, y)$	Image whose pixels are indexed by (x, y)
$I_\gamma(x, y)$	Gamma-corrected image
i	Object detection index
i_j	Image index for pedestrian detection j
J	Number of observed trajectories (data dependent scalar)
j	Meanshift iteration index
j	Object detection index
j	Trajectory index
K_q	Total number of MoG mixture components at pixel location q
k	MoG mixture component index

Name	Description
L	Foreground/background label, used in formal contexts as a random variable
$l_p^{(t)}$	Foreground (1) / background (0) label for pixel location p at time t
l_i	Pedestrian (+1) / non-pedestrian (-1) label for detection i
N	Dimensionality of the color vector $\boldsymbol{c}$ (typically 3)
N	Ground truth number of feature vectors that do not correspond to pedestrians extracted from a given input image
N	Number of HMM states
$\mathbb{N}_p$	Set of 8-connected neighbors of pixel location p where $q \in \mathbb{N}$ if and only if p and q are within one horizontal, vertical, or diagonal pixel location of each other
$\mathcal{N}(\cdot;\cdot,\cdot)$	Multivariate normal (Gaussian) probability distribution function
P	Number of actual pedestrians in a given video frame
$\hat{P}$	Number of detected pedestrians in a given video frame
$\mathbb{P}$	The set of all possible pixel locations in an image
p	Indexes pixel locations. For conciseness, this index is often omitted from variables, *e.g.* variables like $\boldsymbol{\mu}_p^{(t)}$ are often abbreviated as $\boldsymbol{\mu}^{(t)}$ when the pixel location is implicitly known
$p(\boldsymbol{c})$	Probability of observing color $\boldsymbol{c}$ under the color histogram p
q	Indexes pixel locations.
$q(\boldsymbol{c})$	Probability of observing color $\boldsymbol{c}$ under the color histogram q
S_N	The silhouettes provided with the NIST gait data, generated by USF using a semi-automatic method
S_r	Our fully-automatic raw silhouettes (see §5.5)
S_{d3}	S_r dilated with a neighborhood size of 3
S_{d6}	S_r dilated with a neighborhood size of 6
S_p	S_r cleaned and filled using the population model

Name	Description
S_{Hr}	S_r filled using an HMM trained using S_r
S_{Hp}	S_p filled using an HMM trained using S_p
$\hat{s}_i$	Classification score for detection i
$s_{qk}^{(t)}$	The sample sum of observed color values assigned to to MoG mixture component k at location q and time t, modulated by an exponential decay and $\{\rho_{pqk}^{(t)}\}_{p,t}$
s_t	HMM state at time t
$\mathrm{T}(i,j) \in \{0,1\}$	A match graph where a nonzero entry indicates that detection i in the first set has been matched to detection j in the second set of detections
$\mathrm{T}_p\left(l_p^{(t)}, l_p^{(t-1)}\right)$	MRF temporal energy for measuring the incompatibility between label $l_p^{(t)}$ at pixel location p and time t to the label at the same location but at time $t-1$. This is often abbreviated as $\mathrm{T}_p(l_p)$
t	Time index; in this thesis, time is discrete except when doing Kalman tracking
t_F	Foreground label energy used in $\mathrm{D}_p(l_p; \boldsymbol{c}_p)$
t_N	Neighbor label mismatch energy used in $\mathrm{V}_{p,q}(l_p, l_q)$ when $l_p \neq l_q$
t_T	Temporal label mismatch energy used in $\mathrm{T}_p\left(l_p^{(t)}, l_p^{(t-1)}\right)$ when $l_p^{(t)} \neq l_p^{(t-1)}$
$\mathrm{V}_{p,q}(l_p, l_q)$	MRF neighborhood energy for measuring the incompatibility between label l_p at pixel location p and l_q at q
W	Spatial search window for our enhanced MoG background model ($W = \|\mathbb{N}_p\|$)
$\mathrm{w}(x,y)$	Block-sized voting stencil used in computing HOG block features that emphasizes gradients near the center of the block
$\boldsymbol{w}$	Normal vector for the separating hyperplane of a support vector machine (SVM)
w_0	Distance from the origin of an SVM's separating hyperplane

Name	Description
w_j	width pedestrian detection j's bounding box
w_{ji}	A single observed position and motion direction of trajectory j, quantized, indexed by i ($w_{ji} \sim \boldsymbol{\phi}_c \vert c = z_{ji}$)
$\breve{x}$	Quantized position
$\boldsymbol{x}_i$	Image locations where there are positive detection scores
x_j	x-coordinate of pedestrian detection j's centroid
$\boldsymbol{y}^{(j)}$	Meanshift kernel position at iteration j
y_j	y-coordinate of pedestrian detection j's centroid
$\boldsymbol{y}_t$	HMM observation vector at time t
z_{ji}	Index of the cluster chosen for observation i of trajectory j ($z_{ji} \sim \boldsymbol{\pi}_j$)
z_p	Index of the mixture component selected to generate the color observed at pixel location p. For our model in §2, $z_p = (q, k)$ is a two-dimensional index that includes the matched pixel location q and its mixture component index k
α	Base MoG learning rate (reciprocal of an observation's halflife)
α	Bias for how much each $\boldsymbol{\pi}_j$ should resemble $\boldsymbol{\beta}$ (user supplied scalar)
$\boldsymbol{\beta}$	Frequency of each activity cluster, across the whole dataset ($\boldsymbol{\beta} \sim \mathrm{GEM}(\gamma)$)
γ	Bias toward concentrating $\boldsymbol{\beta}$'s mass onto a few activity clusters (user supplied scalar)
θ_p	Bernoulli prior on pixel p being observed as foreground in a pedestrian silhouette
λ	Bias for how much $\boldsymbol{\phi}_c$'s samples should resemble the base distribution (user supplied scalar)
$\boldsymbol{\mu}_{qk}^{(t)}$	Mean statistic for Gaussian mixture component k at time t at pixel location q

Name	Description
$\boldsymbol{\pi}_j$	Frequency of each activity cluster in trajectory j ($\boldsymbol{\pi}_j \sim \mathrm{DP}(\alpha, \boldsymbol{\beta})$)
ρ	Actual data-dependent MoG learning rate
Σ	A generic covariance matrix, used in many contexts
$\Sigma_{img}^{(t)}$	Color covariance matrix for the whole image at time t
$\Sigma_{qk}^{(t)}$	Covariance matrix statistics for Gaussian mixture component k at time t at pixel location q
τ_{bgfrac}	Fraction of the MoG mixing weight that is assumed to belong to static objects, for each pixel
τ_{det}	Minimum amount of overlap required between a pedestrian detection and a corresponding ground truth bounding box for the two to be considered a match
τ_{ds}	Minimum amount of bounding box motion required to classify a dropped object as moving again
τ_{match}	Maximum squared Mahalanobis distance allowed between an observed pixel value and a matching MoG mixture component
$\tau_{meanshift}$	Maximum Bhattacharyya distance to tolerate when performing meanshift tracking
$\tau_{reacquire}$	Maximum Bhattacharyya distance to tolerate when matching objects that have recently left a scene to new entrants
Φ	The set of MoG parameters, $\Phi = \{\omega_{qk}, \mu_{qk}, \Sigma_{qk}\}_{q,k}$
$\boldsymbol{\phi}_c$	Activity cluster: frequency of each quantized observation in cluster c ($\boldsymbol{\phi}_c \sim \mathrm{H}(\lambda)$)
Ω	Number of orientation bins in each cell of a HOG block feature
$\omega_i^{(t)}$	Mixing weight for Gaussian mixture component i at time t for the pixel of interest

Bibliography

[1] Richard Anderson and João C. Setubal. A parallel implementation of the push-relabel algorithm for the maximum flow problem. *Journal of Parallel and Distributed Computing*, 29(1):17–26, 1995. 28

[2] Håkan Ardö and Kalle Åström. Multi sensor loitering detection using online viterbi. In *PETS Workshop*, pages 87–94. IEEE, June 2007. 198

[3] Dejan Arsić, Martin Hofmann, Björn Schuller, and Gerhard Rigoll. Multi-camera person tracking and left luggage detection applying homographic transformation. In *PETS*, pages 55–62. IEEE, oct 2006. 44, 198

[4] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002. 41

[5] Edouard Auvinet, Etienne Grossmann, Caroline Rougier, Mohamed Dahmane, and Jean Meunier. Left-luggage detection using homographies and simple heuristics. In *PETS Workshop*, pages 51–58. IEEE, 2006. 44, 180

[6] A. Baumberg and D. Hogg. Learning flexible models from image sequences. In *Proc of ECCV*, pages 299–308, 1994. 158

[7] M. Bicak. MaxxPI^2 the system bench: Top 10 FLOPS, Apr 2009. `http://www.maxxpi.net/pages/result-browser/top10---flops.php`. 120

[8] Biswajit Bose, Xiaogang Wang, and Eric Grimson. Multi-class object tracking algorithm that handles fragmentation and grouping. In *CVPR*. IEEE, 2007 2007. 192

[9] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9), September 2004. 28, 189

[10] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *Pattern Analysis and Machine Intelligence*, volume 23, pages 1222–1239. IEEE, November 2001. 28, 189

[11] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998. 96, 210

[12] "visual surveillance and monitoring". World Wide Web, 2000. 91

[13] Robert Collins, Alan Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, and Osamu Hasegawa. A system for video surveillance and monitoring. Technical Report CMU-RI-TR-00-12, Robitics Institute, Carnegie Mellon University, may 2000. 19

[14] D. Comaniciu, V. Ramesh, and Peter Meer. The variable bandwidth mean shift and data-driven scale selection. In *International Conference on Computer Vision*, volume 1, pages 438–445. IEEE, 2001. 180, 192

[15] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *PAMI*, 24(5):603–619, May 2002. 108, 115

[16] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1997. 28, 40

[17] NVIDIA Corporation, editor. *NVIDIA CUDA^{TM}Programming Guide*. NVIDIA, 2.2 edition, April 2009. 122

[18] Naresh P. Cuntoor, B. Yegnanarayana, and Rama Chellappa. Activity modeling using event probability sequences. *Image Processing, IEEE Transactions on*, 17(4):594–607, April 2008. 43

[19] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition*, volume II, pages 886–893. IEEE, June 2005. 35, 37, 50, 75, 92, 98, 108

[20] Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. In *European Conference on Computer Vision*. IEEE, May 2006. 37

[21] Gerald Dalley and Tomáš Ižo. Schematic querying of large tracking databases. Technical Report MIT-CSAIL-TR-2006-043, MIT CSAIL, Jun 2006. 43

[22] Gerald Dalley, Josh Migdal, and W. Eric L. Grimson. Background subtraction for temporally irregular dynamic textures. In *Workshop on Applications of Computer Vision*, January 2008. 53

[23] Gerald Dalley, Xiaogang Wang, and W. Eric L. Grimson. Event detection using an attention-based tracker. In *PETS Workshop*, pages 71–78. IEEE, June 2007. 6, 45, 179, 198

[24] Trevor Darrell and A. Pentland. Robust estimation of a multi-layer motion representation. In *Workshop on Visual Motion*. IEEE, 1991. 31

[25] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *ICML*, 2006. 84, 115, 210

[26] Andrew Delong and Yuri Boykov. A scalable graph-cut algorithm for n-d grids. In *CVPR*. IEEE, June 2008. 28

[27] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*. IEEE, 2008. 135

[28] Robert Fergus, Pietro Perona, and Andrew Zisserman. Object class recognition by unsupervised scale-invariant learning. *CVPR*, 2:264, 2003. 39

[29] James Ferryman and James L. Crowley, editors. *PETS*. IEEE, jun 2006. 43, 91, 180

[30] James Ferryman and James L. Crowley, editors. *PETS*. IEEE, jun 2007. 43, 91

[31] Martin A. Fischler and Robert A. Elschlager. The representation and matching of pictorial structures. *Computers*, C-22(1):67–92, Jan 1973. 38

[32] Raymund Flandez. Stop that thief. The Wall Street Journal, 16 June 2008. 17

[33] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003. 19

[34] Alexandre R.J. François, Ram Nevatia, , Jerry Hobbs, Robert C. Bolles, and John R. Smith. Verl: An ontology framework for representing and annotating video events. *Multimedia*, 12(4):76–86, 2005. 43

[35] D. M. Gavrila. The visual analysis of human movement: A survey. *CVIU*, 73(1):82–98, mar 1999. 36

[36] D. M. Gavrila. Pedestrian detection from a moving vehicle. In *ECCV*, pages 37–49, 2000. 36, 91, 158

[37] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd edition, 2004. 21

[38] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, pages 564–584, 1987. 188

[39] National Geographic. Inside grand central. DVD, 2005. 156

[40] Nagia Ghanem, Daniel DeMenthon, David Doermann, and Larry Davis. Representation and recognition of events in surveillance video using petri nets. In *Workshop on Event Mining*, page 112, 2004. 43

[41] H. Grabner, P. M. Roth, M. Grabner, and H. Bischof. Autonomous learning of a robust background model for change detection. In *PETS Workshop*, pages 39–46. IEEE, 2006. 180

[42] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Efficient leraning with sets of features. *JMLR*, pages 725–760, apr 2007. 39

[43] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistics Society*, 1989. 28

[44] Ismail Haritaoglu, David Harwood, and Larry S. Davis. W4: Real-time surveillance of people and their activitis. *Pattern Analysis and Machine Intelligence*, 22(8):809–803, August 2000. 156, 158, 179

[45] Thomas Hofmann. Probabilistic latent semantic indexing. In *Uncertainty in Artificial Intelligence*, 1999. 39

[46] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Putting objects in perspective. In *CVPR*. IEEE, 2006. 80

[47] B.K.P. Horn. *Robot Vision*. MIT Press, 1986. 31

[48] Ronald Hughes, Herman Huang, Charles Zegeer, and Michael Cynecki. Evaluation of automated pedestrian detection at signalized intersections. Technical Report FHWA-RD-00-097, Highway Safety Research Center, U.S. Department of Transportation, Aug. 2001. 91

[49] Yuri A. Ivanov and Aaron F. Bobick. Recognition of visual activities and interactions by stochastic parsing. *PAMI*, 22(8):852–872, 2000. 42

[50] Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory, and Algorithms*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Apr. 2002. 96

[51] A. Johnson and A. Bobick. Gait recognition using static, activity-specific parameters. In *Proc of CVPR*, 2001. 156

[52] Nebojsa Jojic and Brendan Frey. Learning flexible sprites in video layers. In *CVPR*. IEEE, 2001. 56, 185

[53] Seong-Wook Joo and Rama Chellappa. Recognition of multi-object events using attribute grammars. In *ICIP*, pages 2897–2900, oct 2006. 43

[54] A. Kale, A.N. Rajagopalan, and V. Kruger N. Cuntoor. Gait-based recognition of humans using continuous hmms. In *5th International Conference on Automatic Face and Gesture Recognition*, May 2002. 158

[55] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. 41

[56] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *PAMI*, May 2002. 189

[57] N. Krahnstoever, P. Tu, T. Sebastian, A. Perera, and R. Collins. Multi-view detection and tracking of travelers and luggage in mass transit environments. In *PETS Workshop*, pages 67–74. IEEE, 2006. 180

[58] M. Pawan Kumar, P. H. S. Torr, and Andrew Zisserman. Learning layered motion segmentations of video. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2005. 32, 33

[59] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discorse Processes*, pages 259–284, 19998. 39

[60] Aldo Laurentini. The visual hull concept for silhouette-based image understanding. *PAMI*, 16(2):150–162, feb 1994. 44

[61] L. Lee and W.E.L. Grimson. Gait aanalysis for recognition and classification. In *Proc of Face and Gesture*, 2002. 168

[62] Lily Lee, Gerald Dalley, and Kinh Tieu. Learning pedestrian models for silhouette refinement. In *International Conference on Computer Vision*, pages 663–670, Oct. 2003. 155

[63] Bastian Leibe, Edgar Seemann, and Bernt Schiele. Pedestrian detection in crowded scenes. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:878–885, June 2005. 36, 134

[64] T.K. Leung, M.C. Burl, and P. Perona. Probabilistic affine invariants for recognition. In *CVPR*, pages 678–684, Jun 1998. 38

[65] L. Li, R. Luo, R. Ma, W. Huang, and K. Leman. Evaluation of an ivs system for abandoned object detection on pets 2006 datasets. In *PETS Workshop*, pages 91–98. IEEE, 2006. 180, 185

[66] David Lowe. Object recognition from local scale-invariant features. In *ICCV*, volume 2, pages 1150–1157 vol.2, 1999. 37

[67] F. Lv, X. Song, B. Wu, V. K. Singh, and R. Nevatia. Left luggage detection using bayesian inference. In *PETS Workshop*, pages 83–90. IEEE, 2006. 44, 180

[68] J. Martínez-del-Rincón, J. E. Herrero-Jaraba, J. R. Gómez, and C. Orrite-Uruñuela. Automatic left luggage detection and tracking using multi-camera ukf. In *PETS Workshop*, pages 59–66. IEEE, 2006. 44, 180

[69] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-based visual hulls. In *SIGGRAPH Computer Graphics Proceedings*, pages 369–374, 2000. 156

[70] Joshua Migdal. Robust motion segmentation using Markov thresholds. Master's thesis, MIT, September 2003. 27, 28, 60, 189

[71] Joshua Migdal and W. Eric L. Grimson. Background subtraction using markov thresholds. In *IEEE Workshop on Motion and Video Computing*, January 2005. 27, 29

[72] Anurag Mittal and Nikos Paragios. Motion-based background subtraction using adaptive kernel density estimation. In *CVPR*. IEEE, 2004. 29, 54, 55, 59, 64, 68, 69, 70, 71, 184

[73] Woonhyun Nam and Joonhee Han. Motion-based background modeling for foreground segmentation. In *Proceedings of the 4^{th} ACM International Workshop on Video Surveillance and Sensor Networks*, pages 35—44. ACM, September 2006. 56

[74] Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. *ICPR*, 3:850–855, 2006. 133

[75] Hubert Nguyen, editor. *GPU Gems 3*. Addison-Wesley Professional, 2007. 122

[76] Somboom Ongeng, Ram Nevatia, and François Bremond. Video-based event recognition: Activity representation and probabilistic recognition methods. *CVIU*, pages 129–162, 2004. 43

[77] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physics Review*, 65(3-4):117–149, Feb 1944. 28

[78] Opencv: Open computer vision library. World Wide Web, 2008. 88, 142

[79] Peter Orbanz and Joachim M. Buhmann. Nonparametric bayesian image segmentation. Technical Report 496, Department of Computer Science, ETH Zürich, October 2005. 137, 140

[80] M. Oren, C. Papageorgio, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *Proc of CVPR*, 1997. 158

[81] P. Jonathon Phillips, Patrick Grother, Sudeep Sarkar, Isidro Robledo, and Kevin Bowyer. Baseline results for the challenge problem of human id using gait analysis. In *5th International Conference on Automatic Face and Gesture Recognition*, May 2002. 49, 157, 165, 171, 172

[82] P. Jonathon Phillips, Sudeep Sarkar, Isidro Robledo, Patrick Grother, and Kevin Bowyer. The gait identification challenge problem: Data sets and baseline algorithm. *ICPR*, 1, 2002. 48

[83] Robert Pless, John Larson, Scott Siebers, and Ben Westover. Evaluation of local models of dynamic backgrounds. In *CVPR*, pages 73–78, 2003. 59

[84] Fatih Porikli and Jay Thorton. Shadow flow: A recursive method to learn moving cast shadows. In *ICCV*, 2005. 61

[85] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*, chapter 6, pages 321–389. Prentice Hall, 1993. 163, 168

[86] Pedro Canotilho Ribeiro, Plinio Moreno, and José Santos-Victor. Detecting luggage related behaviors using a new temporal boost algorithm. In *PETS Workshop*, pages 63–69. IEEE, June 2007. 198

[87] Peter Sand and Seth Teller. Particle video: Long-range motion estimation using point trajectories. In *Computer Vision and Pattern Recognition*. IEEE, 2006. 33

[88] S. Sarkar, P.J. Phillips, Z. Liu, I.R. Vega, P. Grother, and K.W. Bowyer. The humanid gait challenge problem: data sets, performance, and analysis. *PAMI*, 27(2):162–177, feb 2005. 48

[89] Sudeep Sarkar and Zongyi Liu. *Handbook of Biometrics*, chapter Recognition from Gait, pages 110–129. Springer Verlag, 2008. 156, 158

[90] M. Scott, M. Niranjan, and R. Prager. Realisable classifiers: improving operating performance on variable cost problems, 1998. 66, 84

[91] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter sensitive hashing. In *ICCV*. IEEE, 2003. 36

[92] Linda Shapiro and G. Stockman. *Computer Vision*. Prentice Hall, 2002. 28

[93] Yaser Sheikh and Mubarak Shah. Bayesian object detection in dynamic scenes. In *CVPR*. IEEE, 2005. 29, 55, 57, 59, 68, 71

[94] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR*, pages 593–600, jun 1994. 88, 142

[95] P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell. Face recognition by humans: Nineteen results all computer vision researchers should know about. *Proceedings of the IEEE*, 94(11):1948–1962, Nov. 2006. 48

[96] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering object categories in image collections. In *ICCV*, 2005. 39

[97] K. Smith, P. Quelhas, and D. Gatica-Perez. Detecting abandoned luggage items in a public space. In *PETS Workshop*, pages 75–82. IEEE, 2006. 180, 192

[98] Juan Soulie. C++ language tutorial: Arrays. World Wide Web, April 2009. 139

[99] Chris Stauffer. Automatic hierarchical classification using time-based co-occurrences. In *CVPR*, pages 333–339. IEEE, jun 1999. 44

[100] Chris Stauffer. *Perceptual Data Mining: Bootstrapping Visual Intelligence from Tracking Behavior*. Phd, Massachusetts Institute of Technology, May 2002. 19

[101] Chris Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. In *CVPR*, pages 246–252. IEEE, 1999. 22, 29, 53, 60, 80, 179, 184

[102] Erik Sudderth. *Graphical Models for Visual Object Recognition and Tracking*. Ph.d., Massachusetts Institute of Technology, Cambridge, MA, 2006. 137

[103] Avarind Sundaresan, Amit Roy Chowdhury, and Rama Chellapa. A hidden markov model based framework for recognition of humans from gait sequences. In *ICIP*, 2003. 48, 49, 158

[104] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), Mar 2005. 120

[105] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, December 2006. 39, 137, 140, 142

[106] M. Everingham *et al.* The 2005 pascal visual object classes challenge. In *PASCAL Challenges Workshop*. Springer-Verlag, 2006. 99, 111

[107] Kentaro Toyama, John Krumm, Barry Brumitt, and Brian Meyers. Wallflower: Principles and practice of background maintenance. *International Conference on Computer Vision*, 1:255, 1999. 29, 66, 68, 69

[108] Ken Turkowski. *Graphics Gems I*, pages 147–165. Academic Press, 1990. 123

[109] Vibhav Vineet and P.J. Narayanan. Cuda cuts: Fast graph cuts on the GPU. In *Workshop on Computer Vision on GPUs*, June 2008. 28

[110] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001. 37

[111] Van-Thinh Vu, François Bremond, and Monique Thonnat. Temporal constraints for video interpretation. In *ECAI*, 2002. 42

[112] Hanzi Wang and David Suter. A re-evaluation of mixture-of-gaussian background modeling. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1017–1020. IEEE, March 2005. 29

[113] Hanzi Wang and David Suter. Background subtraction based on a robust consensus method. In *International Conference on Pattern Recognition*, volume 1, pages 223–226. IEEE, August 2006. 29

[114] John Wang and Edward Adelson. Representing moving images with layers. In *Transactions on Image Processing*, pages 625–638. IEEE, September 1994. 31, 56

[115] Xiaogang Wang, Keng Teck Ma, Gee-Wah Ng, and W. Eric L. Grimson. Trajectory analysis and semantic region modeling using a nonparametric bayesian model. In *CVPR*. IEEE, 2008. 47, 50, 137, 139

[116] Xiaogang Wang, Xiaoxu Ma, and W. Eric L. Grimson. Unsupervised activity perception by hierarchical bayesian models. In *CVPR*. IEEE, 2007. 45

[117] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. *CVPR*, 2:2101, 2000. 39

[118] Greg Welch and Gary Bishop. An introduction to the kalman filter. TR 95-041, University of North Carolina at Chapel Hill, 2004. 41

[119] Christopher K. I. Williams and Michalis K. Titsias. Greedy learning of multiple ojbects in images using robust statistics and factorial learning. In *Neural Computation*, volume 16, pages 1039–1062. MIT Press, May 2004. 185

[120] John Winn and Andrew Blake. Generative affine localisation and tracking. In *Advances in Neural Information Processing Systems*, volume 17, pages 1505–1512, 2004. 32, 56

[121] John Winn and Andrew Blake. Variational message passing. *Journal of Machine Learning Research*, 6:661–694, 2005. 32

[122] Christian Wojek, Gyuri Dorkó, André Schulz, and Bernt Schiele. Sliding-windows for rapid object class localization: A parallel technique. In *Lecture Notes in Computer Science*, volume 5096/2008, pages 71–81. Springer Berlin / Heidelberg, 2008. 37, 50, 121, 122, 128

[123] Li Zhang and Ram Nevatia. Efficient scan-window based object detection using GPGPU. In *CVGPU Workshop at CVPR*. IEEE, 2008. 128

[124] Tao Zhao and Ram Nevatia. Tracking multiple humans in complex situations. *Pattern Analysis and Machine Intelligence*, 29(9):1208–1221, September 2004. 42

[125] Tao Zhao, Ram Nevatia, and Bo Wu. Segmentation and tracking of multiple humans in crowded environments. *PAMI*, 30(7):1198–1211, July 2008. 41

[126] W. Zhao, Rama Chellapa, Ariel Rosenfield, and P.J. Phillips. Face recognition: A literature survey. *ACM Computing Surveys*, pages 399–458, 2003. 48

[127] J. Zhong and Stan Sclaroff. Segmenting foreground objects from a dynamic textured background via a robust kalman filter. In *ICCV*. IEEE, 2003. 56, 68, 71

[128] S. Zhou and R. Chellappa. Probabilistic human recognition from video. In *Proc of ECCV*, pages 681–697, 2002. 158

[129] Yue Zhou and Hai Tao. A background layer model for object tracking through occlusion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1079–1085. IEEE, 2003. 32, 185, 204

[130] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:1491–1498, 2006. 37, 134