ABSTRACT

Title of Document: TOOLS FOR ADVANCED VIDEO METADATA MODELING

Sameer Avinash Kibey, Master of Science, 2007

Directed By: Dr. David Doermann
University of Maryland Institute for Advanced Computer Studies (UMIACS)

In this Thesis, we focus on problems in surveillance video analysis and propose advanced metadata modeling techniques to address them. First, we explore the problem of constructing a snapshot summary of people in a video sequence. We propose an algorithm based on the eigen-analysis of faces and present an evaluation of the method. Second, we present an algorithm to learn occlusion points in a scene using long observations of moving objects, provide an implementation and evaluate its performance. Third, to address the problem of availability and storage of surveillance videos, we propose a novel methodology to simulate video metadata. The technique is completely automated and can generate metadata for any scenario with minimal user interaction. Finally, a threat detection model using activity analysis and trajectory data of moving objects is proposed and implemented. The collection of tools presented in this Thesis provides a basis for higher level video analysis algorithms.

TOOLS FOR ADVANCED VIDEO METADATA MODELING

By

Sameer Avinash Kibey

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2007

Advisory Committee:
Associate Professor Carol Y. Espy-Wilson (Chair/Advisor)
Dr. David Doermann (Co-Chair)
Associate Professor Min Wu

Acknowledgements

During the Masters program at the University of Maryland, I have been fortunate to interact with several people who have influenced me greatly. I would like to take this opportunity to thank everyone who has helped me in successful completion of my Thesis and the graduate studies at the University.

Firstly I would like to thank Dr. David Doermann for his guidance and support in my research at the *Language and Media Processing Laboratory*. He has greatly inspired the direction of my research by his constructive inputs. I cannot thank him enough for his supervision and advice at every stage of my work. Dr. Doermann's reviews and insightful comments have helped me immensely in writing this Thesis.

I wish to thank my academic advisor Professor Carol Espy-Wilson for her encouragement and timely help through the entire Masters program. I am also grateful to the Thesis committee including Professor Espy-Wilson, Dr. Doermann and Professor Min Wu for agreeing to be on the Committee and taking out the time to review my Thesis.

I would like to thank the sponsor of this research - *Panasonic System Solutions Development Center* (PSDU) at Princeton, NJ. Part of the work presented in this Thesis are results and extension of results of the collaborative work with Dr. K. C. Lee, Dr. Hasan Timucin Ozdemir, and Dr. Lipin Liu during summer internship at

Panasonic. I am sincerely thankful to Dr. Lee, Dr. Hasan and Dr. Liu for providing me with opportunity to work on very interesting problems in visual surveillance. PSDU has a very intellectually stimulating environment and I learnt a great deal through the internship. I am thankful to Panasonic for allowing me to include some of the ideas developed over the internship at PSDU in the Thesis. The support of this research by Panasonic is gratefully acknowledged.

I would also like to thank my colleagues at the ECE Department and the *Language and Media Processing Laboratory* –Yang Yu, Mudit Agrawal, Zhe Lin, Burcu Karagol-Ayan, Xiaodong Yu, Yi Li, May Huang, Xu Liu, Ming Luo, Huanfeng Ma, Tandeep Sidhu, David Mihalcik, Daniel Ramsbrock, Yefeng Zheng, Jian Liang and Dr. Stefan Jaeger for assisting me with the experiments on snapshot algorithm evaluation. Many thanks to Kaushik Mitra and Zhe Lin for the useful discussions. I benefited a lot from their comments and suggestions.

Special thanks to Denise Best at the *Language and Media Processing Laboratory* for her help in administrative matters. Thanks to Maria Hoo from the ECE Graduate Office for her help with the official issues and paperwork at various stages in my graduate studies.

I would also like to thank Anne Geronimo and Kurt Flick who were my supervisors at the *Office of Research Administration and Advancement* during the first two

semesters at the University of Maryland. Working at ORAA was a wonderful experience and I am sure it will help me a lot in the professional life ahead.

I wish to thank my family for being very understanding and supportive in my pursuit for graduate studies. Their love and blessings have been a constant source of motivation and no amount of words can do justice to acknowledging them.

Finally I have made many friends along the way and they have helped me in some way or the other in the Graduate life. Thanks to my roommates Abhijit Deshmukh and Saurabh Srivastava, my friends Kaushik Mitra, Sandeep Manocha, Mudit Agrawal, Omkar Dandekar, Aniruddha Kembhavi, Ashis Banerjee, Amit Sirsamkar, Vijay Puppala, Aditya Kalyanpur, Suresh Kumar Santhana Vannan and Giridhar Ramachandran.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In recent years there has been a great deal of interest in developing robust algorithms for the automated analysis of surveillance video and enhancing current security systems. A typical state of the art video surveillance system consists of a network of security cameras, which either capture and record video or feed it to a central location for recording and monitoring by security personnel. The recorded video is often used for forensic purposes and is typically indexed by time and location so that it can be retrieved and played back. For real time video, however, it may be physically impossible for operators to watch all video feeds at the same time. Such videos may have no activity for long periods of time, making it difficult for a human to pay attention continuously. During lapses important details can be missed. However, if we can limit the human activity to key decision points and remove them from monitoring aspects of the system, efficiency can be improved. The goal is to automate the task of analyzing the video and making intelligent hypotheses to prompt the security personnel when abnormal events occur. A key challenge for surveillance systems is therefore not just recording and archiving video, but in making intelligent use of tools that will allow an operator to access the video in real time in response to a specific need.

Although the ultimate goal of surveillance video systems is to automatically detect various activities and alert the security personnel, current technology is far from adequate. We are a long way from being able to detect a car theft from a parking lot or a bank robbery being committed. The activity may appear normal (someone gets in

a car and drives away) or the scene can be too complex (for example, successfully tracking a person running in a crowded place).

There is also a class of problems where the goal is not necessarily to identify an activity when it is occurring, but rather to gather information as rapidly and reliably as possible, so that it can be used for forensic examination. For instance, consider an event like theft occurring in a parking lot. The security officer would like to use the recorded surveillance videos to identify what happened and the exact time instance when the theft occurred. He would then want to track those people to see where they came from or where they went, identify their vehicle in the parking lot or identify others they have been talking with. Unfortunately the current video recording systems provide little more than playback and time indexing of the stored video.

In this thesis we focus on providing a set of techniques that can be used to build tools to allow operators to control and manipulate video in meaningful ways such as outlined above. A key component of developing these capabilities is being able to manage the metadata provided by the low-level vision processes in a useful manner. Since doing video analysis on the fly is not practical, it is helpful to provide a structure where this metadata can be efficiently indexed and retrieved.

The thesis builds upon the work done previously at the University of Maryland on developing applications involving analysis, storage and retrieval of video containing moving people and vehicles. The existing framework consisted of a robust codebook

based background subtraction module in the form of an API [1], modules for tracking of objects segmented from the background and annotation of objects with MPEG-7 descriptors including color, shape and motion for incorporation to higher level retrieval systems [2, 3]. These modules are used as the basic tools upon which the work in this thesis is based.

In this work, we first explore the problem of determining an optimal view (i.e. a snapshot) of a person from a video sequence. In scenarios such as theft, the long duration surveillance videos need to be scanned quickly to extract maximum information possible for further investigation. Since human operators are very good at identifying people from snapshots, we explore the problem of automatically determining an optimal view of a person from an extended video. A measure of quality of the snapshot includes the ability to discern the face, proximity to the camera and a lack of occlusion. Given such a capability, the surveillance system could rapidly build a set of candidate suspects for the operator to view and allow them to choose suspects to follow.

Learning semantics of the video scene over long periods of time is another active area of research. Work has previously appeared in the literature on generating semantic models of scenes, learning entry/exit points, gathering points, routes and junctions [4]. In this thesis we learn static occlusion points in the scene using moving foreground blobs. The motivation is to determine locations of doors, desks and other stationary objects that occlude moving objects and to generate structural information

about the scene. Occlusion analysis can be useful for tracking of moving objects. It is known that tracking under occlusions is a difficult problem and various techniques have been presented. We present a mathematical model to locate static occlusions using overlap of the contours of foreground blobs obtained by robust background subtraction.

Another challenge in processing surveillance video is obtaining appropriate video sequences for development of surveillance algorithms. In addition to physical storage and memory constraints, there are legal issues in monitoring public areas like airports, train stations or people in indoor environment like offices or shopping malls. Although these areas need strong security systems, recording video for experimental purposes is prohibited in such places. Furthermore, recent trends toward statistical techniques require extensive training data. By noting that in most cases we need to store the metadata and not the actual video, we propose to address this problem by "simulating" certain video metadata for a given scene. We can then extract useful metadata features from the simulated data for low-level vision algorithm analysis rather than extracting the metadata from real video sequences. In this thesis we develop a tool to automatically generate metadata for surveillance scenarios with minimal user interaction.

Activity recognition and abnormal behavior detection in videos is also an active area of research in surveillance video analysis. The primary task of surveillance system is to assist the operator in preventing a security breach. Hence learning abnormal

behavior patterns that constitute "threat" will be immensely useful in preventing such situations. We consider the problem of modeling trajectory patterns that correspond to a "threat" relative to a pre-specified "target". Some examples of behaviors that constitute threat are - a person entering a restricted area, performing activities like running, loitering and fast approaching a specified target. We describe a model to compute the threat level at each trajectory point of the moving foreground object and present the progression of threat level to the operator visually. Thus our threat level scoring engine analyzes the trajectory and gives real time scoring based on the learned model. Visual alarm notification is generated when a threshold is reached.

The following summarizes our contributions:

1. An enhanced snapshot selection algorithm based on eigen-analysis of faces has been proposed and implemented. This algorithm has been tested on various video sequences and an evaluation of the results against manually labeled ground truths is presented.
2. A method to learn occlusions in a scene based on long observations has been proposed, implemented and evaluated.
3. A methodology to *simulate* the metadata for surveillance videos is proposed and implemented. The proposed technique is completely automated and can be used to generate metadata for any scenario with minimal user interaction.
4. A *threat detection model* using activity map analysis through long observations is proposed and implemented. We present the results for several scenarios.

The remainder of this thesis is organized in five chapters. In Chapter 2, we provide a literature review of techniques related to snapshot selection in videos and present our eigen-analysis based snapshot metadata extraction algorithm in detail. In Chapter 3 we discuss occlusion modeling using long observations in surveillance videos. A mathematical model, its implementation and results are described for various video sequences. Chapter 4 proposes a novel methodology to automatically simulate video metadata for a given surveillance scenario. The proposed technique is illustrated for an office and shopping mall scenario. The utility of this tool is demonstrated by using the simulated metadata for entry/exit zone modeling and route learning algorithms. In Chapter 5, a threat metadata model is described and results are presented for a shopping mall scenario. In Chapter 6, we summarize the accomplishments.

Chapter 2: Enhanced Snapshot Selection

Consider a scenario where a crime has been committed; the available video footage needs to be scanned quickly and information about the possible suspects needs to be collected as soon as possible. Since neither the perpetrator, nor the exact time when the crime was committed may be known, manual playback of a very long video sequence to locate the possible suspects may be required. We propose to automate this scanning process by extracting snapshots of people from the video sequence.

This problem is similar to the problem of key frame extraction from video with the added constraint of selecting visible faces. Recent approaches to extract key frames can be found in [5-8]. However, the objective of these algorithms is to pick key frames that capture the summary of the video. These authors present a video summary in the context of search, retrieval and browsing. Key frame extraction from the point of view of surveillance has also been studied in the literature. A multicamera surveillance strategy for non-overlapping cameras has been proposed by Porikli [9] where video sequences are generated for each tracked object instead of storing videos from each camera separately. Latecki et. al. have studied the problem of detecting changes in surveillance videos, including the appearance/disappearance of objects and changes in velocity and direction of moving objects using the trajectory data [10]. However, none of these address the problem of identifying snapshots of the people present. Human operators can be very accurate in identifying people and vehicles from snapshots. We explore the problem of determining the best discernable view of

a person from an extended video to produce a summary consisting of snapshots of people present. These snapshots will be presented to the operator for viewing. A measure of quality of the frame includes the ability to discern a face, size and absence of occlusion. Given such a capability, the system could rapidly build a set of candidate suspects for the operator to view and allow them to choose suspects to follow.

The first step in determining the best view of a person is to detect a face in the moving foreground objects. A measure of "faceness" of the blob is then used to decide the appropriateness of the image for snapshoting. (For example, a frontal face view would be more appropriate as a snapshot than a profile of the face.) In the following sections we describe the approach used for frontal face detection and ranking of frames as snapshots.

2.1 Frontal Face Detection

Numerous methods have been proposed to detect faces in a single grayscale or color image. Chellappa et. al. published a survey on face recognition and detection methods in 1995 [11] and a more recent survey on face detection alone was published by Yang et.al. [12] and provides the categorization as shown below (see Figure 1).

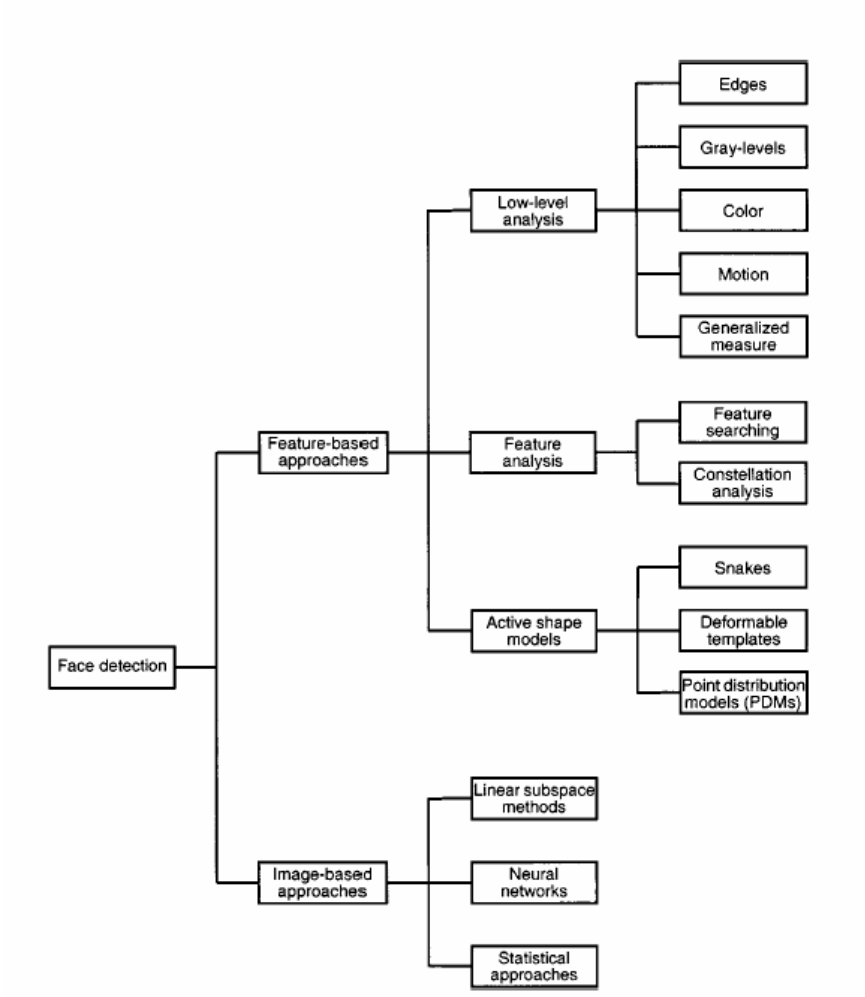


Figure 1: Categorization of methods for face detection [12]

Facial feature based methods commonly extract facial features such as eyebrows, eyes, noses, mouths, and hair-lines using edge detectors. Based on the extracted features, a statistical model is built to describe their relationships and to verify the existence of a face. Examples of these detectors have been presented by Govindaraju [13], Graf et. al. [14] and Leung et. al. [15].

Texture based approaches have been used by Dai and Nakano to separate faces from different objects [16]. In their work, color information is also incorporated with the face-texture model.

In template matching based approaches, a standard face pattern (typically frontal) is manually predefined or parameterized by a function. Given an input image, the correlations are computed with standard patterns for the face contour, eyes, nose, and mouth independently. The presence of a face is determined based on the correlation values. This approach is inadequate for face detection since it cannot effectively deal with variation in scale, pose, and shape [17].

Various neural network architectures have been proposed for face detection. The advantage of using neural networks for face detection is the feasibility of training a system to capture the complex class conditional density of face patterns. However, one drawback is that the network architecture has to be extensively tuned (number of layers, number of nodes, learning rates, etc.) to get exceptional performance [17, 18].

Another approach in face detection and recognition is the eigen-space decomposition [19, 20]. With this method, the image under consideration is projected into a low dimensional feature space that is spanned by the eigenvectors of a set of test faces. For the detection task, the image is reconstructed and the resulting error between original image and the reconstructed image is computed. An image pattern is classified as a face if its distance to the face space is smaller than a determined

threshold. PCA can also be used for face recognition by comparing the resulting PCA coefficients (principal components) to those of images in the database.

In this thesis, we use PCA based frontal face detection to build the snapshoting algorithm. It should be noted that any other face detector could be used in place of the PCA method and the approach for snapshot selection would remain the same. Experiments show that with the PCA based face detector, the background leads to a significant number of false classifications if the face region is relatively small. As color is the most discriminating feature of a facial region, the first step of many face detection algorithms is a pixel-based color segmentation to detect skin-colored regions. We also include color component information to localize the skin regions in the image. A skin color probability image is generated by color analysis and PCA is performed on the new image. This helps in identifying a relatively smaller subset of the original image for face detection rather than searching the whole luminance image and reduces the false detections.

2.2 Skin Color Detection

The color of human skin is distinctive from the color of many other natural objects, hence color is a very important feature that can be used for face detection [18]. Analyzing skin-tone color statistics, one observes that skin colors are distributed over a small area in the chrominance plane and the major difference between skin tones is intensity. Thus, the image is first converted into a color space that provides a separation into a luminance channel and two chrominance components in the case of the YCbCr color space. The distribution of the training skin pixels in the CbCr plane

is given in Figure 2. The figure shows that the color of human skin pixels is confined to a very small region in the chrominance space.

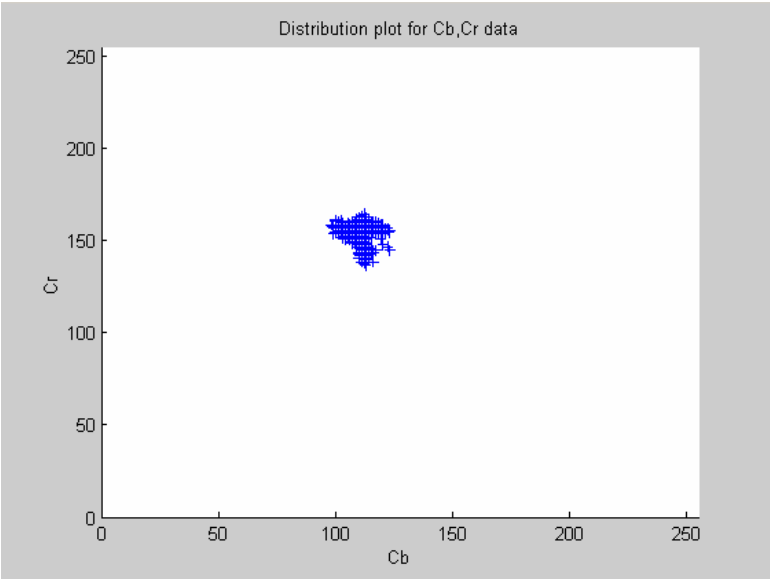


Figure 2: Distribution plot for Cb, Cr Data

2.2.1 Skin Color Database

Based on the above observation, the skin color distribution CbCr plane is modeled by a 2D Gaussian distribution. The database shown in Figure 3 is used to build the Gaussian model.

Figure 3: Database of sample images for skin color

The database is created by cropping subimages from the different color images of faces. A total 50 skin images of various size from 10x10 up to 50x50 are use for training. The images are manually cropped from colored face images taken from the CMU color face database [21]. The face database contains people with various complexions.

2.2.1 Skin Color Model

For a database of M training pixels, the Gaussian distribution is given as $N(\mu_s, \Sigma_s^2)$ with $\mu_s = (\overline{Cr}, \overline{Cb})$ where,

$$\overline{Cr} = \frac{1}{M}\sum_{i=1}^{M} Cr_i \qquad \text{(mean)} \qquad (2.1)$$

$$\overline{Cb} = \frac{1}{M}\sum_{i=1}^{M} Cb_i \qquad \text{(mean)} \qquad (2.2)$$

$$\Sigma_s = \begin{pmatrix} \sigma_{Cr,Cr} & \sigma_{Cr,Cb} \\ \sigma_{Cb,Cr} & \sigma_{Cb,Cb} \end{pmatrix} \quad \text{(covariance matrix)} \qquad (2.3)$$

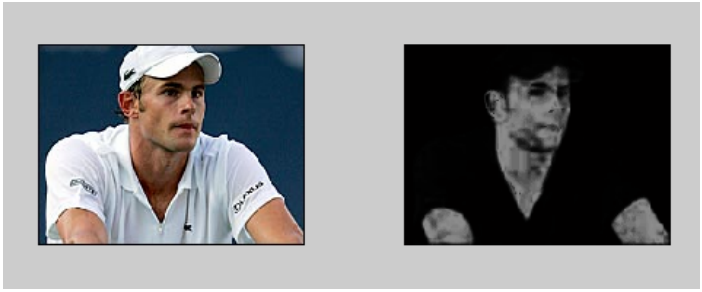
We create a skin probability image, which indicates the probability of each image pixel belonging t o the skin class S. Let $w_{ij} = [\text{Cb Cr}]^T$ denote the chrominance vector of an input pixel. Then the probability that the given pixel lies in the skin distribution is given by

$$p(w_{ij} \mid S) = \frac{\exp\left[-\frac{1}{2}\left(w_{ij} - \mu_S\right)^T \Sigma_S^{-1}\left(w_{ij} - \mu_S\right)\right]}{2\pi \mid \Sigma_S \mid^{\frac{1}{2}}} \qquad (2.4)$$

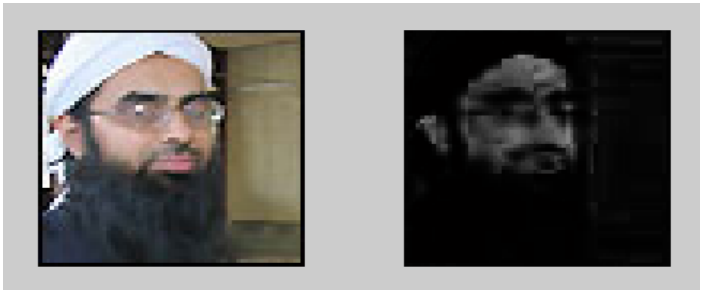
where, μ_s is the mean and Σs is the covariance matrix of the distribution.

2.2.3 Skin Detection Results

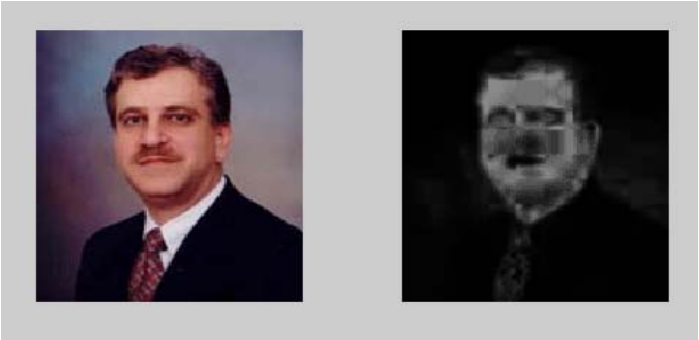
The skin probability images obtained after processing different test images are shown below. In each case, the image on the left is the original image and the image on the right is the intensity map obtained after skin detection. Since this is a probability map, the brighter regions in the skin map imply greater probability of skin color being present. Figures 4 (a), (b) and (c) show that the skin color on the face and arms is correctly detected with a higher probability. However, a person wearing skin colored clothes may give false alarms by this method.



(a)



(b)



(c)

Figure 4: (a) to (c) Original image vs. Skin probability image

2.3 Face detection using Principle Component Analysis

PCA for face detection is based on the eigenface representation of faces. The main idea is to decompose face images into a small set of characteristic feature images called eigenfaces, which may be thought of as the principal components of the original images. These eigenfaces function as the orthogonal basis vectors of a linear subspace called "face space". Detection is performed by projecting a new image into this face space and then computing its distance from face space [19, 20].

2.3.1 Construction of Face Space

Suppose a face image consists of *N* pixels, so it can be represented by a vector Γ of dimension *N*. Given a training set of *M* face images, the average face of these *M* images is given by

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i \tag{2.5}$$

Then each face Γ_i differs from the average face Ψ by Φ_i:

$$\Phi_i = \left(\Gamma_i - \Psi\right)^v \tag{2.6}$$

A covariance matrix of the training images can be constructed as follows:

$$C = DD^T \tag{2.7}$$

where $C = \lfloor \Phi_1 \Phi_2 \Phi_3 ... \Phi_M \rfloor$. The basis vectors of the face space, i.e., the eigenfaces, are then the orthogonal eigenvectors of the covariance matrix $\mathbf{C}$.

Finding the eigenvectors of the N by N matrix $\mathbf{C}$ is computationally very expensive task for typical image sizes, hence, a simplified calculation has to be adopted [19]. Since the number of training images is usually less than the number of pixels in an image, there will be only M-1, instead of N, meaningful eigenvectors. Therefore, the eigenfaces are computed by first finding the eigenvectors, $\boldsymbol{v_i}$ (i = 1 to M), of the M by M matrix $\boldsymbol{L}$:

$$L = D^T D \tag{2.8}$$

Then it can be shown that $u_i = \boldsymbol{D}.v_i$ is the eigenvector of $\mathbf{C}$

The eigenvectors are ordered in the decreasing order of eigenvalue. The eigenvector is rearranged into an $\sqrt{N} \times \sqrt{N}$ eigenface. These images give the eigenspace corresponding to the training database.

2.3.2 Projecting an Image on the Face Space

The new input image is first preprocessed by subtracting the mean image from it.

$$\Phi = \left(\Gamma - \Psi \right)^v \tag{2.9}$$

It is then projected onto the face space by taking the dot product between the eigenfaces and the preprocessed image.

$$w_i = u_i^T \Phi \qquad \text{for i = 1 to } p,\ p < M \qquad (2.10)$$

where p is the reduced face space dimension used for face detection and u_i is the normalized eigenvector.

2.3.3 Face Detection

The reconstructed image is obtained

$$\Phi_r = \sum_{i=1}^{p} u_i w_i \qquad (2.11)$$

The reconstruction error is obtained by the Euclidean distance between the original image and the reconstructed image.

$$\varepsilon = \| \Phi - \Phi_r \|^2 \qquad (2.12)$$

The error is referred to as "distance from face space" (DFFS). If the DFFS is greater than a threshold the input image is NOT a face. Otherwise it is classified as a face.

2.3.4 Implementation and Results

In this section we present the results obtained by the above method. A database of 86 images was used to construct the face space (downloaded from Rice database [22]. This contains images of 32 individuals with two to three different face images of the same person (see Figure 5). The sample images from the database are each of the size 30x25. The resulting eigenfaces for these images are shown in Figure 6. The skin detection algorithm was run on the input color image. For each resulting skin blob in the image, a sub-image was selected from the original image for face detection. The image was then converted into gray scale and PCA was run on the regions identified as skin blobs. The global minimum of all the candidate locations, whose error was

less than the threshold, was selected as a face region. Figures 7 through 11 show the results of skin detection and face detection for several test images.

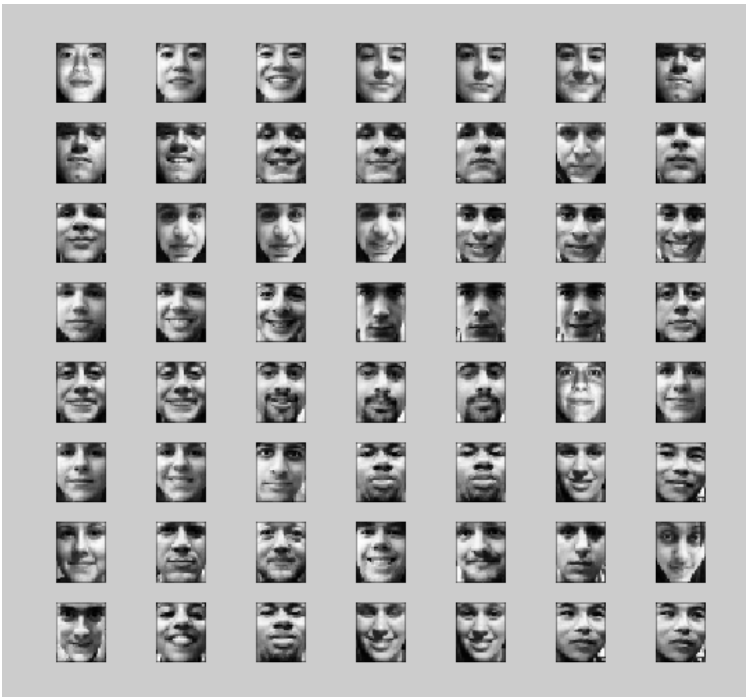


Figure 5: Sample face images from the database

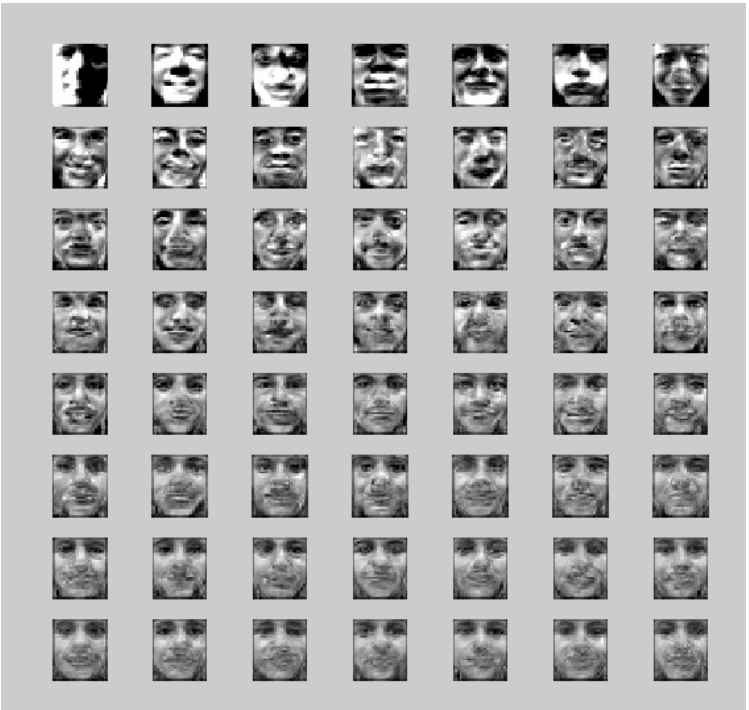


Figure 6: Eigenfaces generated from the above set of faces

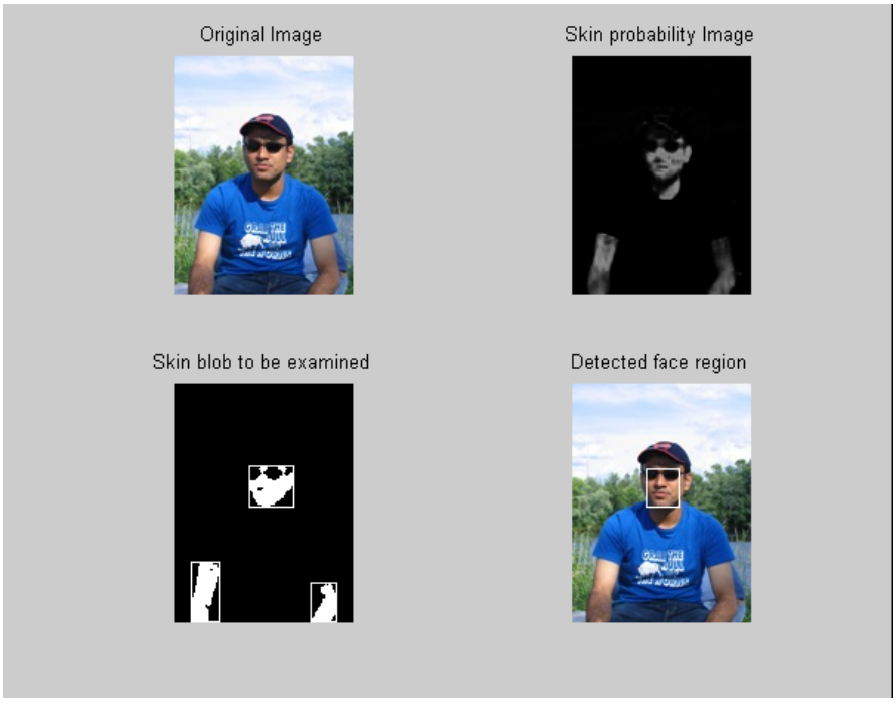


Figure 7: Face detection result – Correctly detected

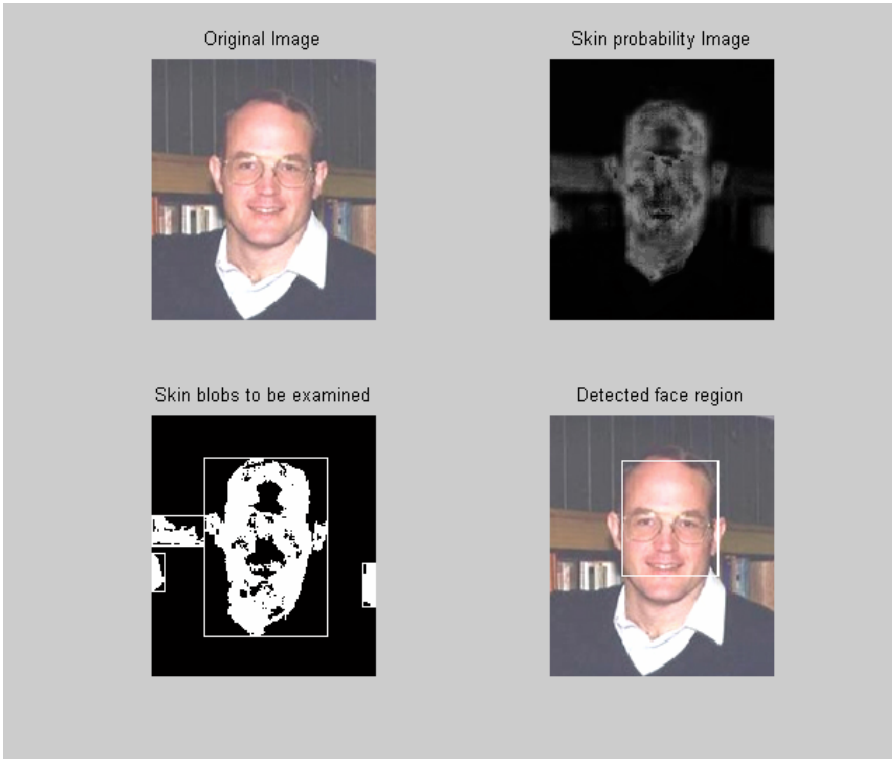


Figure 8: Face detection result- Correctly detected

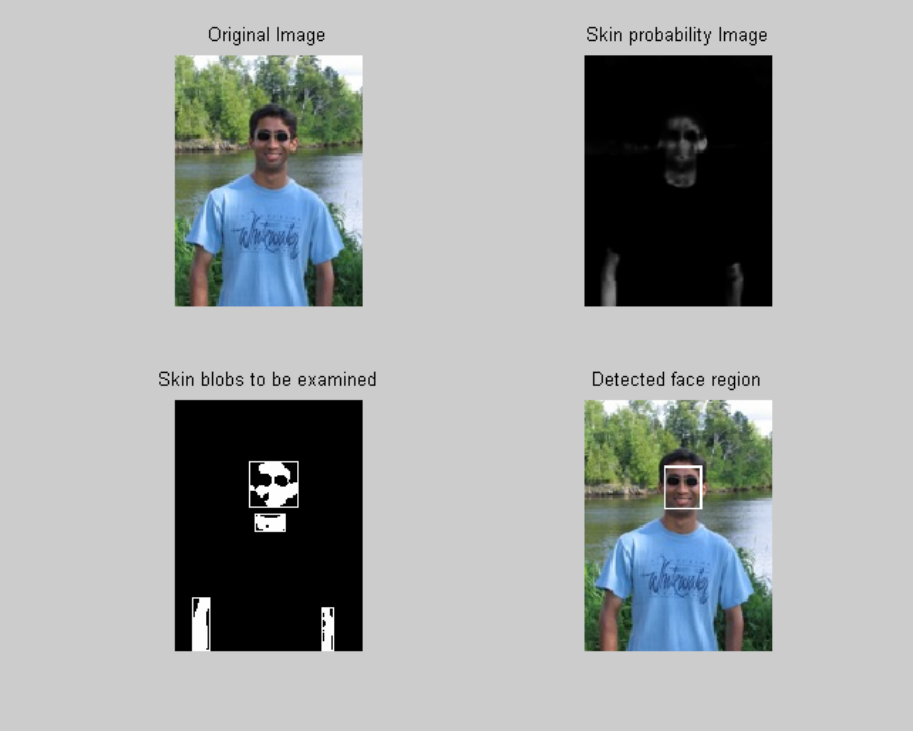


Figure 9: Face detection result- Correctly detected

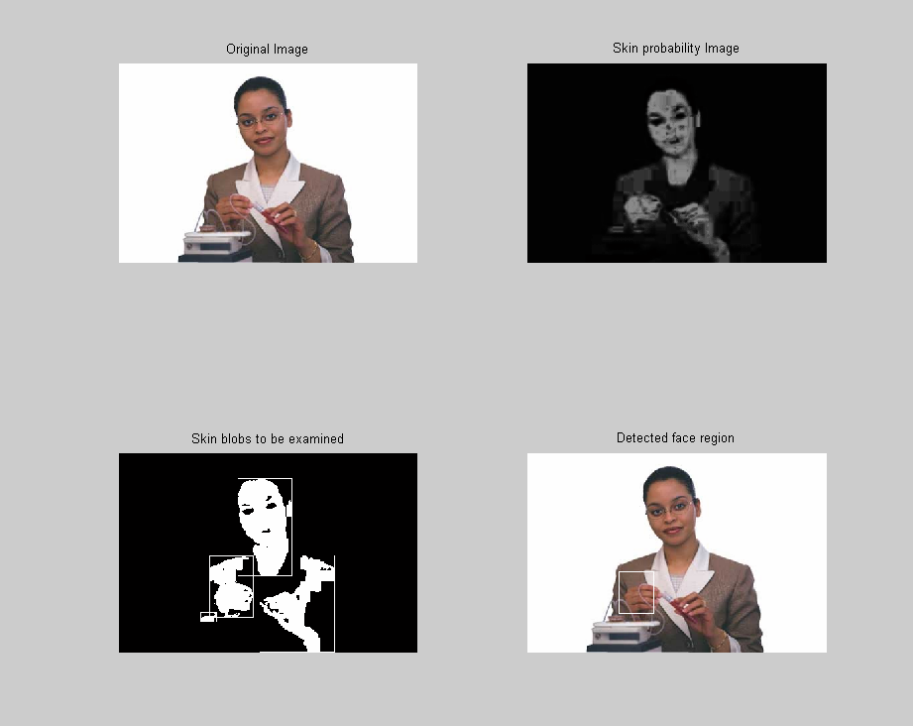


Figure 10: Face detection result – missed detection for a tilted face

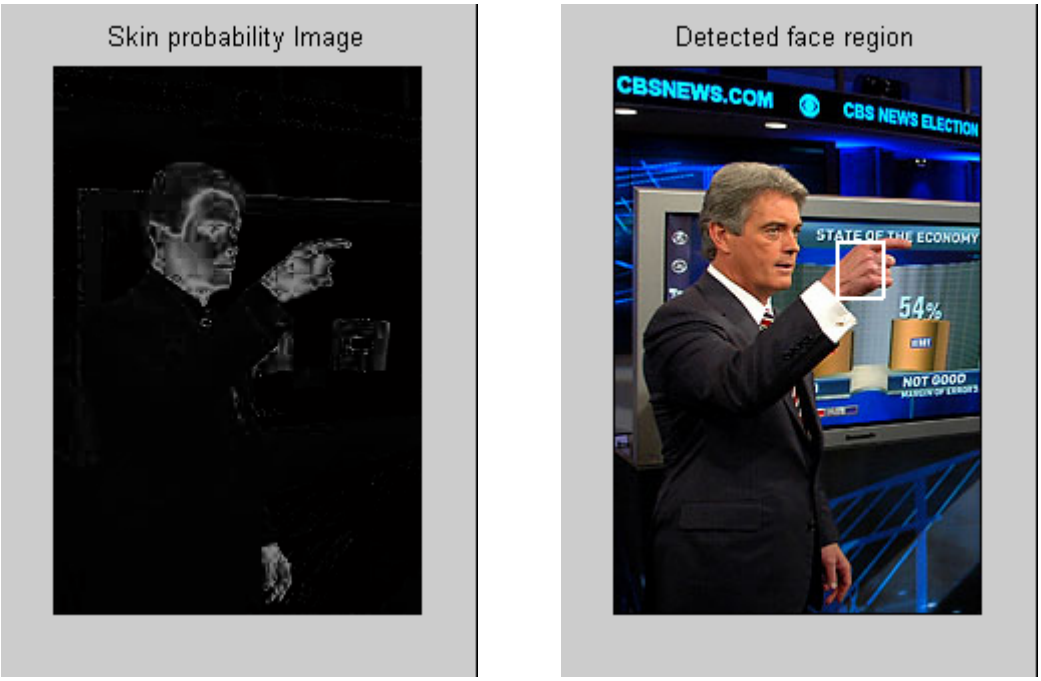


Figure 11: Face detection result – false detection for a profile face

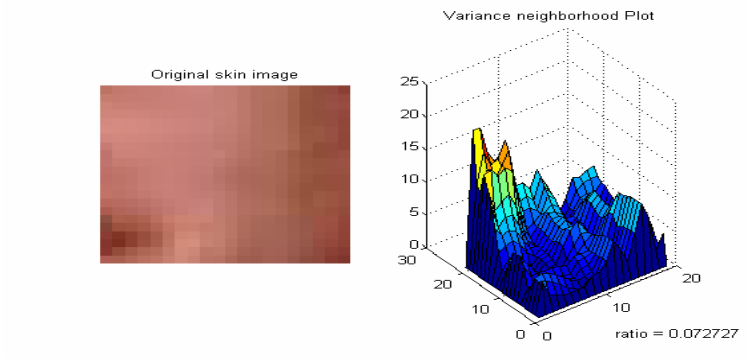
2.4 Reducing false alarms

As seen above, the face detection performs quite well for frontal faces, except for a few false alarms in cases like profiles. In addition, smooth blobs that sometimes correspond to hands or feet may produce false alarms. We filter out the hands and feet by doing ellipse fitting and measuring the ratio of major & minor axes. In addition, any skin blobs present below the object's centroid are typically not a face and can be rejected. To eliminate the smooth blobs, the blob smoothness measure proved very effective. Blob smoothness measure is defined as:

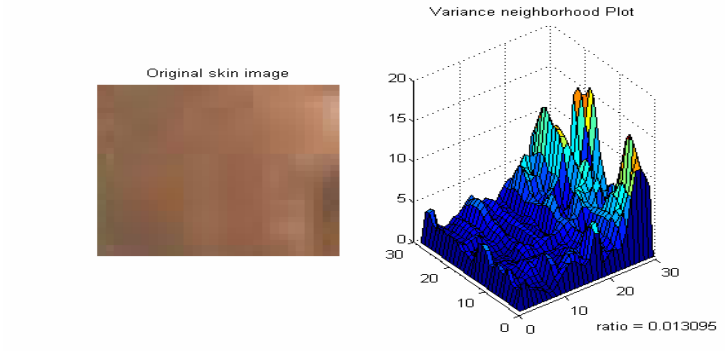
$$\gamma = \frac{\text{No of pixels with high variance neighborhood}}{\text{Total pixels in the skin blob}} \qquad (2.13)$$

Following figures shows the plots for variance neighborhood and the corresponding values for γ. The threshold for high variance is computed empirically and is set to 10.

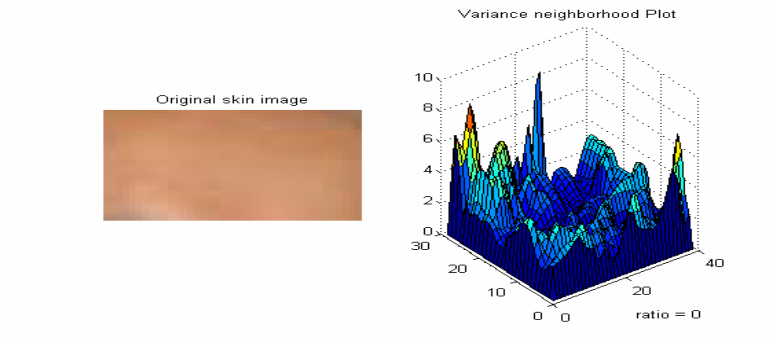
It can be seen that the smooth blobs (Figure 12 (a) to (e)) have much lower value of γ than the frontal face blobs (Figure 12 (f) to (i)) and therefore can be rejected.



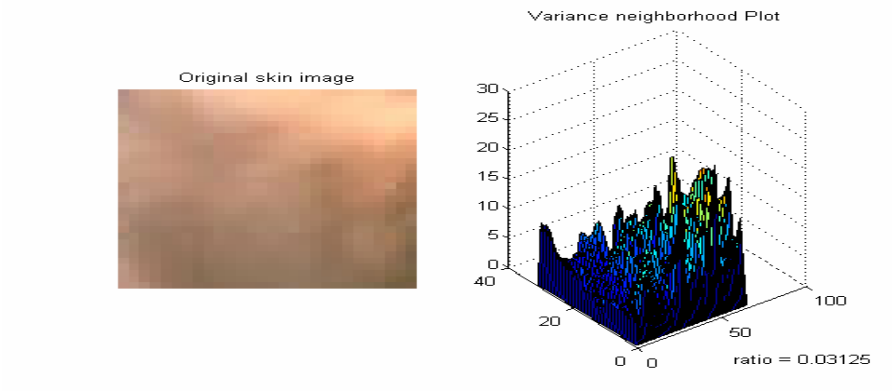
(a) $\gamma = 0.072727$



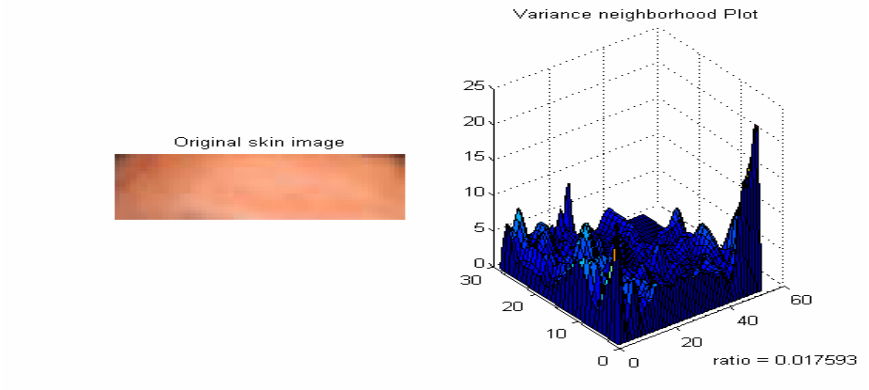
(b) $\gamma = 0.013095$



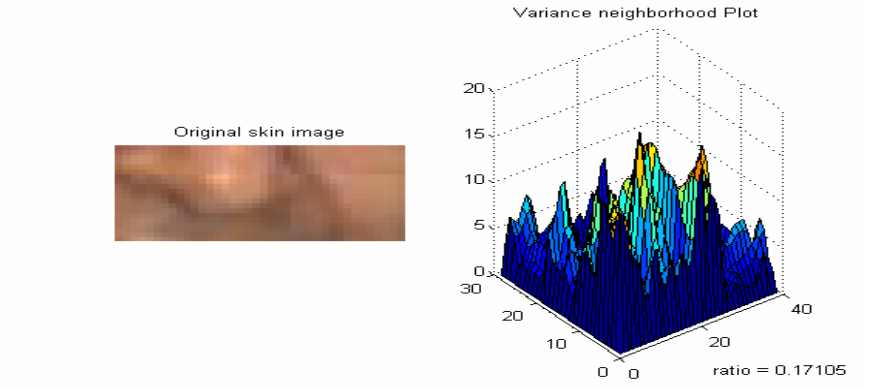
(c) $\gamma = 0$



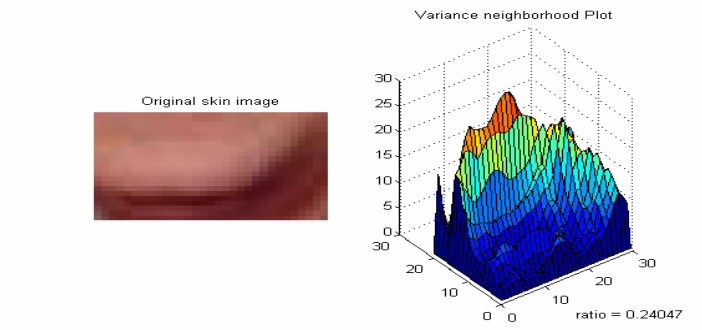
(d) γ= 0.03125



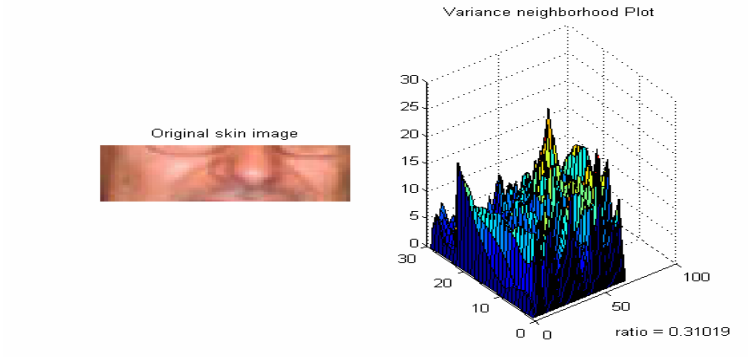
(e) γ = 0.017593



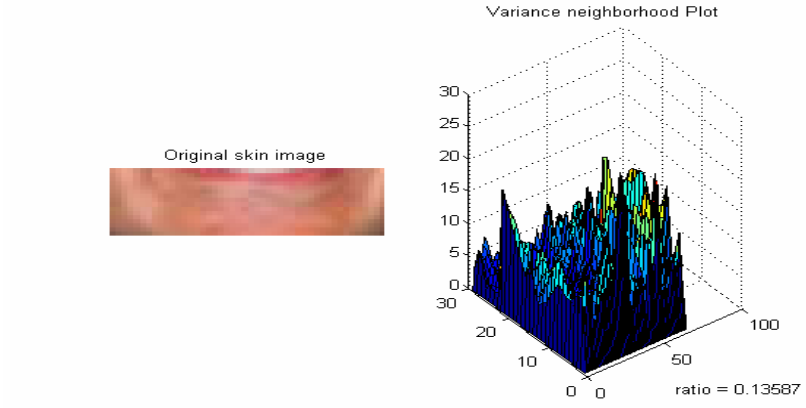
(f) γ = 0.17105



(g) $\gamma = 0.24047$



(h) $\gamma = 0.31019$



(i) $\gamma = 0.13587$

Figure 12 (a) to (i): Variance neighborhood plots for various face blobs with corresponding blob smoothness measure γ

2.4 Snapshot Algorithm Evaluation

2.4.1 Data Set

The problem of snapshot selection in a video can be considered as performing skin color and face detection on individual frames of the video. Thus the algorithms previously discussed i.e. skin and face detection on images can be applied in the context of video by considering one frame of the video at a time. These frames are finally ranked in decreasing order of their face confidence measure. We consider ten test video sequences to evaluate the performance of the snapshoting algorithm. Each sequence is between 500 and 1000 frames with one or more people each. In these videos, a subject typically walks in the scene along a circle or back and forth from one end to the other. The subject looks at the camera for a few seconds while moving, thereby allowing frontal face to be captured in a few frames in the video. The remaining frames contain side or back views of the subject where the face may or may not be visible. The snapshot summary of such a video sequence should contain the frontal faces as snapshots. The sequences were manually evaluated by a group of 20 participants who were asked to choose up to five best (subjective) frames for each video sequences that would qualify as snapshots (i.e. help establish the identity of the person in the video). The participants were asked to rank the chosen frames in decreasing order of their usefulness as snapshots. The frame ranks are attached weights from 5 (highest ranking snapshot) to 1 (lowest ranking snapshot). For a given video, all frames selected by participants are included in the snapshot ground truth data. By counting the number of votes each frames receives, the snapshot confidence measures is calculated as follows:

$$ConfidenceMeasure_i = \sum_j w_j \qquad \text{for all i and all j} \qquad (2.14)$$

where i is the frame number in the video sequence, j is the participant number and w_j is the weight attached to the vote by participant j for frame i. Snapshots are then ranked in the decreasing order of their confidence measure. To illustrate this procedure, we shall consider an example. Let us say the following frames are chosen and ranked as snapshots by a group of three participants for a video consisting of 100 frames.

Frame Rank	**Weight Factor**	**Frames chosen by Participant 1 (Frame #)**	**Frames chosen by Participant 2 (Frame #)**	**Frames chosen by Participant 3 (Frame #)**
1	5	10	15	10
2	4	15	20	11
3	3	20	10	15
4	2	45	45	5
5	1	12	11	20

Table 1: Example of snapshot ground truth generation by voting

Frame Number	**Weight Factor**			**Confidence Measure**
	Participant 1	**Participant 2**	**Participant 3**	
5	-	-	2	2
10	5	3	5	13
11	-	1	4	5
12	1	-	-	1

15	4	5	3	12
20	3	4	1	8
45	2	2	-	4

Table 2: Computing confidence measure to rank snapshots in the ground truth summary

The frames are sorted in decreasing order of the confidence measure, to generate the ground truth snapshot summary for the video sequence. In the above example, the final ground truth summary will be:

Snapshot Rank	Frame #	Confidence Measure
1	10	13
2	15	12
3	20	8
4	11	5
5	45	4
6	5	2
7	12	1

Table 3: Final snapshot ground truth summary

To evaluate summary obtained by the snapshot selection algorithm, the algorithm was applied to each sequence and the DDFS values were monitored. The candidate images were then ranked in the decreasing order of DFFS and these snapshot results were compared with the ground truth data. Figure 13 shows a summary of the steps involved in snapshot algorithm.

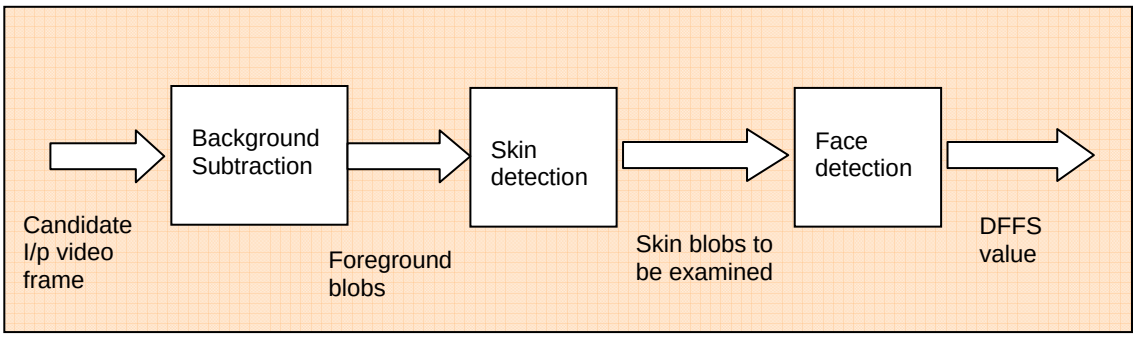


Figure 13: Steps involved in Snapshot algorithm

2.4.1 Metrics: Recall and Precision

Recall is a measure of how many frames in the ground truth are matched by the candidate snapshot summary [23]. A reference frame is matched if there exists at least one frame in the candidate summary that is similar to the reference frame according to some preset criterion. Recall is computed as:

$$Recall = N_{ref\_m} / N_{ref} \tag{2.15}$$

where N_{ref} is the total number of frames in the reference summary and N_{ref_m} is the total number of reference frames being matched.

Precision is a measure of how many frames in the candidate snapshot summary match the frames in the reference summary. Intuitively, it gives an indication of how precise the candidate summary is.

The cumulative average precision (AP) is defined as:

$$AP(i) = \sum_{k=1}^{i} p(k) / i \tag{2.16}$$

where p(k) is the precision at frame k given by:

$$Precision = N_{can\_m} / N_{can} \tag{2.17}$$

N_{can} is the total number of frames in the candidate summary and N_{can_m} is the number of frames being matched.

2.4.2 Results

Sequence #	Snapshot Ground Truth	Snapshot algorithm results
1	18 17 5 15 34 16 1 4 6	35 23 18 5 4 22 3 19 2
2	9 10 27 28 26 18 30 29 11 17	29 30 9 34 31 26 10 32 28 33 27 24
3	8 7 9 10 11 4	12 11 8 7 10
4	8 7 9 10 5 1 4	5 9 8 1 12 2 15 14 13
5	8 10 9 11 2 4 1 6	2 3 11 10 4 5 1
6	7 9 8 10 1 4 3	9 7 8 1 10 2 3 4
7	11 12 9 10 1 6 5 8	11 4 3 10 2 9 12 5 1
8	8 9 11 12 10 2	12 1 8 11 10 9 14 2
9	9 10 11 8 12 4 13 16	17 11 10 9 16 15 7 8 14
10	9 10 15 14 8 11 13 2 12	9 11

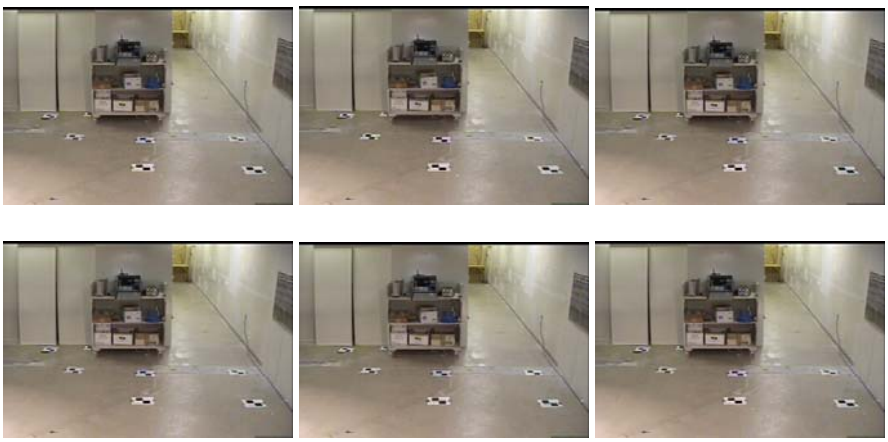
Table 4: Comparison of Ground truths and snapshot algorithm results for the test sequences

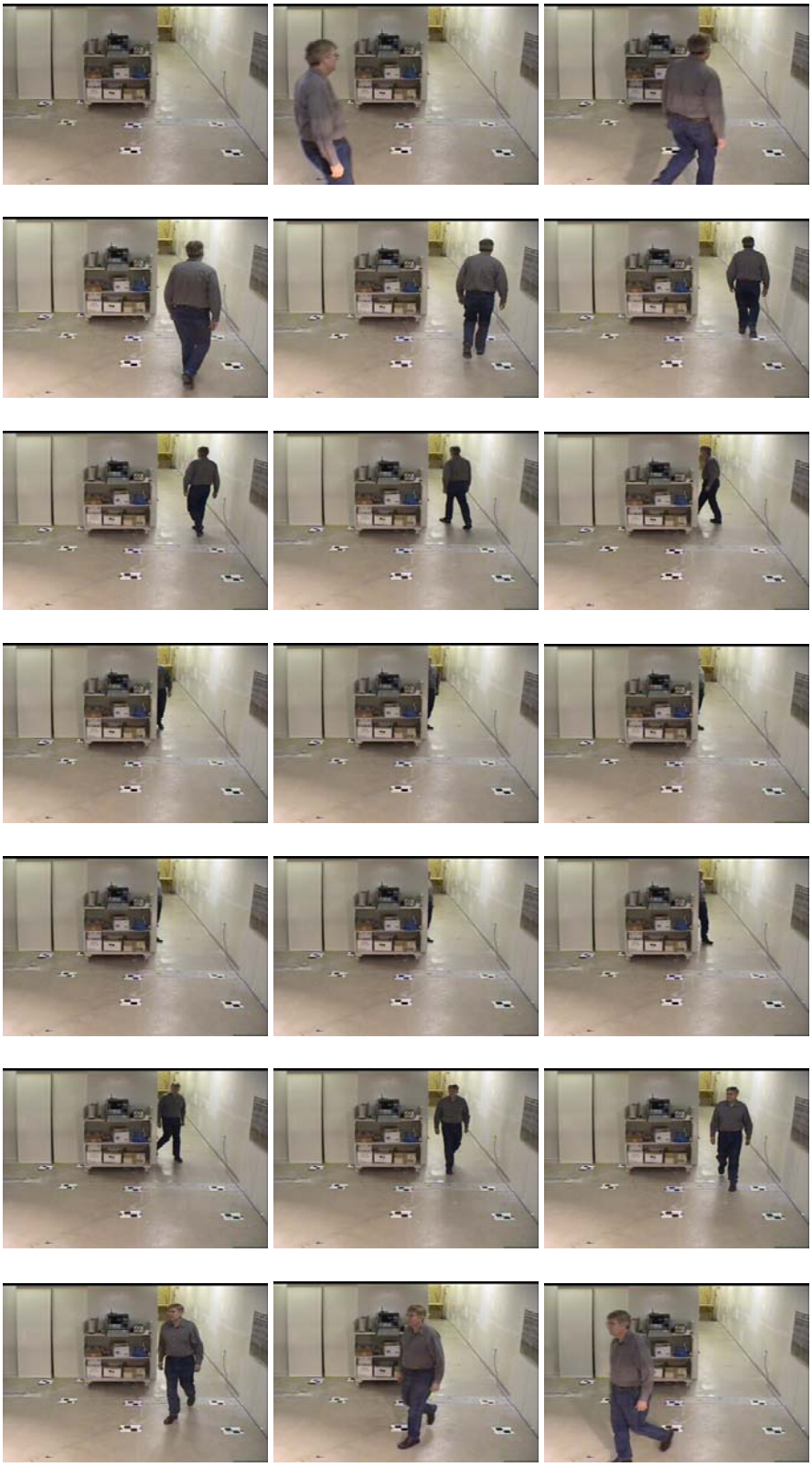
Sequence #	Recall (%)	Precision
1	66.67	0.68
2	80	0.827
3	66.67	0.659

4	57.14	0.728
5	62.5	0.782
6	100	0.961
7	75	0.661
8	100	0.8
9	62.5	0.679
10	22.2	1.0

Table 5: Recall and Precision results the for the test sequences

Figures 14, 17 and 20 show three video sequences for snapshot algorithm testing. Figures 15, 18 and 21 show the DFFS plots for these sequences respectively. The final selected snapshots are shown in Figures 16, 19 and 22 respectively. Table 1 shows a comparison of ground truths and snapshot algorithm results the for the test sequences. Recall and precision results for the snapshot algorithm are shown in Table 2.





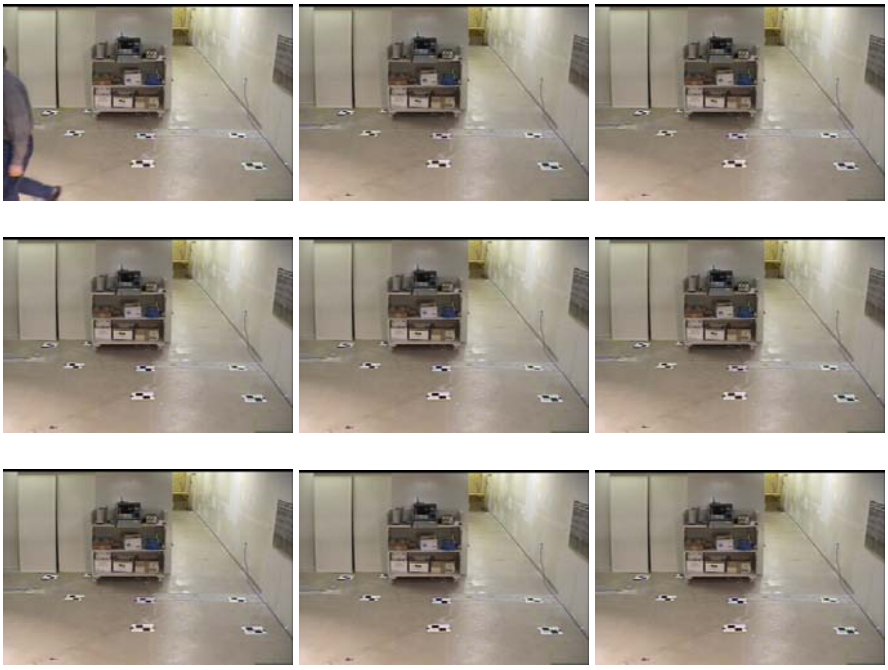


Figure 14 Test Sequence

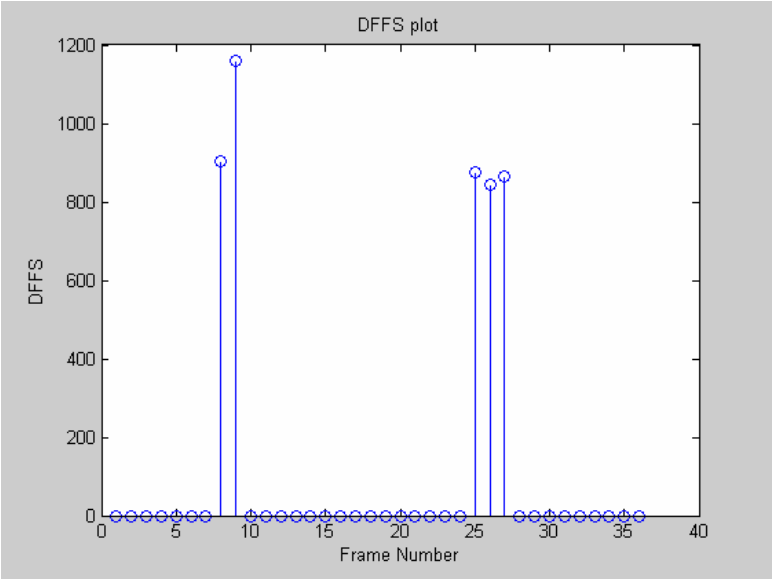
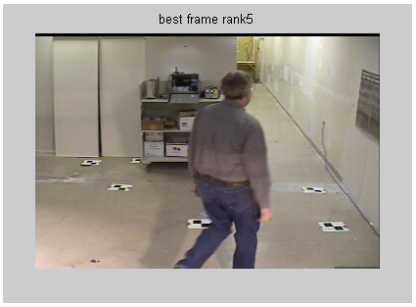


Figure 15: Distance from Face Space (DFFS) plot for sequence in Figure 17

26 27

24 8



9

Figure 16: Optimal snapshots for sequence in Figure 17

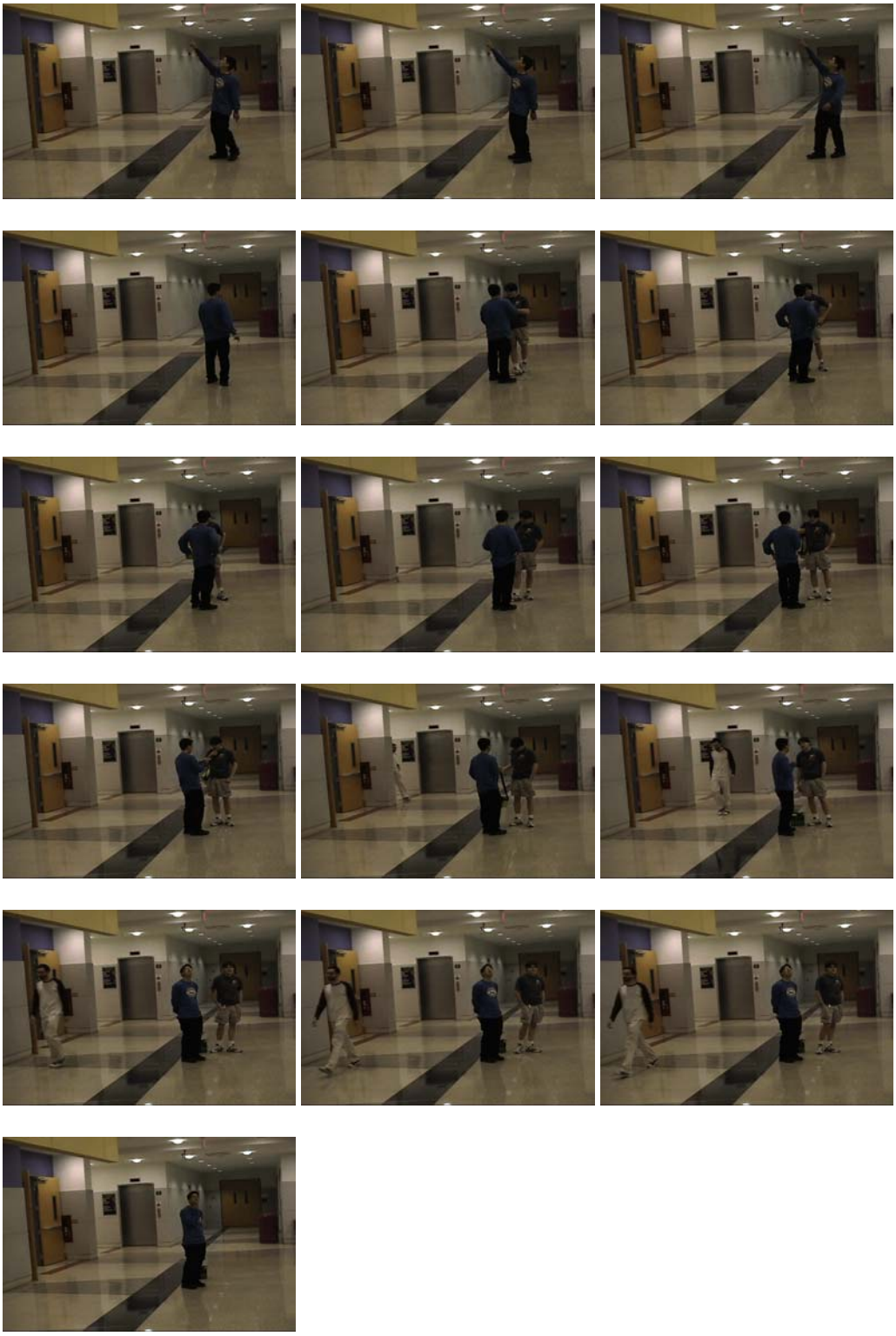


Figure 17: Candidate frames for a test Sequence

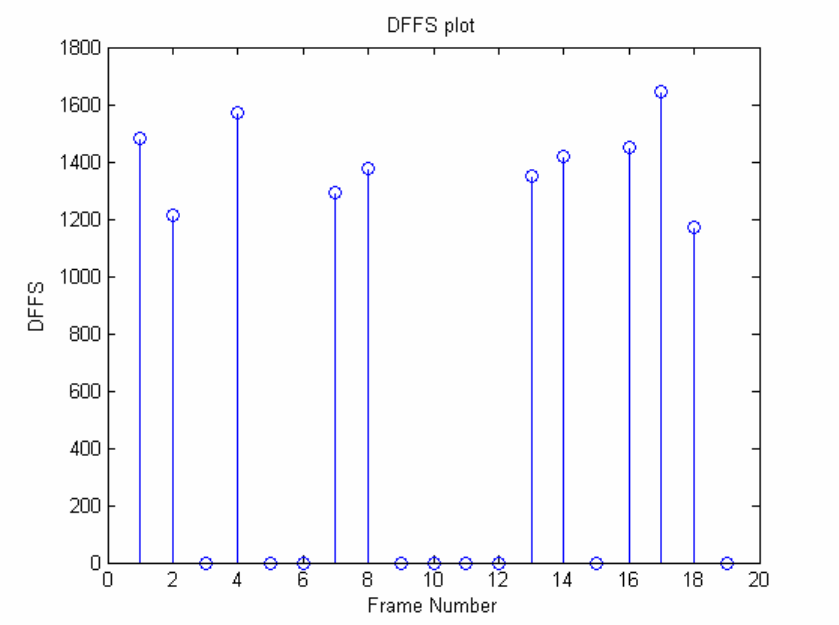
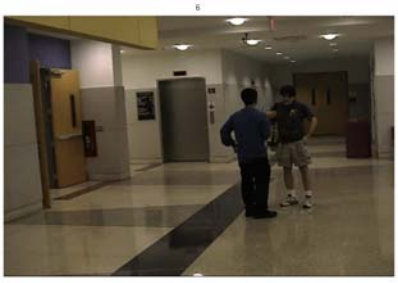


Figure 18: Distance from Face Space (DFFS) plot for sequence in Figure 19

18

2

7



11

8

Figure 19: Optimal snapshots for sequence in Figure 20

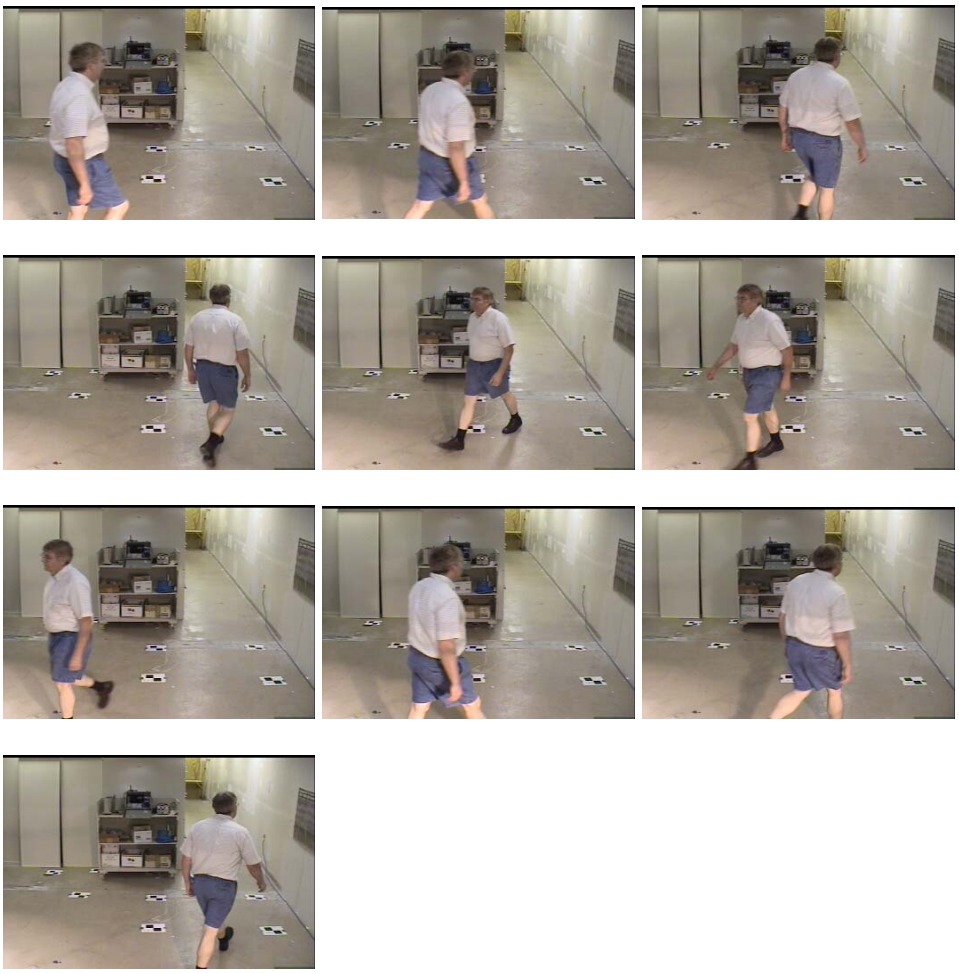


Figure 20: Candidate frames for a test sequence

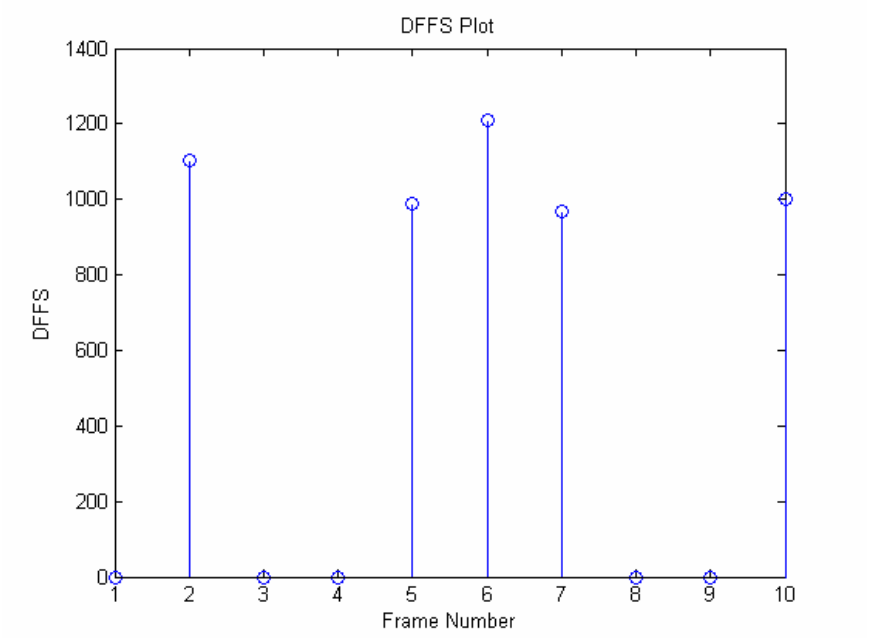
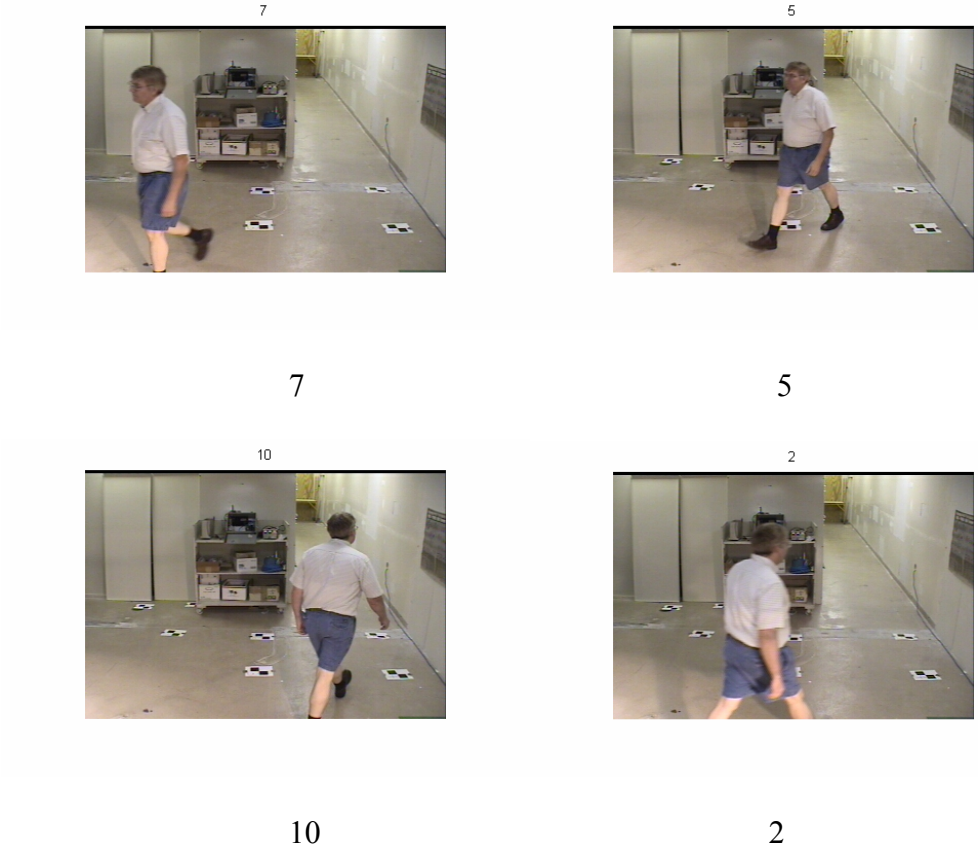


Figure 21: DFFS Plot for sequence in Figure 23



7 5

10 2

6

Figure 22: Optimal snapshots for sequence in Figure 20

2.4.3 Discussion

The snapshot results in Figures 16, 19 and 22 show that the performance of the snapshot algorithm depends upon the accuracy of frontal face detection engine. In some cases false alarms in face detection can result in incorrect snapshots, for instance the frames where a person wears skin colored clothes may lead to false alarms for skin detection (Figure 22).

The snapshot summary may therefore contain some outliers. However, the recall and precision figures indicate that in most cases the ground truth summary frames appear in top few frames of the snapshot summary.

The algorithm can be extended to multiple camera scenarios, where tracking results can be used to correlate the same object across multiple cameras and their snapshots results can be combined.

Chapter 3: Occlusion Modeling

3.1 Introduction

One fundamental problem of automated surveillance is to track foreground objects reliably as they move in the scene. The tracking of an object becomes difficult when the object is either partially or completely occluded by structures such as walls, doors or desks or the object interacts with other moving objects in the scene and gets occluded. Several techniques have been presented to successfully track objects under occlusion. A typical blob based method detects occlusions by merge and split of blobs [24]. Haritaoglu et al. have used appearance models to handle occlusion problem. They combine the gray-scale texture appearance and shape information of a person together in a 2D dynamic template [24]. Roh et al. use an appearance model based on temporal color to track multiple people in the presence of occlusion. They combine color values with associated weights determined by the size, duration, frequency of an object [25]. Senior et. al. use an adaptive color based appearance model to track people under occlusion [26]. Greenhill et. al. model structural static occlusions in the scene by generating a depth map of the scene structure at each pixel from a training set of observations of detected moving people [27]. The depth map is used to predict the visibility of the object at any point in the image and improves the accuracy of tracking under occlusion. In addition, stereo cameras have also been used for automatic detection of static occlusions in a scene [28].

These techniques involve the analysis of complex appearance models and are often computationally intensive. We present a simple, yet efficient technique to gather

structural information about the "static occlusions" in the scene using long observations from surveillance video. The motivation for learning occlusions is to determine locations of doors, desks and other stationary objects that occlude the moving objects in the scene and generate structural information about the scene. This information will be useful in improving the accuracy of tracking modules. For instance, when the object passes by an occlusion we can adapt the appearance model and use only the visible part of object (which is not occluded) for tracking. Or we can choose to reset the appearance model and end the track at the occlusion.

In this section we present a model to learn static occlusions in the scene using contours of foreground blobs obtained by robust background subtraction. The results of occlusion modeling are shown for sample video sequences and compared with the ground truth.

3.2 Approach 1: Contour overlap

3.2.1 Overview

When an object moves into an occlusion, the contour of the object changes at all points except at the points that correspond to the occlusion. This means that the contours of a moving object overlap (generally partially) along the edges of occluding objects like a door or desk. For example, Figure 23 (a) shows sample frames from a five minute sequence recorded at 30 fps. Here, a person walks back and forth in the scene which consists of a door and partitioning wall of a cubicle as the occlusions. We perform robust codebook based background subtraction on the video frames and

extract the moving foreground blobs [1]. By tracing the boundary of the detected foreground blob, we extract its boundary contours. Figures 24 (a) and (b) show two contours of the person from successive frames extracted by the background subtraction and boundary extraction processing. In Figure 24 (c) these contours are shown superimposed. The right edge with maximum contour overlap corresponds to the cubicle partitioning wall in the scene. In the remaining portions of the contours, there is not a strong overlap as the person has moved.

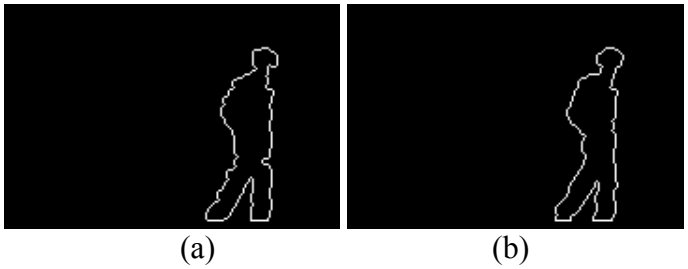
We accumulate the overlap counts for each pixel for the successive frames in the video sequence. The overlap between contours is determined by a binary AND operation on the binary contour images. The resulting boundary overlap count is displayed in the form of intensity map in Figure 25. As the number of observed frames increases (Figure 25 (a) to Figure 25 (h)) the occlusions become stronger. The final occlusion map is obtained by applying a Canny edge detector on Figure 25 (h) and is shown in Figure 25 (i). The Canny edge detector finds edges by looking for local maxima of the gradient of image [29]. The gradient is calculated using the derivative of a Gaussian filter. The Canny method uses two thresholds, to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. This results is a binarized occlusion map and a comparison to the ground truth Figure 23 (b) shows the map overlaps with the ground truth occlusion boundaries wherever the foreground objects have been occluded by these edges. The top edge of the door has not been captured by the occlusion map as the boundary overlaps are weak along this edge.

3.2.2 Results

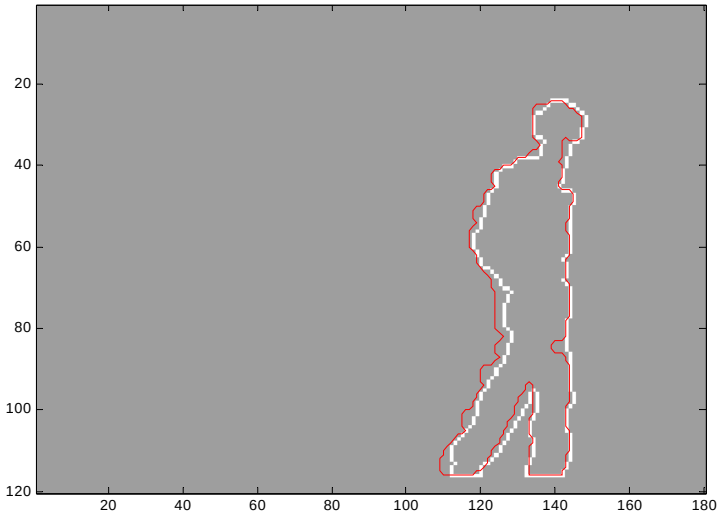
(a)

(b)

Figure 23: (a) Indoor scene for occlusion analysis (b) Ground truth. Occlusion edges marked in red

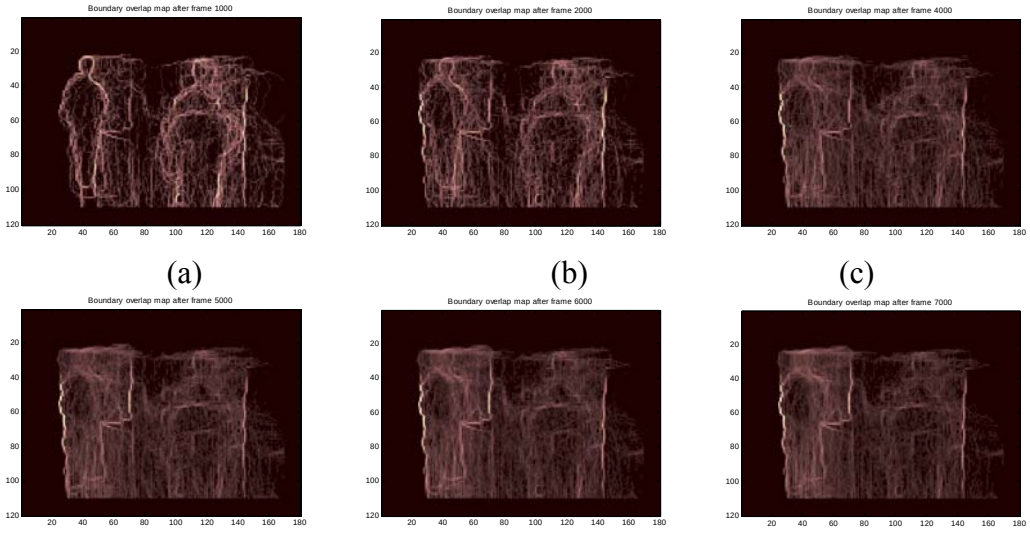


(a) (b)



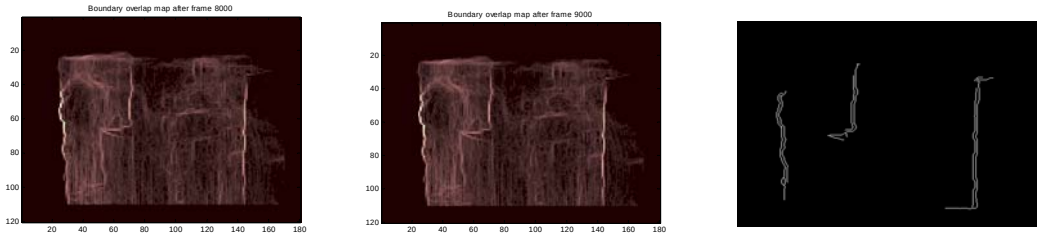
(c)

Figure 24: (a) and (b) contours of moving object for successive frames (c) contours from (a) and (b) shown overlapped. The right edge of the contours corresponds to an occlusion in the scene.



(a) (b) (c)

(d) (e) (f)



(g) (h) (i)

Figure 25: Occlusion map after (a)1000 frames (b)2000 frames (c)3000 frames (d)4000 frames (e)5000 frames (f)6000 frames (g) 8000 frames (h)9000 frames (i) Final occlusion map by applying Canny filter.

3.2.3 Enhancement

Although the result in Figure 25 are as expected along the occluding edges (large value in the boundary overlap map), the result is very noisy. This is because we consider only successive frames in finding boundary overlap votes. We can remove the false and noisy votes by noting that in the presence of an occlusion, the overlap along the contours of the object will persist for several frames with the number depending on the speed at which the object is moving. We use this observation to modify the boundary overlap strategy in the following manner:

$$I_{boundary} = \{[(I_1 \,\&\, I_2) + (I_2 \,\&\, I_3) + (I_3 \,\&\, I_4) + (I_4 \,\&\, I_5) + (I_5 \,\&\, I_6)] > \tau\} \quad (3.1)$$

$$I_{result} = I_{result} + I_{boundary} \quad (3.2)$$

where $\&$ is the binary AND operation, I_1 to I_6 are six successive images, $I_{boundary}$ is the boundary overlap map for these six images, I_{result} is the accumulated boundary overlap image i.e. occlusion map and τ is a threshold value.

The results for this approach are shown in Figure 26. It can be seen that the occluding edges are stronger and the noise in the occlusion map has been reduced. Figure 26 (j)

shows the result of applying Canny edge detector on the occlusion map in Figure 26 (i). This result is closer to the ground truth map in Figure 23 (b) than the result in Figure 25 (i). The occluding edges in the scene that the objects pass during learning are correctly identified by the algorithm.

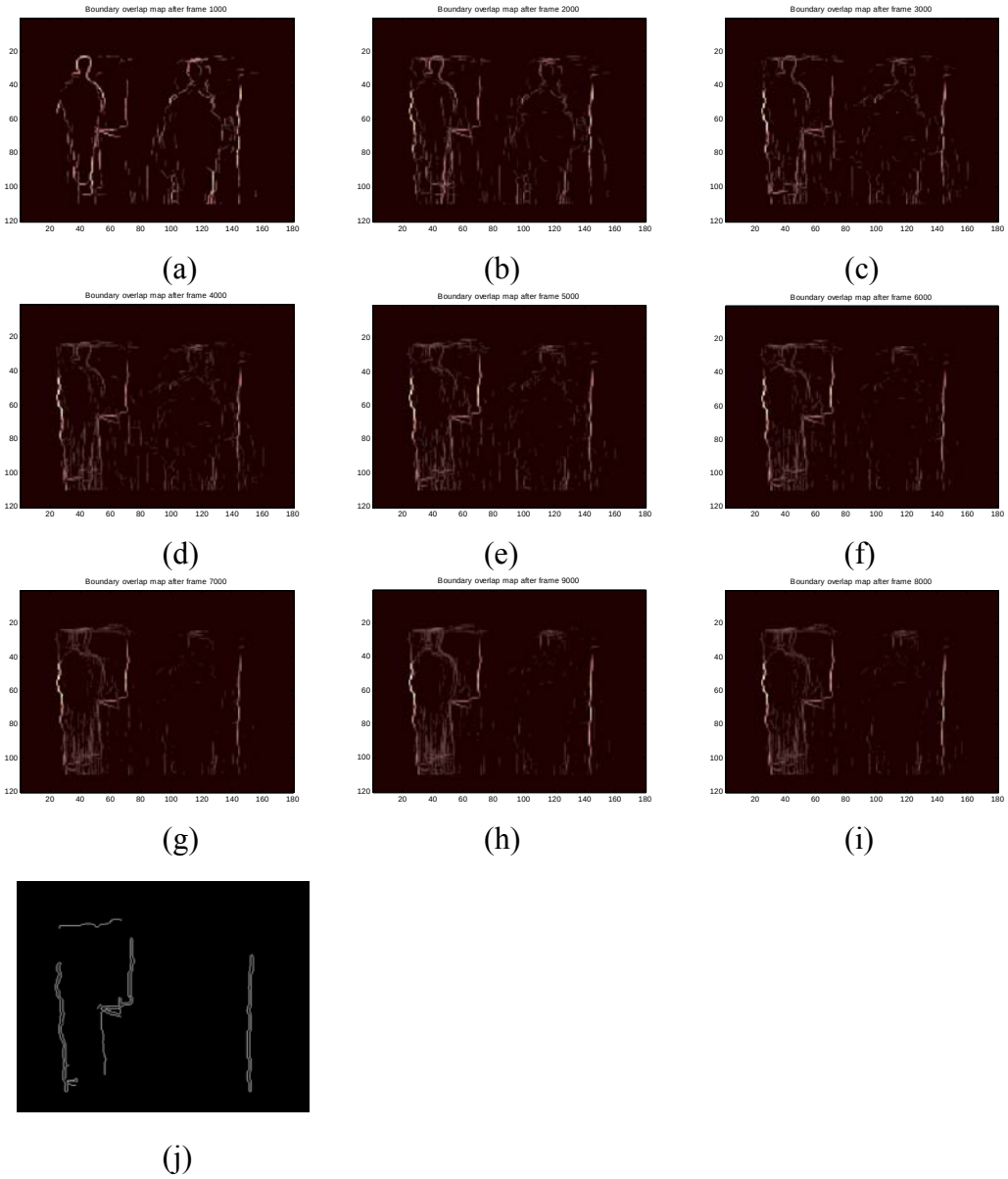
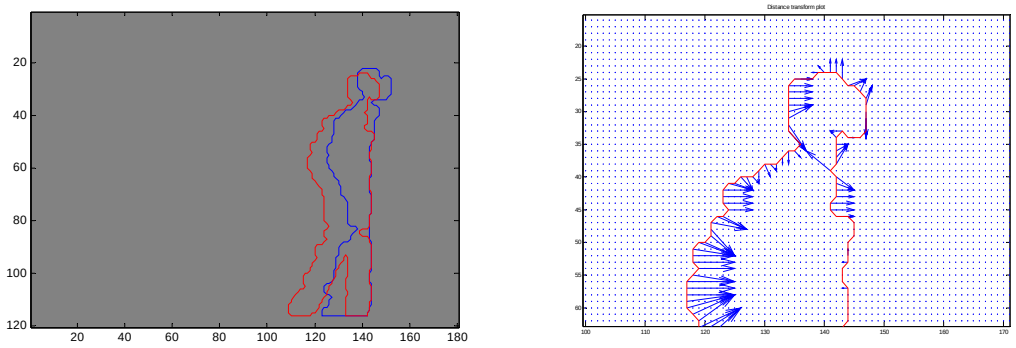


Figure 26: Occlusion map after (a)1000 frames (b)2000 frames (c)3000 frames (d)4000 frames (e)5000 frames (f)6000 frames (g) 7000 frames (h)8000 frames (i)9000 frames (j) Final occlusion map by applying Canny filter

3.3 Approach 2: Distance Transform

3.3.1 Overview

Since we are interested in locations where the contours of the moving object overlap, we can also use a distance transform metric to determine the pixels that are close to each other in the successive contours. Traditionally, the result of distance transform applied to a binary image is a gray level image where the gray level intensity of points inside foreground (white) regions are calculated based on the distance to the closest boundary from each point [30]. In this context, we define the contour distance transform as the distance from a point on first contour to the nearest point on the second contour. This is illustrated in Figure 27. Figure 27 (a) shows two contours from successive frames superimposed. A blue arrow in Figure 27 (b) show the correspondence between a point on contour 1 and its nearest point on contour 2. Contour distance transform is the length of this arrow.



(a) (b)

Figure 27: (a) Overlapping contours from successive frames (b) Contour distance transform

The contour distance transform is thresholded and resulting occlusion map is accumulated as follows:

$$I_{occlusion} = \{[\gamma(I_1, I_2) + \gamma(I_2, I_3) + \gamma(I_3, I_4) + \gamma(I_4, I_5) + \gamma(I_5, I_6)] < \tau\} \quad (3.3)$$

$$I_{result} = I_{result} + I_{occlusion} \quad (3.4)$$

where $\gamma(I,J)$ is the result of contour distance transform of images I and J, I_1 to I_6 are the six successive images, $I_{occlusion}$ is the occlusion map for these six images, I_{result} is the net accumulated occlusion map, τ is a threshold value.

3.3.2 Results

The results for this approach are shown below. Figure 28 represents sample images from another indoor video sequence recorded at 30 fps (about 8 minutes). Figure 29 shows the ground truth occlusion map where the occluding edges are marked red. The stronger edges in the intensity images shown in Figure 30(a) to 30(h) belong to the occlusions in the scene. The occlusion map appears stronger as the number of observed frames increases (Figure 30(a) to Figure 30(h)). The final occlusion map obtained by applying Canny edge detector on Figure 30(h) is shown in Figure 30(i).

Figure 28: Scene for occlusion analysis

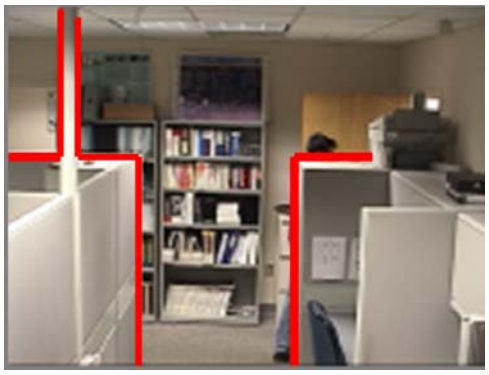
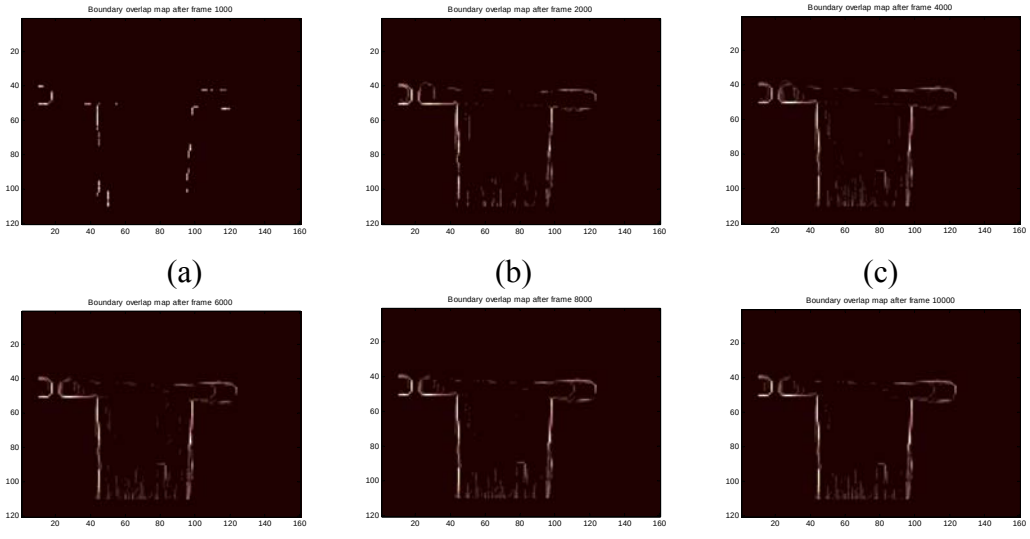


Figure 29: Ground truth. Occlusion edges marked in red



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 30: Occlusion map for scene in Figure 28 after (a) 1000 frames (b) 3000 frames (c) 5000 frames (d) 7000 frames (e) 10000 frames (f) 11000 frames (g) 13000 frames (h) 15000 frames (i) Final occlusion map by applying Canny filter

3.4 Discussion

We have presented two approaches to learn occlusions in a scene. Both the methods are based on the observation that boundaries of a moving object overlap with the edges of the occlusion as the object passes the occluding edge. By accumulating the overlaps we obtain an intensity map that indicates the occlusions. A Canny filter is used to connect the strong edges and binarize the intensity map.

The results of our approach have been presented for indoor office scenarios with video sequences of between 5 and 8 minutes duration. The occlusion map is seen to grow stronger and more accurate as more and more frames are observed. We have evaluated the results against manually marked ground truth occluding edges.

It is seen that as long as the occluding edges get passed by the moving foreground objects during the learning period, the final occlusion map captures the edges. However, in some cases the moving objects do not pass the occlusion, for instance, the upper edge of a door may not overlap moving object's boundary due to its height.

In these cases, that edge of the occluding object will not appear in the occlusion map. Thus the results are sensitive to the activity in the scene during the learning period. A longer learning period will be required by the algorithm to ensure that all the occlusions in the scene are passed by the foreground objects. Moreover, depth information is not obtained from this method. The resulting binary map indicates spatial locations of occlusions, but we cannot determine the 3D structure of the occlusions using the intensity map.

Our motivation to learn occlusions is to improve tracking results. Once occluding edges are determined, for example, we can use only part of the appearance model when labeling objects that overlap the occluding edges. Or we may decide not to label the object when it is partially occluded. By predicting whether the moving object is occluded, we can thus improve the accuracy of tracking results.

Chapter 4: Video Metadata Simulation

4.1 Introduction

To address various problems in surveillance video analysis like tracking [24], object classification [31, 32], face detection and recognition [11, 12], systems often extract low level features from the objects of interest (see Figure 31). This data about the video, called metadata is then processed further. Some algorithms that involve learning using long video observations (e.g. learning entry/exit or stop zones in a scene, or other semantics like routes, junctions etc) require large data to gain statistical stability [4]. Applications like data mining also operate on very large video repositories [33, 34].

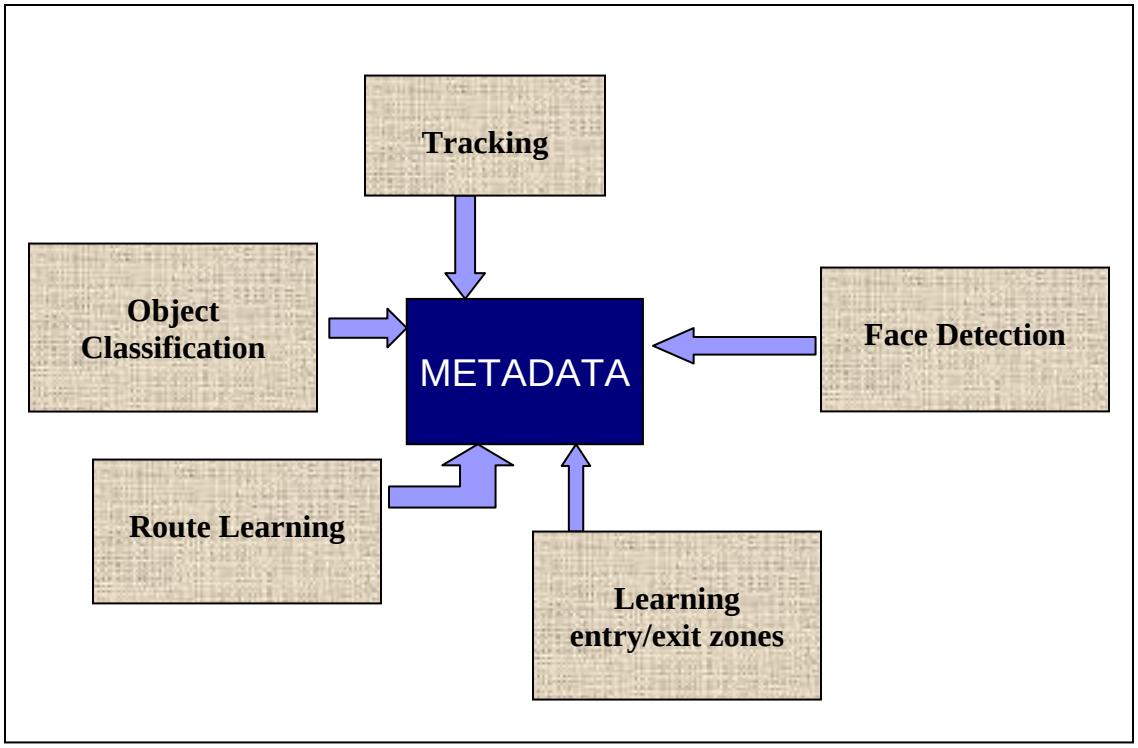


Figure 31: Examples of surveillance video analysis problems

The storage of large amounts of surveillance video data, however, is difficult and there are legal issues in monitoring public areas like airports, train stations or people in indoor environment like offices or shopping malls (at least in the United States !). Although these areas need strong security measures, recording videos for experimental purposes is prohibited in such places.

By noting that in most cases we need to store the metadata and not the full video, we propose to address this problem by "simulating" the video metadata for the given scene. We can then extract useful features from the simulated data for data mining rather than extracting the metadata from real video sequences. Simulated metadata will approximate the real data depending on the quality of simulations. The advantage of such an approach is that there is a reduced necessity to capture or store video sequences. Although the simulated metadata can be used as a test bench for evaluating the performance of a vision algorithm, the results may vary when the algorithm is tested with real surveillance videos. Care should be taken to incorporate all possible cases that occur in the given surveillance scenario in the simulation.

A similar approach has been used by Qureshi and Terzopoulos [35] in testing their surveillance system for wide field of view (FOV) and pan-tilt-zoom (PTZ cameras). The authors proposed a Virtual Vision approach to designing surveillance systems using a virtual train station environment populated by virtual pedestrians that perform various activities [35, 36]. The system in [37] is designed for New York Penn Station scenario where the objective is to schedule an array of FOV and PTZ cameras in such

a manner that each pedestrian is in the field of view of at least one camera during their stay in a designated area. To test the camera-scheduling algorithm, the authors use a virtual 3D model of Penn Station populated by virtual pedestrians to emulate the video streams generated by real surveillance cameras.

Our approach to simulate the video data is different from the 3D animation method used by Qureshi and Terzopolous. We propose using a real scene image captured by a single camera, simulation moving foreground blobs and superimposing these blobs on the scene image. Our methodology involves defining normal and abnormal patterns of people's movements in a scene with minimal user interaction. The proposed tool can then simulate random trajectories of moving objects based on the given patterns. We explain the proposed technique in detail in the next sections.

4.2 Approach

In a given scenario, the objects normally move along specific paths. For instance in a traffic scene, the moving objects (vehicles) take specific paths and in specific directions depending on which side of the road they are moving. At a traffic intersection, the trajectory followed by a vehicle lies within a predictable path. Similarly, consider another example of an office scenario, shown in Figure 32. Figure 32 (a) shows the original scene to be monitored by a surveillance camera. Figures 32 (b) to (g) show different possible paths (shown by green lines) that people can take as they move in the scene. To simulate metadata for such a scenario, a realistic assumption about moving objects will be that they chose one of these paths randomly

and move either in the forward or the backward direction. Thus we need a mechanism to simulate trajectories randomly for a given path.

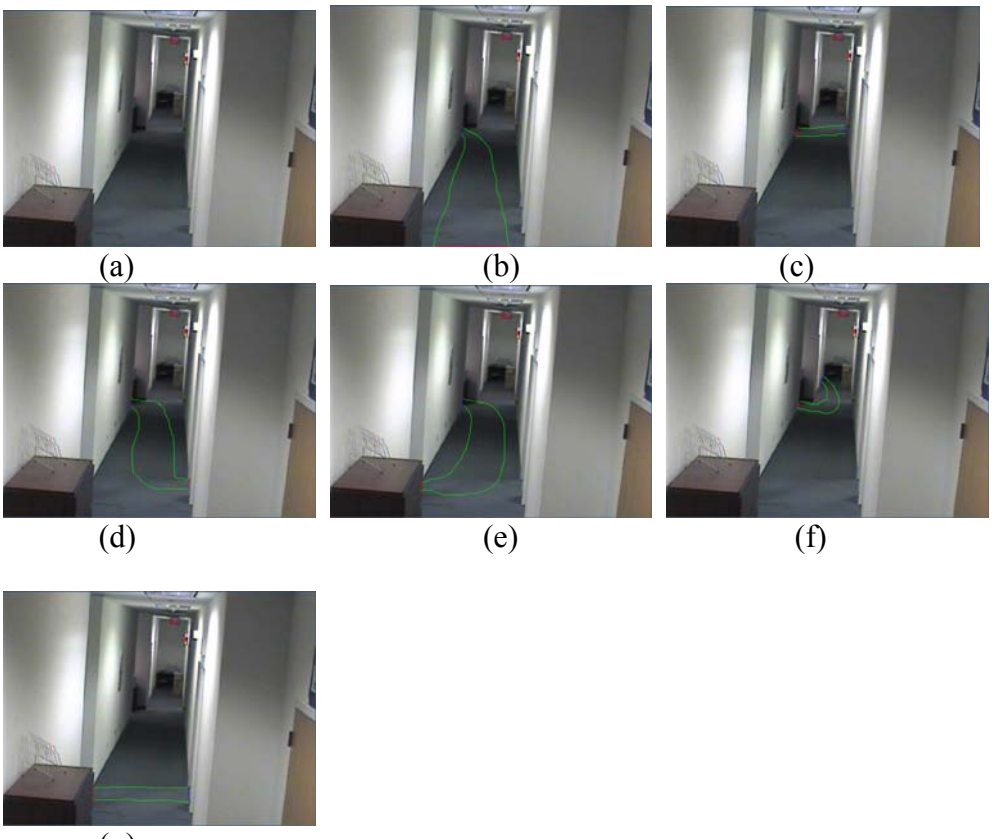


Figure 32: (a) Original scene, (b) to (g) possible trajectory paths for the given scene

4.2.1 Simulating object trajectories

To do this we select one of the *n* possible paths randomly (Figure 33(a)). The first step to simulate a random trajectory along this path is to pick a starting point X for the trajectory. Since objects are generally more likely to stay in the center of a path than the edges of the path, this is done according to a Gaussian distribution between the two end points (A and B) of the path boundary (Figure 33 (b)). Gaussian

distribution achieves a higher frequency of objects moving along the center of the path than the path edges. The Gaussian distribution is given by

$$\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{4.1}$$

Where mean μ is taken to be the center of the line segment AB and standard deviation σ is chosen such that 6σ is approximately equal to length (AB) (see Figure 33(b)). A point generated according to this distribution is translated to an image point by simple arithmetic. The two path boundaries (shown blue and red in Figure 33(c)) are sampled into equal number of points and these points are joined by line segments as shown in Figure 33(d). Based on the ratio that the starting point X makes along line segment AB, the remaining points of the trajectory are selected on the other line segments (Figure 33(d)). Since in practice, the trajectories of moving objects extracted by vision algorithms are jittery (not smooth), we add Gaussian noise to perturb the trajectory points. The resulting trajectory obtained by this algorithm is shown in Figure 33(f). It should be noted that if we select the other end of the path, then we obtain a trajectory for the object moving in reverse direction. Thus we can configure direction of motion (forward or backward) by selecting an appropriate end of the path. We have not considered the more complex cases like lane changes or an object stopping and turning around. It is, however, certainly possible to simulate these special cases. For instance, to model stop zones a trajectory point can be repeated for a number of times when the object lies in a manually specified stop zone. A slowing object can be simulated by increasing the duration between time stamps of trajectory points as the object approaches a stop zone. For a lane change, trajectory points need to be interpolated between the current trajectory point and nearest point in the new

lane. However, we have not considered these special cases in the example simulations.

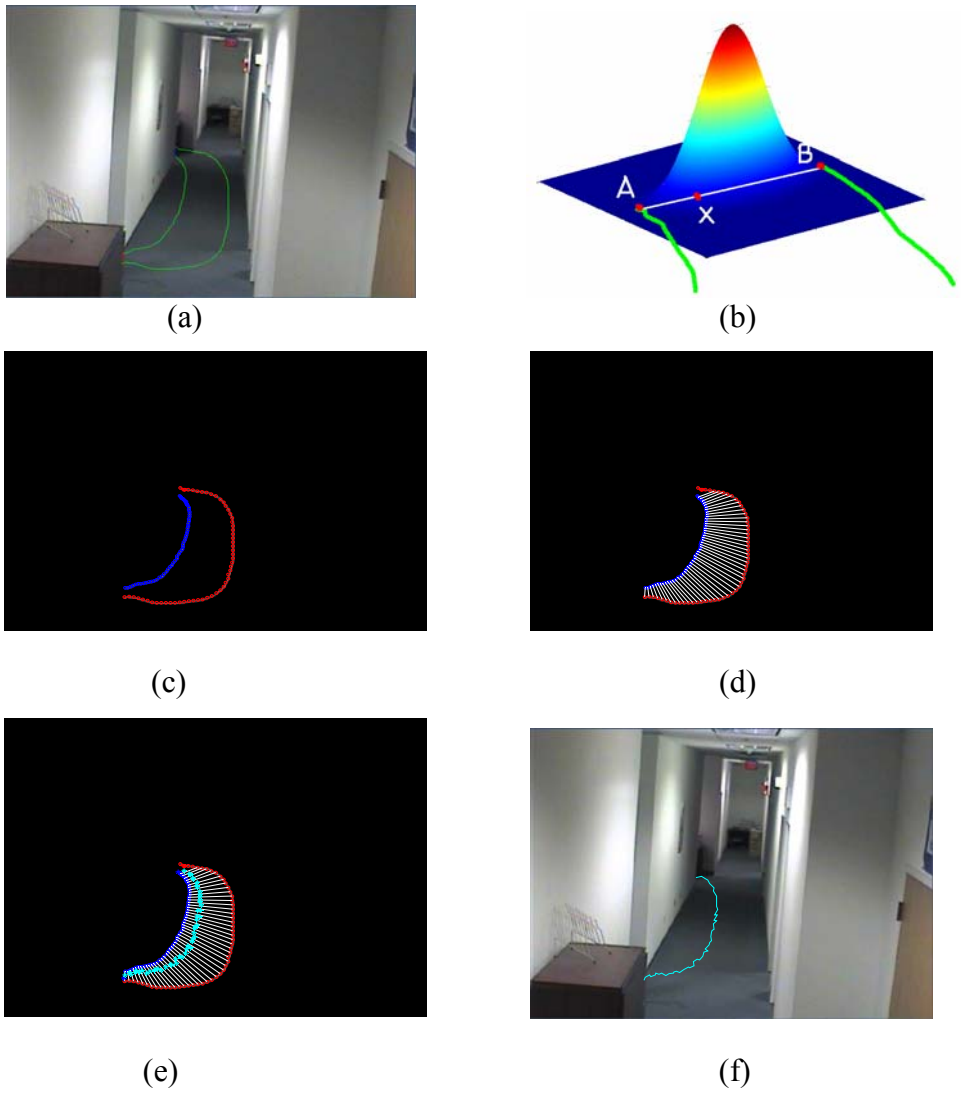
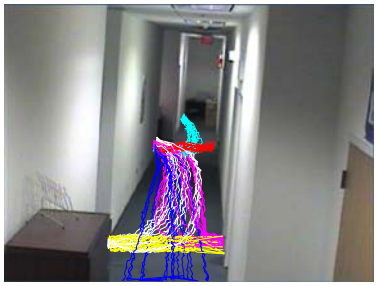


Figure 33 (a) Path image for a scene (b) Selecting start point using Gaussian distribution (c) Path boundaries (d) Sampled path boundaries (e) Trajectory points obtained by a Gaussian distribution (f) Final Trajectory

Using this technique it is possible to automatically generate a set of trajectories for the given scenario by choosing one of the *n* possible paths randomly and generating a random trajectory for that path. Figure 34 (a) shows 100 random trajectories

generated for a particular path in the office scenario of Figure 32. Different paths in a scene can in general have different usage frequencies. We assign probabilities to the different paths, so that some paths are more frequently used than others. Figures 34 (b) and (c) show simulations of 100 random trajectories by selecting random paths with equal probabilities and unequal probabilities respectively. It should be noted that the advantage of this simulation is that it is completely automatic and very fast. Also, we can apply this technique to any scenario.

(a)



(b)

(c)

Figure 34 (a) Simulated random trajectories for path 3 in given scene (b) 100 trajectories, all paths are equiprobable (c) 100 trajectories with probabilities assigned to paths, Blue: 50%, White – 20%, Yellow and Cyan – 10%, Red and Magenta – 5%

The configuration parameters for the trajectory simulation are the number of paths, the probability of path usage and speed of moving objects for a path. Change in speed of the moving object is easily achieved by varying the number of points in which path

boundaries are sampled. For higher speed, the number of sample points is lower, so that the blob moves a larger distance between two points on a trajectory. Thus we can have different zones in the scene where objects move with different speeds, e.g. pedestrians and vehicles.

4.2.2 Simulating moving foreground objects

Existing approaches in the literature use virtual environments to model a scene and 3D animated objects to model the moving objects [36, 37]. A virtual reality 3D environment is typically stored in VRML or X3D file formats and is not compatible with common video file formats like AVI or MPEG. For this reason, virtual reality models cannot be used to simulate moving foreground objects for a video sequence. Moreover, virtual reality models do not look realistic.

Our aim in simulating foreground objects is to provide a way of viewing the video sequence associated with a given metadata file. This will enable a visualization of the video sequence corresponding to given metadata or start and end time stamps. We propose an approach different from virtual reality to simulate moving objects given a scene. By using robust background subtraction algorithm, we extract foreground objects in a scene from sample video sequences (Figure 36). Next, we simulate the foreground object moving along the random trajectories by taking these blobs and superimposing them on the trajectories with suitable scaling (Figure 35 and 37). For instance, in a sequence where people are walking, a set of foreground blobs corresponding to the portion in which the person is walking are selected and scaled appropriately to fit the new scene image. We create a database of 60 blobs of a

walking human consisting of views from different angles. These consist of front, back and sideways views of moving humans. During the simulation, appropriate sets of blobs are selected based on the path chosen and the location of the blob in the image.

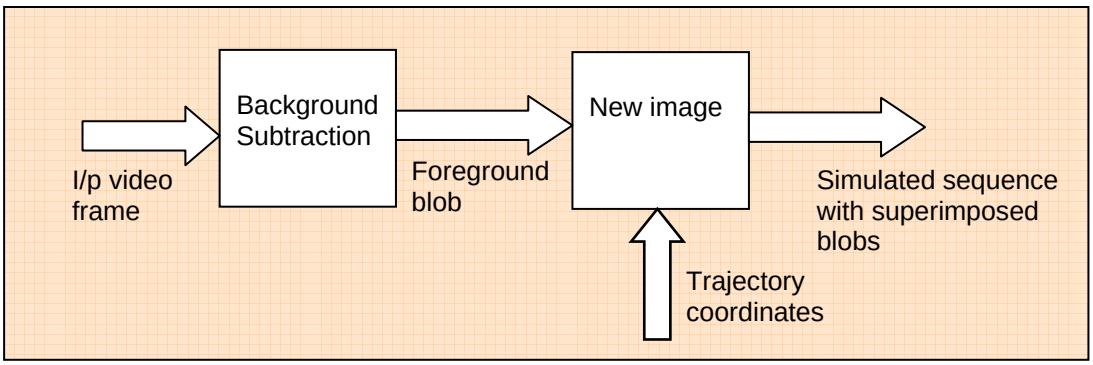


Figure 35: Simulating the foreground blob

Figure 36: Examples of blob libraries extracted by background subtraction ([38])

Figure 37: Example of simulated scene of a person walking

Ideally speaking the camera can have countless viewpoints and we need a larger database to cover all possible orientations of the blobs. However, current simulations include only four viewpoints front, back, left and right.

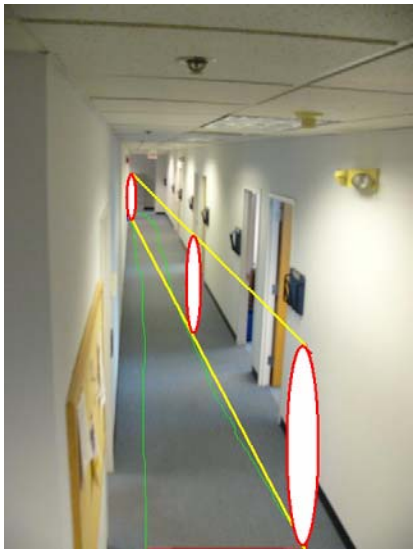


Figure 38: Scaling the blob based on its location

To accurately superimpose the blob at a trajectory point, we need to scale it appropriately. We assume the blob's expected height is known at the two ends of a path. At any intermediate point on a trajectory we linearly interpolate between these two heights to get the new scaling factor (Figure 38). Different paths can be programmed to have different speeds.

4.2.3 Metadata simulation

Different types of data is simulated using these moving foreground blobs

- Object's center of gravity
- Blob binary mask
- Object size
- Object velocity

- Object acceleration
- Object type (e.g. car, person etc)

To simulate surveillance scenario metadata we assume that an alarm is triggered by one of the many possible sources (motion detection, access control, manual recording, schedule recording etc.). The user specifies the timestamps between which the simulated metadata needs to be generated. The tool then generates random alarms every *t* minutes and associated with each alarm, a random trajectory using above technique is generated. Thus we simulate a hierarchy of metadata including alarm level metadata (Alarm type, timestamp, camera ID), Video level metadata (Video ID, Video file location, timestamp), Frame level metadata (Frame ID, video ID, fps, video resolution), and Blob level metadata (Blob ID, Blob size, centroid, speed, color, object type, binary mask). We provide a tool to automatically generate this metadata for the specified timestamps and write the simulated metadata into a text file.

4.3 Interface

The metadata simulation methodology is shown in Figure 39. We have implemented the proposed methodology in MATLAB. To use the interface, the base trajectories are created manually. These are simple to create and require minimal effort. They should encapsulate all possible regions where different objects can appear. Table 3 shows the configuration parameters that the user needs to specify to run the metadata simulation tool. The simulated metadata is written into text files.

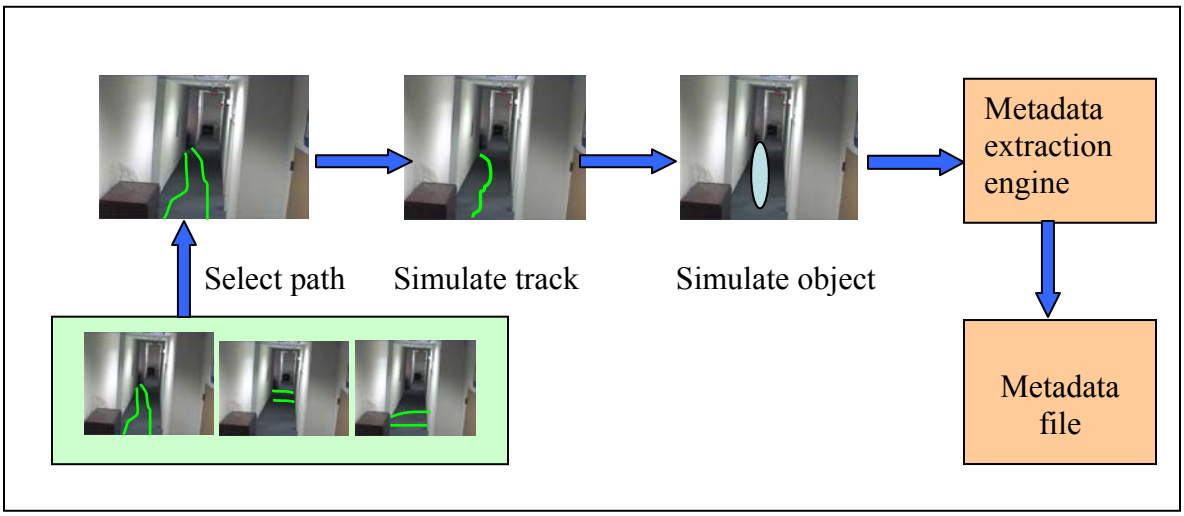


Figure 39: Metadata simulation methodology

No.	Parameter	Use
1	START_CLOCK	Starting clock time: [year month date hr min sec]
2	DATA_COLLECTION_DURATION	Duration in hrs for generating the data
3	ALARM_WAIT_TIME	One alarm is generated every ALARM_WAIT_TIME minutes
4	NUM_TRACK_IMAGES	Number of images available used to simulate different paths
5	PATH_SPEED	1 - slow, 2 - medium, 3 - fast
6	PERTURB_FLAG	0 - smooth track, 1- noisy track
7	DEFAULT_CAMERA_ID	Camera id number
8	DEFAULT_FPS	Frames per second value
9	DEFAULT_VIDEO_LOC	Location of the default camera video file
10	CAMERA_DEFAULT_IMAGE	Location and filename of the image to be used for the scene
11	IMAGEFILE_PATH	Path to the intermediate image files
12	BLOB_PATH	Path to the black and white blob image files
13	PROBABILISTIC_TRACKS	1 - assign probability to paths 0 - use equal probability of occurrence for all paths

Table 6: Configuration Parameters for Metadata Simulation

The tool is completely automated, given the scene and manually defined paths and no storage of video sequences required.

We can use the tool to generate metadata for different "scenarios". The simulated metadata can be used to test other low level vision algorithms, which would otherwise require collection of a large amount of video data. We give a few examples of using the simulated metadata to test vision algorithms in the next sections.

Using this tool it is also possible to generate large video metadata for "mining". The performance evaluation of different query and visualization tools of a mining system can be done using a simulated metadata set.

4.4 Application 1: Activity Map Visualization

We present a tool in this section that can be used to visualize the trajectory metadata generated by the simulation tool described in the previous sections. Viewing a large amount of recorded data in a meaningful way is a challenging problem for surveillance systems. One approach is to generate a summary of recorded sequences and determine which areas in the scene have more frequent activity (e.g. busy aisles in a supermarket) and which have limited or no activity (e.g. restricted access areas). Such a summary of the recorded video sequences can be visualized by the activity map of the scene. An activity is a 2D histogram of moving foreground objects superimposed on the original scene image. It is an intensity plot where brighter regions indicate higher level of activity and darker regions indicates less active areas. The activity map obtained from multiple cameras could generate summary

information for an observed site. We propose an activity map generation tool that will allow user to monitor the site usage summary across different time intervals. User can specify the following parameters:

a) Camera ID

b) Time interval

c) Time resolution

The system builds a density matrix by using the Video Content Meta Data to generate a matrix. This matrix has 3 dimensions, X,Y coordinates in the field of view of camera, and time intervals.

4.4.1 Density Matrix Generation

The Density Matrix represents the probability of detecting an object at the particular location (in the field of view of camera) at the particular time interval.

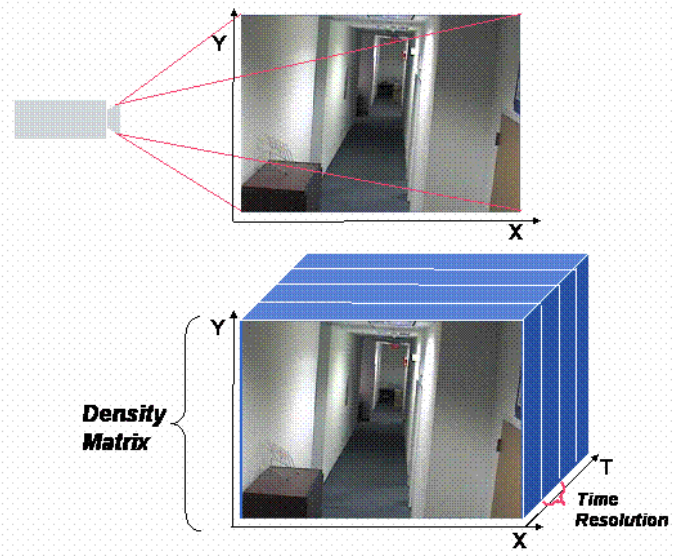


Figure 40: Density Matrix

For example, DM(X1,Y1,T2) gives the probability of object detection at location (X1,Y1) between T2begin and T2end. The resulting density matrix is superimposed

on camera default image based on the selected time point. This allows user to see the activity map for different times.

Figure 41 depicts the execution flow of density matrix generation operation.

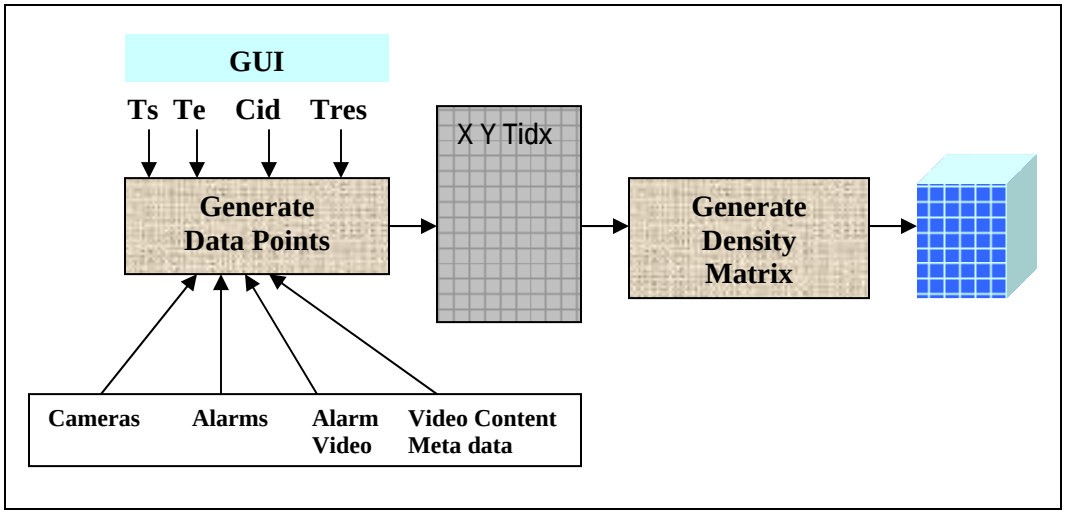


Figure 41: Density Matrix Generation Flow

The user provides the start time (Ts), end time (Te), camera identifier (Cid), and time resolution (Tres) parameters from GUI and starts the operation.

Generate Data Points operation uses the given criteria and the data files to generate data points. Each data point is represented by a tuple. The X and Y denote the location and Tidx denotes the time interval. There are (Te-Ts)/Tres+1 number of time intervals. Generate Density Matrix operation generates the 3D matrix data from the given data points.

The following algorithm generates the Density Matrix from given N center of blob data points.

```
GenerateDM(p[N],DM[][][])
{
      i=0;
      // accumulate the counts
      while (i<N) {
         DM(p[i].tidx,p[i].x,p[i].y)++;
         i++;
      }
      // Normalize
      for i=0 to Tidx {
       for j=xmin to xmax {
          for k=ymin to ymax {
             DM(i,j,k)/=N;
          }
       }
      }
      return;
}
```

The other possible approach is to generate the Density Matrix using the entire blob mask as the given N data points.

```
GenerateDM(p[N],DM[][][])
{
      i=0;
      total=0;
      // accumulate the counts
      while (i<N) {
        for each detected pixel (dp) in binary mask of p {
            DM(p[i].tidx,p[i].dp[j].x,p[i].dp[j].y)++;
            total++;
            i++;
        }
      }
      // Normalize
      for i=0 to Tidx {
         for j=xmin to xmax {
           for k=ymin to ymax {
               DM(i,j,k)/=total
           }
         }
      }
      return;
}
```

4.4.2 Generating Activity Map

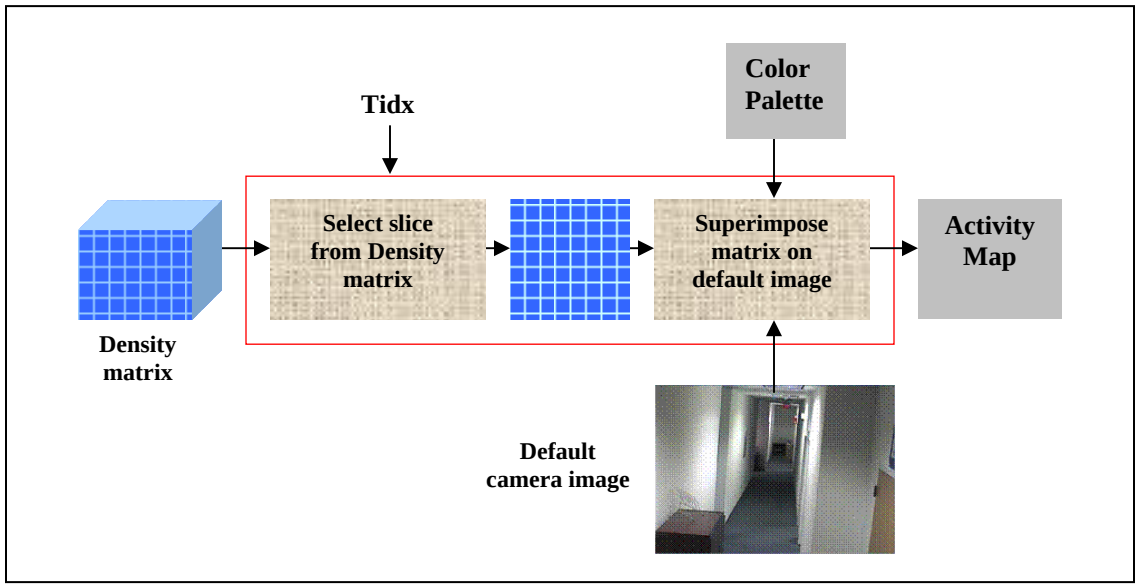


Figure 42: Activity Map Image Generation Flow

Figure 42 depicts the flow for generating Activity Map for a given time slice by using the density matrix and default camera image. Figure 43 shows the GUI for creating activity map.

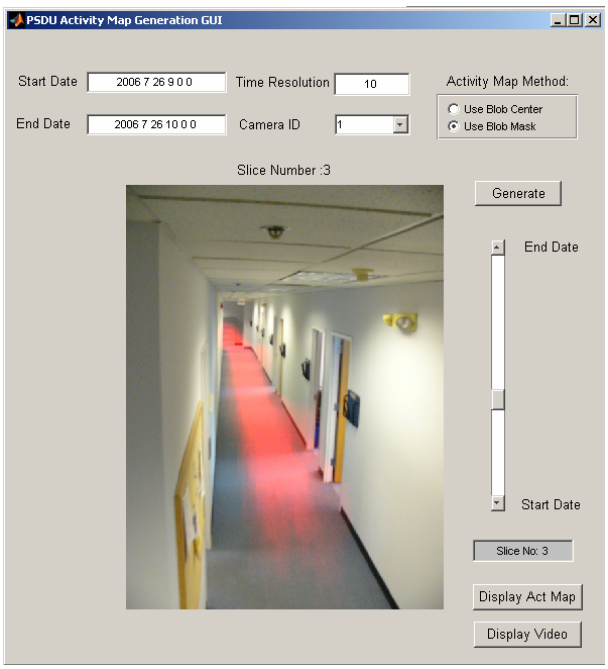


Figure 43: Example GUI for Activity Map Generation

The user can specify the following parameters as input to the GUI (Table 4):

Field	Type	Description
Start Date	YYYY-MMM-DD HH-MM	Start Date
End Date	YYYY-MMM-DD HH-MM	End Date
Camera	CHAR16	Camera Identifier
Time Resolution	INT32	Minutes
Generate	Button	Starts the activity map generation process
Display Act Map	Button	Display the generated activity map
Activity Map Method	Radio Button	1. Use blob center: the center of the blob is used to generate the 2D activity map. 2. Use blob mask : select the entire blob to generate the activity map histogram
Display Video	Button	Display the simulated video

Table 7: GUI input parameters

4.4.3 Advantages and Usage of Activity Map

The activity map can be used

a) in retail store applications for identifying frequently visited areas in the store,

b) to model the normal detection probabilities of observed site for different time intervals

c) to detect abnormal patterns that occur in low detection probability zones.

d) to view probability of detection for different types of objects in the scene e.g. people, vehicles etc (assuming the video metadata contains result of an object type classification module)

4.5 Application 2: Entry/Exit zone modeling and route learning

In this section we demonstrate the use of metadata simulation tool discussed in section 4 to generate test data for other vision algorithms namely entry/exit zone modeling. The entry/exit zone modeling algorithm is based on trajectory clustering from Makris and Ellis [4]. The metadata used by this algorithm is a set of trajectories gathered over a period of time.

Here we simulate the trajectories for a given scenario using our tool and demonstrate the working of trajectory clustering algorithm to learn entry/exit zones.

4.5.1 Algorithm and results

To cluster trajectories, we use the algorithm from Makris and Ellis [4], which learns entry/exit zones in the scene for trajectory clustering.

To learn entry/exit zones, only the start and end points of trajectories are required. The standard Expectation-Maximization (EM) algorithm is used to cluster these points. The EM algorithm gives parameters of a mixture of Gaussian distributions that best represent the clusters.

As EM is a supervised algorithm, it requires the number of clusters to be known beforehand. However, number of entry/exit zones in a surveillance scene is generally not known. So we start with a larger number of clusters and then eliminate some clusters based on their density of observation. Also, small clusters that are close enough to each other are merged into one cluster.

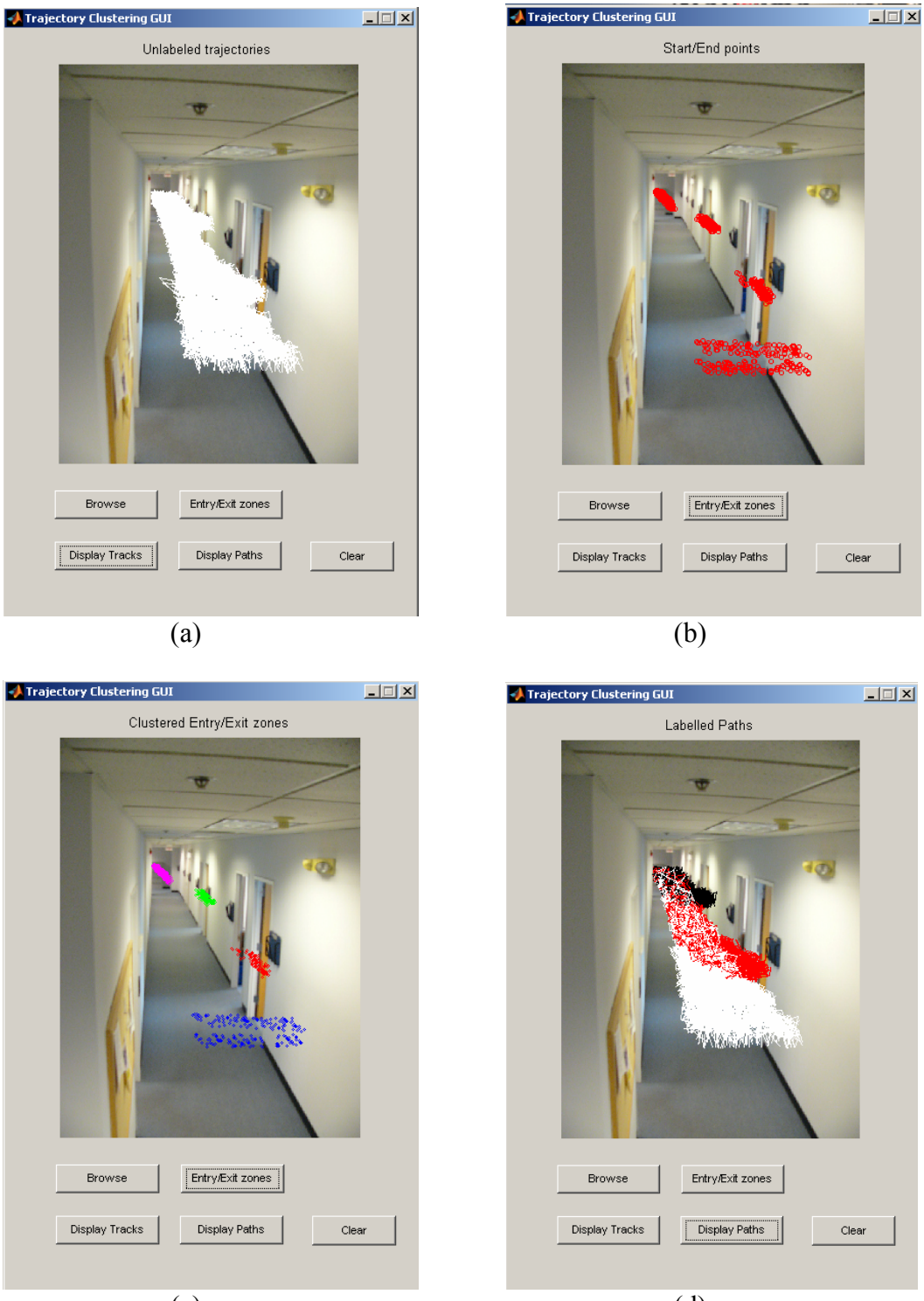


Figure 44: (a) Unlabeled trajectories (b) Start and end points (c) Clustered entry/exit zones using EM (d) Trajectories classified into learned routes

Once the entry/exit zones are identified, defining the possible paths in the scene is a straightforward combinatorial task. We can then assign labels to the trajectories and classify them into routes. Figure 44 shows the results of this algorithm for metadata simulated by the tool described in Section 4.2 and 4.3. Figure 44 (a) displays a plot of simulated trajectories read from the metadata file. Figure 44(b) shows the start and end points of these trajectories. The result of EM clustering is shown in Figure 44(c) where the four entry /exit zones are correctly clustered. These in turn imply presence of three possible routes in the scene. Figure 44(d) displays the trajectories classified into learnt routes.

4.6 Discussion

In this chapter we have presented a solution to address the problem of availability and storage of surveillance video data. By observing that in most computer vision algorithms, we need low-level metadata for processing, we propose simulation of the video metadata for a given scenario instead of using real video sequences. Once the normal and abnormal paths in the scene are defined, our tool simulates random trajectories of moving objects. A set of foreground blob libraries for different types of objects are created and stored beforehand. These are used to simulate the blob metadata as the object moves along the trajectory. By varying the configuration parameters, we simulate objects moving at various speeds along different paths in the scene. The advantage of this technique is that it can be applied to any scenario. Only one image of the scenario is required and there is a reduced need to record or store the video sequences for metadata extraction.

We have illustrated the technique for an office scenario. Also, we have shown two example applications of the metadata simulation tool - the activity map visualization tool and the entry/exit zone modeling and route-learning algorithm. For both these applications, the metadata simulation tool is used to generate the trajectories.

Although the metadata simulation technique can be very useful, the current version has some limitations. We currently support simulation of trajectories of a single object at a given timestamp. The algorithm needs to be enhanced to include multiple trajectories at the same time. This will be required for simulating more complex scenarios.

The accuracy of simulated metadata depends on the set of foreground blob images i.e. the blob libraries. To include different poses and shapes and behavior patterns, we need extensive collection of these blob libraries. Another issue with this technique is the merging of object with the background. For the purpose of visualization, superimposition of the blob is acceptable, however, if the blob boundary is not smooth, the 'simulation' will be more noticeable and the scene will look less realistic. Blob merging will be difficult if there are sharp color and contrast differences between the object and background e.g. under lighting changes.

As discussed, one application of simulated metadata is to test the functionality of the computer vision algorithms. It should be noted that results obtained by testing these

algorithms on real video sequences would be certainly different than those for simulated metadata. Additional practical problems like occlusion, lighting changes will arise when dealing with real videos. Our technique of using simulated metadata provides a quick and effective way to generate data for testing the basic functionality, at the cost of possible reduced accuracy of results.

Chapter 5: Progressive Threat Detection Modeling

5.1 Introduction

Video surveillance systems that record non-stop video footage 24 hours a day can have prohibitively large storage requirements. Likewise, continuous monitoring requires an operator to be viewing the video continuously. During most of the video, however, there may be little or no activity in the scene. One approach to reduce the recording or monitoring requirements is to either record or monitor video footage only upon receiving an alarm. An alarm can be triggered in one of many possible ways – by activity (certain modern security cameras are capable of detecting motion in the scene), with access card control (e.g. when someone unlocks a door using access card and enters the monitored room), by pre-scheduling, with sensor-generated alarms or manually by an operator in case of emergency. Without loss of generality, we will assume all alarm based recording also encompasses alarm based monitoring. An alarm based system preserves stores video footage starting a few seconds prior to the trigger and a few seconds after the alarm has been turned off, for instance a motion based alarm will be turned off when the object has left the scene. Such discrete notifications reduce the storage requirement considerably and ensure that important data is not missed. A surveillance system that stores continuous video may do so for a limited amount of time and compress old video data (say anything more than a week ago) by storing just segments of video that correspond to alarms.

An interesting problem associated with the alarm-based recording is to automatically determine the moment at which the recording should start i.e. when to trigger the alarm. This decision should be based on the video content. Background subtraction for motion detection is one simple approach. However, it will be useful if the alarm-triggering algorithm can also determine the level of threat the event corresponds to. For example, an alarm triggered by a person entering a restricted area may be higher threat than a motion detection alarm triggered by a person walking in a room. In this chapter we focus on the problem of analyzing video data to determine a threat level and then automatically trigger an alarm depending on the threat level.

Detection of abnormal behavior can be used as a decision factor in determining threat level. Some examples of behaviors that may constitute a threat include - a person entering a restricted area; activities like running, loitering, theft, fighting, abandoning baggage; vandalism; or abnormal trajectories that do not confirm to a regular pattern. For example, a vehicle going in wrong traffic lane or a person moving in opposite direction of flow of the crowd.

Activity recognition and abnormal behavior detection in videos has been an active area of research. Johnson and Hogg [39] presented a neural network based model to learn the probability distribution functions of object trajectories from video sequences. The normal trajectories are modeled by a flow vector to recognize atypical movements. Rao and Sastry [40] used a maximum likelihood framework to characterize normal activities in a scene based on motion trajectories. They tested

their algorithm for normal flow of traffic with abnormal events being a pedestrian trajectory and vehicle trajectory in opposite direction to the flow. Other examples of related works are detection of higher level activities. Cuntoor and Chellappa have developed a model to characterize high level activities by a sequence of key frames rather than dominant characteristics over the entire video sequence [41]. Detection of specific suspicious activities like abandoning a package has been studied in [42, 43].

In our approach we define the threat level using trajectory data of the moving object. In particular, we focus on the object's motion trajectory relative to a "target". We detect events where an object is in close proximity, or is loitering near a target, or an object is approaching a target. The faster the object approaches the target, the higher the threat level. A person running toward the target will be considered a higher threat than a person just walking towards the target. Similarly, a person walking away from the target is not a threat, whereas a person moving toward the target may be a threat. Thus the detection is based on distance of the object from the specified target, its trajectory data and the direction or rate at which the object approaches the target. We do not incorporate higher level behavior detection like a person abandoning an object, or crowd gathering near dispersing away from an object or abnormal trajectories that do not confirm to the regular flow.

For prevention of a security breach, an intelligent systems needs to know the progression of threat level. Our threat level scoring engine that analyzes the trajectory

and gives real time scoring based on the learned model. Visual alarm notification is generated when a threshold is reached.

5.2 Overview

A simple implementation of a threat detection model would be to use the activity map generated by monitoring trajectory data over a period of time. The activity map gives a 2D histogram or probability distribution of activity in the scene. By comparing the new trajectory points to the probability of occurrence values from the activity map, we determine a threat level, assuming that the threat is only a function of the current position and the previous activity in that area. For example, consider the activity map shown in Figure 45 for a shopping mall scenario. Figure 46 shows a trajectory of a burglar who breaks into the first shop on the left in the image. The trajectory drawn enters the shop through the glass where the mannequins are placed. The threat detection score engine result is shown below in Figure 47, where green color denotes low threat, orange medium threat and red denotes high threat level.

Clearly there are other factors, such as where potential targets are located, the direction and speed of the threat, duration of the threat, and even properties of the threat (size, class, and history, for example). We extend the threat detection by including enhancements to the threat level modeling. The mere presence of an object in an area where there was no activity previously does not establish it as a threat. Our extensions focus the relative proximity and activities of a potential threat with respect to the target.

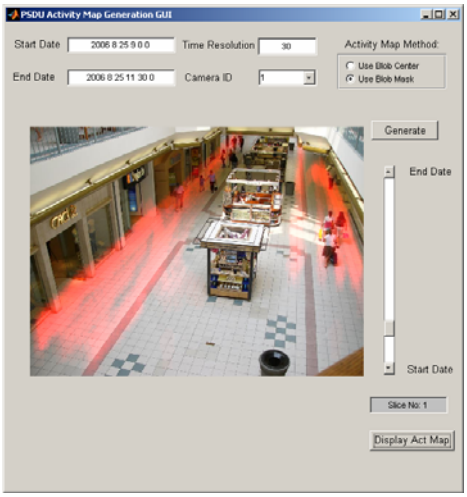


Figure 45: Activity map for shopping mall scenario

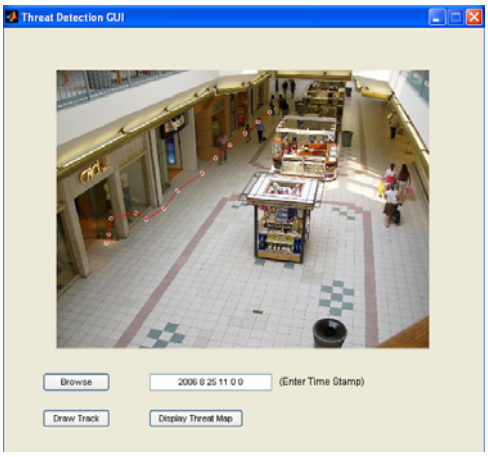


Figure 46: Sample Trajectory

Figure 47: Threat map

1. We first consider the location of objects of interest in the scene (called targets). Threat modeling with respect to these targets will be more meaningful. This is very common in video analysis where the proximity of two objects has meaning (two people may be considered meeting if they have some relative proximity over some minimal amount of time).
2. We use a decay factor to model the distance function around a potential target. A decay factor can be used to dampen the computed distance to vary the threat level based on object's spatial location relative to the target. For example, a decay

factor may be low for a cash register, and higher for a person. A threat does not need to be as close to a person to be identified a threat as they would to a cash register.

3. We incorporate directionality of the moving object with respect to the target. An approaching object is more likely to be a threat than an object moving away from the target, even if they are at the same distance from the target.

We consider speed and the duration of a potential threat in the scene. Someone loitering around a target may increase the threat. Likewise someone running to the object would increase it as a threat. A potential threat simply walking by may be ignored. This is done by keeping track of time duration for which an object is present within a distance threshold of the target.

5.3 Threat Measures

Models for various threat measures based on above threat patterns are discussed in this section. The overall model is a linear combination of these factors.

5.3.1 "Proximity Measure" using damping factor and activity map

The proximity component of the threat is related to the distance from the target. A person that is closer to the target is a higher threat. The proximity threat decreases with the object's distance from the target. By observing trajectories of objects moving in the scene over a period of time, we can get a measure of activity in the region. An activity map is a 2D histogram of the given scenario generated based on trajectories monitored over a period of time. Larger values in the activity map imply the region is busier than other regions with low value of activity. The activity map value is also taken into consideration when calculating the threat factor. For a region where there is

more activity, *(1.0 - Activity)* value will be a small number and will have an effect of reducing the proximity threat. We use *(1.0 - NormalizedDist),* so that the proximity threat increases as the object gets closer to the target.

$$ProximityThreat = (1.0 - NormalizedDist) * DampingFactor * (1.0 - Activity) \quad (5.1)$$

where,

$$NormalizedDist = \frac{\text{(Euclidean distance between target and object)}}{\text{(max possible euclidean distance between target and object)}}$$

Note that we have not converted distances to physical coordinates but knowledge of the scene can provide a linear conversion to world coordinates.

5.3.2 "Approach Measure" using direction and speed

An object that is approaching the target will be considered a greater threat than an object heading away from the target, independent of the distance away. If the object approaches the target and then moves away, the threat level should initially increase and then decrease. Furthermore, the rate at which the object is approaching a target also affects the threat level and can be measured as the component of the moving object's velocity toward the target. If the object is approaching the target, this component is positive. For an object moving away from the target, the component would be negative. This information will be used to determine the threat by an approaching object as:

$$ApproachThreat = \left\| \frac{dx}{dt} i + \frac{dy}{dt} j \right\| \cos\theta \tag{5.2}$$

where

(dx/dt) is the velocity component in x direction,

(dy/dt) is the velocity component in y direction,

θ is the angle in between direction of velocity and line joining target and the object.

5.3.3 "Loiter Measure"

An object that stays within the proximity of the target for a sufficiently long time may also be classified as a threat. To detect loitering, we first determine whether the object is staying in a particular region for a predetermined minimum time. If yes, we increment the time counter to store the loitering time. To determine loitering threat factor, for each point on the trajectory, we compute

$$LoiterRatio = \frac{N_r}{N} \tag{5.3}$$

where,

Nr is the number of points that are within a radius R of the current point.

N is the LOITERING_ORDER. This is number of past samples used to determine if there is loitering.

If *LoiterRatio* is greater than a threshold, it means the object has been resident in the same region for too long and we start incrementing a counter. The Loitering factor is defined as the log of this count, making the loitering constant grow gradually as the object continues loitering. We decrement the loitering time counter if the object starts moving, which means the *LogRatio* is smaller than the chosen threshold.

Pseudocode

```
LoiterRatio = Nr/N

if (LoiterRatio > LOITER_RATIO_THRESHOLD)
        // increase loitering factor by 1
        LoiteringFactor(i) = max(LoiteringFactor) + 1;
else
        // reduce loitering factor by 1
        LoiteringFactor(i) = max(max(LoiteringFactor) - 1,0);
end

LoiterFactor = log(LoiterFactor)/K, where K is a constant.
```

5.3.4 Total Threat measure

The net threat factor is computed as a combination of the above measures. To summarize, the metadata extracted to determine threat level consists of the following:

(a) For trajectory points:

1. Velocity
2. Normalized distance from the target
3. Damping factor
4. Activity map value
5. Direction (approaching target or moving away)

(b) At object level

1. Loitering time duration

$$TotalThreatVal = max(0, w1 * ProximityThreat + w2 * ApproachingThreat + w3 * LoiterThreat) \quad (5.4)$$

w1, w2, w3 are weighting factors determined empirically for different operational scenarios.

Thresholds to determine threat level are based on final threat value (Table 5):

Threat Value	Threat Level	Event
< T1	None (GREEN)	-
< T2	Low (YELLOW)	Object loitering in proximity of the target
< T3	Medium/intermediate (ORANGE)	Object approaching the target
Else	High/critical (RED)	Object very close to the target OR Object running towards the target

Table 8: Threat Level Thresholds and Corresponding Events

The simulated trajectory data is used to generate activity map for the given scene for a given time duration. The input to the threat detection model is the activity map, test trajectory, time stamp for the test trajectory and target location and thresholds used in computing various threat measures i.e. loitering duration, loitering order (number of past samples used), proximity threshold and damping type (linear or Gaussian).

5.4 Example scenarios:

Without loss of generality, we present the example results for our threat modeling approach for simulated trajectory metadata from Chapter 4, for a shopping mall scenario. The metadata simulation tool discussed earlier is used to generate an activity map for the mall scenario. A target is specified by drawing a bounding box on the input image.

Case 1: Object approaching target

Figure 48 shows the result for threat detection for an object that approaches the target. The trajectory is shown in (a). Resulting threat color map is shown in (b). The plot of total threat measure against frame number is shown in (c). Finally the

individual threat measures are shown in (d). The Proximity threat is seen to increase as the object approaches the target. The approach threat has position value as the object comes closer to the target, but becomes negative as the object moves away from the target. The loitering threat is zero throughout for this case.

Case 2: Object passing by the target

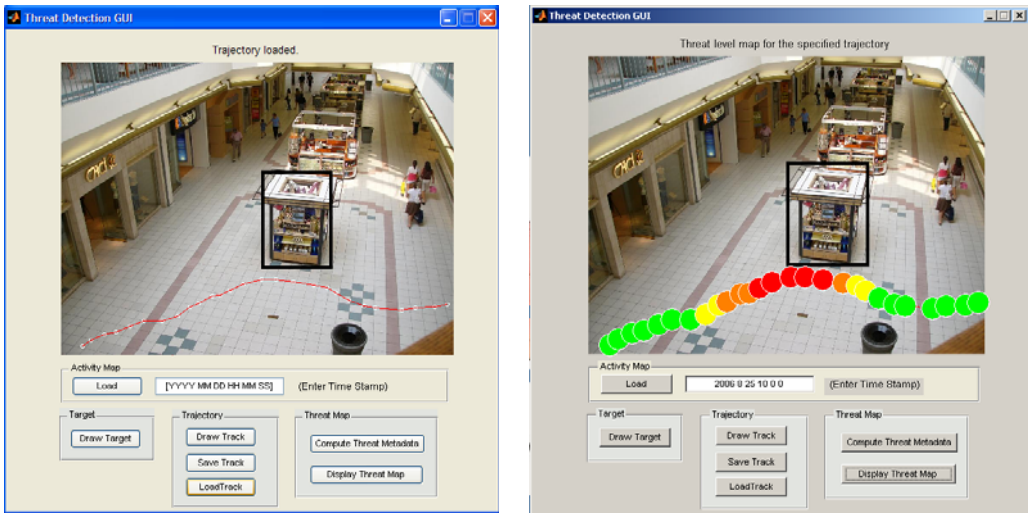
Figure 49 shows the result for threat detection for an object that passes by the target. The trajectory is shown in (a), resulting threat color map is shown in (b). The plot of total threat measure against frame number is shown in (c). Finally the individual threat measures are shown in (d). All the individual threat measures are low in this case and also the final threat measure is low. Consequently, the color map shows all trajectory points as green.

Case 3: Object loitering near target

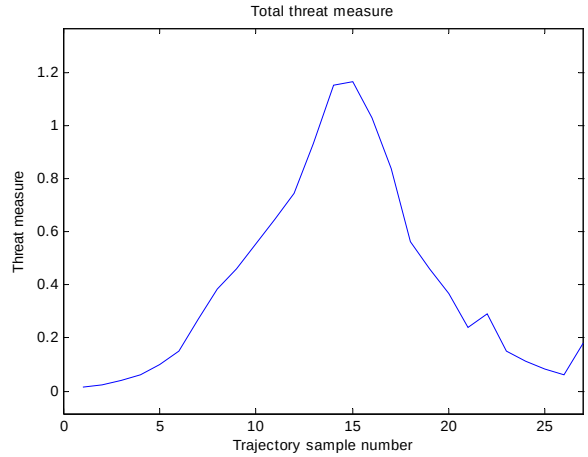
Figure 50 shows the result for threat detection for an object that loiters near the target. The trajectory is shown in (a), the resulting threat color map is shown in (b), the plot of total threat measure against frame number is shown in (c) and finally the individual threat measures are shown in (d). Loitering threat factor is seen to increase as time passes by. The overall threat measure grows too, as expected.

Case 4: Object loitering near target and then going away

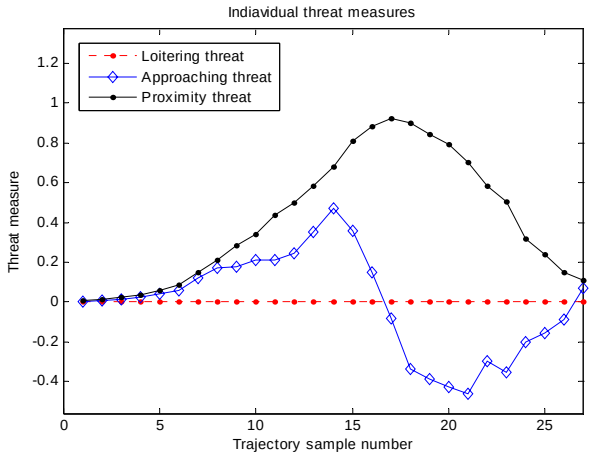
Figure 51 shows the result for threat detection for an object that approaches the target. In this case, the result is similar to case 3 for loitering. As the object moves away from the target, the loitering threat factor reduces gradually, as does the overall threat measure.



(a) (b)

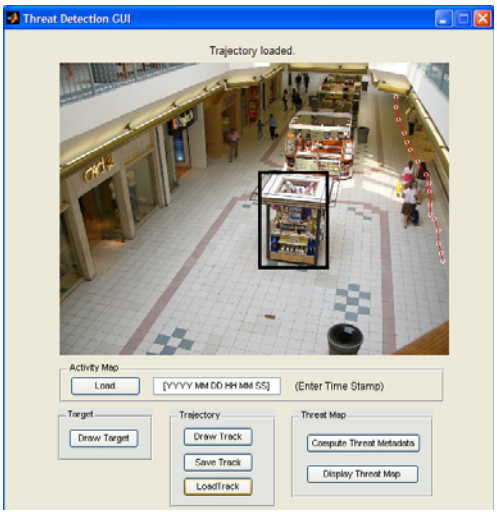


(c)

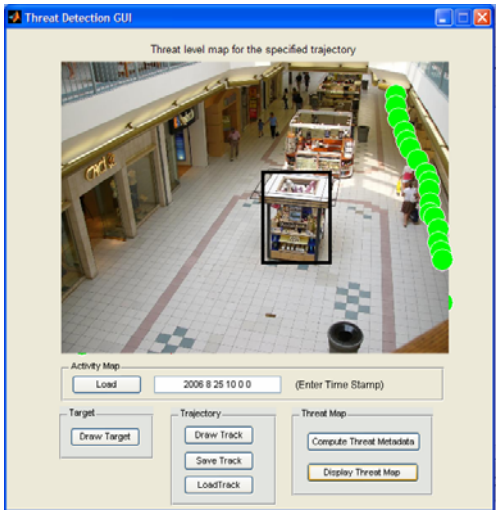


(d)

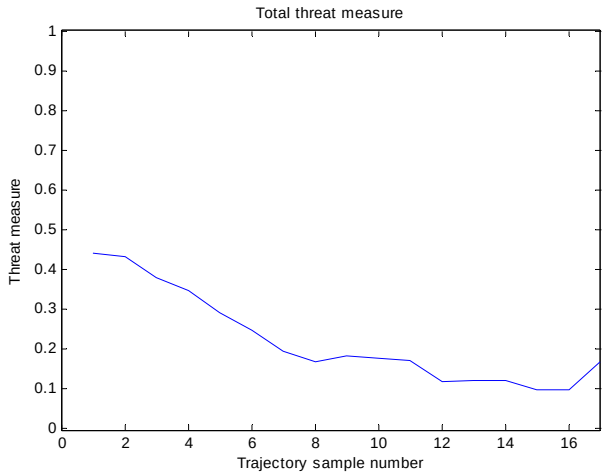
Figure 48: Case 1 - Threat measure for an object approaching target (a) Total threat measure (b) Loitering measure, approaching measure and proximity measure shown separately



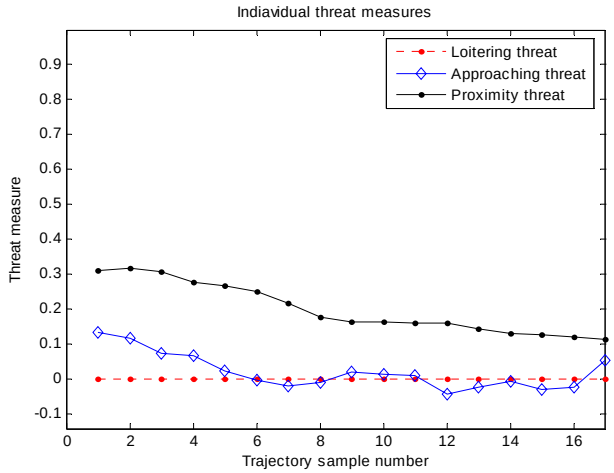
(a)



(b)

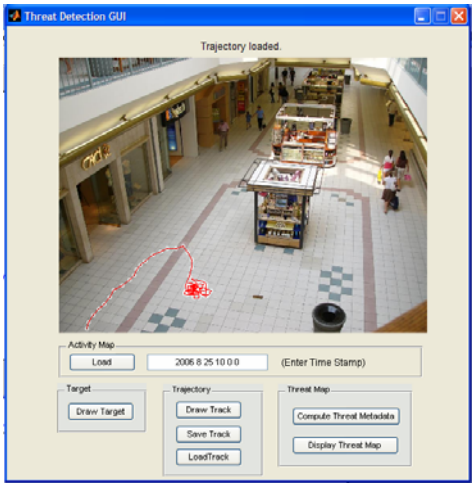


(c)



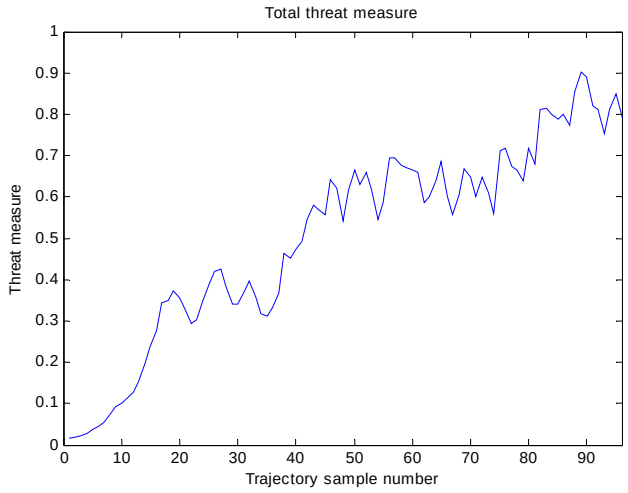
(d)

Figure 49: Case 2- Object passing by target (a) Total threat measure (b) Loitering measure, approaching measure and proximity measure shown separately

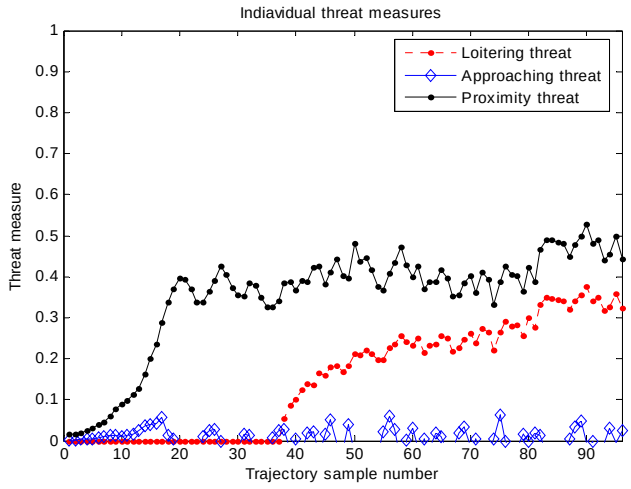


(a)

(b)

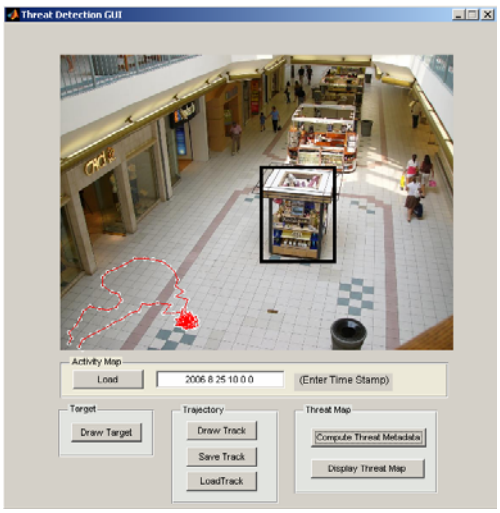


(c)



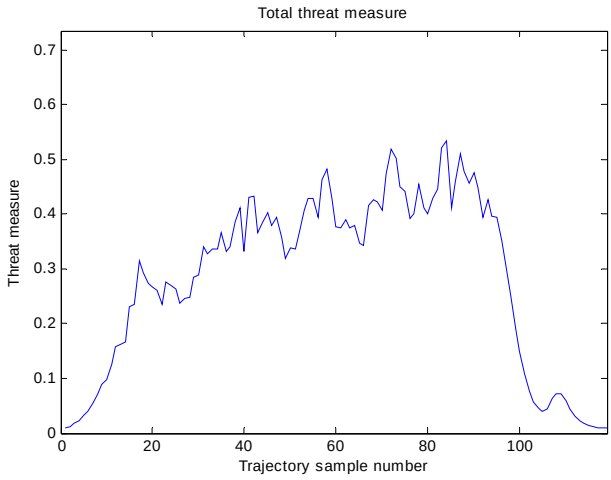
(d)

Figure 50: Case 3 - Object loitering around the target (a) Total threat measure (b) Loiter threat measure, Approach threat measure and Proximity threat measure shown separately

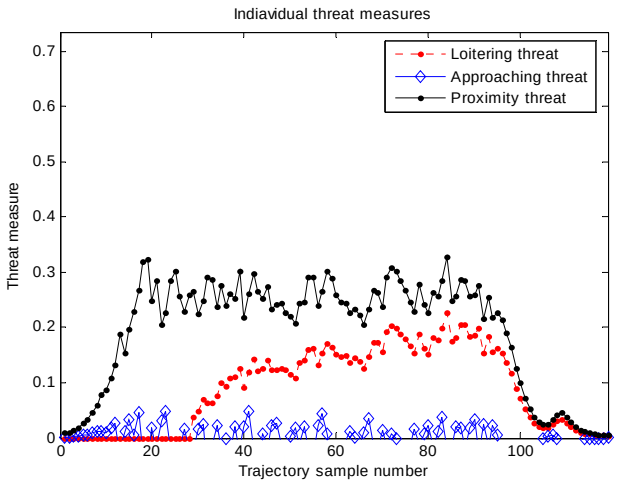


(a)

(b)

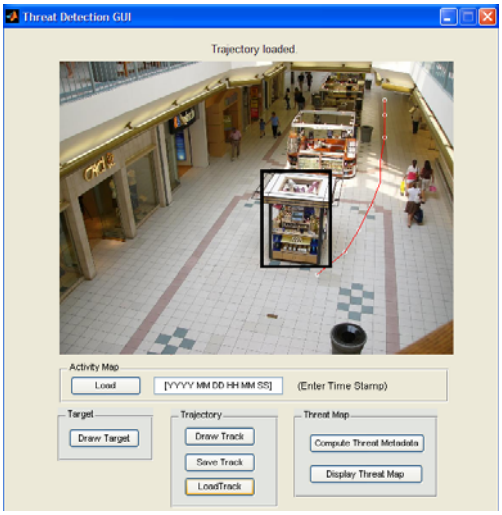


(c)

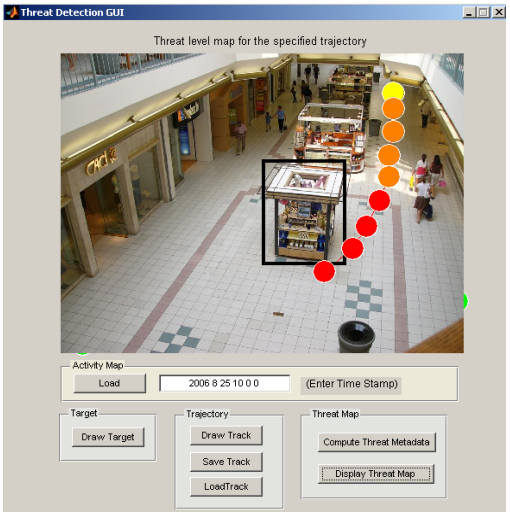


(d)

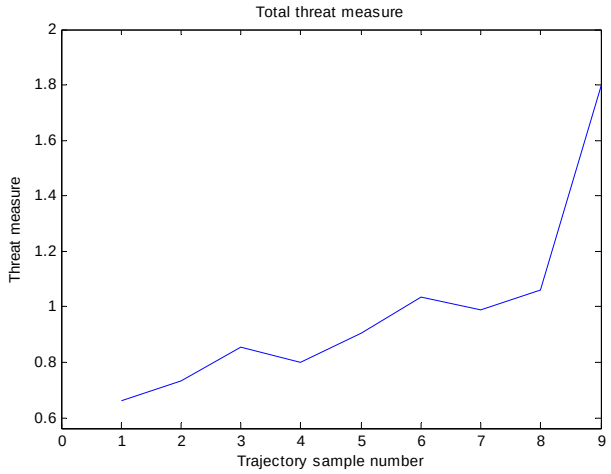
Figure 51: Case 4 - Object loitering and then leaving (a) Total threat measure (b) Loiter threat measure, Approach threat measure and Proximity threat measure shown separately



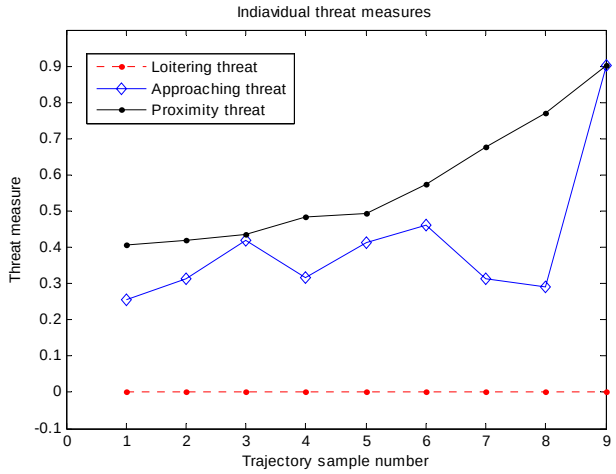
(a)



(b)



(c)



(d)

Figure 52: Case 5- Object running towards target (a) Total threat measure (b) Loitering measure, approaching measure and proximity measure shown separately

Case 5: Object running towards the target

Figure 52 shows the result for threat detection for an object that approaches the target. The trajectory is shown in (a). Trajectory points are spread apart more than the previous trajectories, indicating an object moving with greater speed. Here the velocity component in the direction of the target is high and as the object approaches the target. Hence the combined threat measure is high as expected.

5.5 Discussion

We have presented a threat model based on trajectory metadata of a moving object. In our model, we estimate threat relative to a target. Specifically, we consider threat to be a measure of object's proximity to the target, approach speed and loitering factor. The model can be extended by detecting high level complex events like theft, a baggage being abandoned in public place, abnormal crowd movement like gathering near an object or moving away from a possibly harmful object. Current model incorporates a single target. The model can be enhanced to include more than one target.

Another issue that needs to be addressed is the perspective distortion present in the camera-captured images. To accurately determine the distance between two locations in an image, say two people, we need to get their location on the ground plane. This involves estimating the ground plane homography and removal of perspective distortion, so that distance between two points on the ground plane is accurate with a scaling factor.

Chapter 6: Summary and Future Work

In this Thesis , problems central to the automated analysis of surveillance videos were explored and we proposed advanced metadata models to address them. Our goal was to automate the task of analyzing the surveillance video by developing intelligent tools to alert the operators when unusual events occur.

We have explored the problem of building a snapshot of a person from a video sequence. Given such a capability, systems can rapidly build a set of candidate suspects for the operator to view and allow them to choose suspects to follow. In the proposed method for snapshot selection, we first perform skin detection to locate the possible face blobs in the objects detected by background subtraction. Then an eigenface based frontal face detection is performed on these skin blobs. The frames are finally ranked in the order of their "facedness". Snapshoting results for several video sequences were presented. The performance of the algorithm depends upon the accuracy of frontal face detection engine. As a future work, the algorithm can be extended to a multiple camera scenario, where tracking results and appearance models can be used to correlate trajectories of the same object across multiple cameras. The snapshot summaries from these multiple views can be combined and ranked.

We have proposed a simple and effective technique to learn static occlusions in a scene using long observations from the surveillance videos. By accumulating the overlap of boundaries of an object across consecutive frames, we generate an

occlusion map for the scene. We described two methods to obtain the occlusion map and results are presented for indoor office scenarios. The occlusion model can be extended and applied to tracking algorithms to improve results. By adding depth information, we can also use this model to learn structural information about static occluding objects in a scene like doors and desks.

Another contribution was an automated metadata simulation technique. There are legal and storage issues in obtaining video sequences for surveillance scenarios. By noting that most low level vision algorithms require metadata to be extracted from the video, we have developed a tool to directly simulate the metadata given the surveillance scenario. Our tool has the advantage that it requires minimal user interaction and is applicable to any scenario. We have illustrated our approach for an office scenario. To show its usefulness, we present two application of the metadata simulation technique – activity map analysis and learning of entry/exit zones or routes in the scene. As these examples show, simulated metadata can be used to test the functionality of computer vision algorithms. It should be noted that results obtained by testing these algorithms on real video sequences may be different than those for simulated metadata. Practical problems like occlusion and lighting changes arise when dealing with real videos. Our technique of using simulated metadata provides a quick and effective way to generate data for testing the basic functionality, at the possible cost of reduced accuracy of results. The technique can be enhanced in several ways. For instance, the current version of the tool can simulate a single trajectory at a given time stamp. In future, the metadata simulation tool can be

extended by including multiple trajectories i.e. more than one object at the same time stamp. This will be required for simulating complex but more realistic scenarios. Moreover, the accuracy of simulated metadata depends on the set of foreground blob images i.e. (blob libraries). To include different poses and shapes and behavior patterns, extensive collection of these blob libraries will be needed. Another possible enhancement is visualization of simulated blobs. For this smooth merging of foreground object with the background is required. In the current version, we have superimposed the simulated blob directly on the scene image. To make the animated video look more realistic, blob merging should handle boundary, color and contrast differences between the object and the background e.g. under lighting changes.

Another contribution of our work was to facilitate alarm-based recording of videos based on a threat model. We proposed a threat level scoring engine that analyses the trajectory metadata of foreground objects and detects abnormal behavior. The threat model includes proximity of the moving object to a target, its direction of motion, a measure of loitering and a previously learned activity map. The scoring engine generates a visual notification to the operator allowing a progressive monitoring of threat levels. Once the threat level exceeds a predetermined threshold, alarm based recording can be triggered. We have shown the threat detection results for several different behavioral patterns of people moving in a mall. As future enhancement, we can use higher level activity detection like theft or abnormal crowd behavior to improve threat modeling. We can also convert the distances between two objects in

the image into world coordinates on the ground plane by using ground plane homography information in the scene.

To summarize, we have explored the problems of snapshot selection, occlusion modeling, metadata simulation and progressive threat detection modeling in surveillance videos. These metadata modeling techniques will automate the task of analyzing surveillance videos and help the security operators to control and manipulate the videos in meaningful ways.

Bibliography

1. K. Kim, T. Chalindabhongse, D. Harwood and L. Davis. Background Modeling and Subtraction by Codebook Construction, *IEEE International Conference on Image Processing (ICIP)*, Vol. 5, 3061- 3064, 2004
2. B. Manjunath, J.Ohm, V. Vinod, and A. Yamada, Color and Texture Descriptors. *IEEE Trans. Circuits and Systems for Video Technology*, *11* (6). 703-715, 2001
3. M. Bober, MPEG-7 Visual Shape Descriptors. *IEEE Trans. Circuits and Systems for Video Technology*, *11* (6). 716–719, 2001
4. D. Makris, T. Ellis, Automatic Learning of an Activity-Based Semantic Scene Model. in *IEEE International Conference on Advanced Video and Signal Based Surveillance*, 183-188, 2003
5. V. Vinod, H. Murase. Video shot analysis using efficient multiple object tracking *IEEE Int. Conference on Multimedia Computing and System*, 501-508, 1997
6. A. Divakaran, K. Peker, R. Radharkishnan, Z. Xiong and R.Cabasson. Video Summarization Using MPEG-7 Motion Activity and Audio Descriptors. in *Video Mining* ed. A. Rosenfeld, D. Doermann, D. DeMenthon., Kluwer Academic Publishers, 2003
7. D. DeMenthon, V. Kobla and D. Doermann, Video Summarization by Curve Simplification. in *Proc. ACM international conference on Multimedia*, (Bristol, United Kingdom), 211 – 218, 1998

8. Y. Gong, X. Liu, Video summarization using singular value decomposition. in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 174-180, 2000

9. F. Porikli. Multi-Camera Surveillance: Object-Based Summarization Approach *MERL Technical Report TR2003-145,* 2004

10. L. Latecki, W. Xiangdong and N. Ghubade, Detection of changes in surveillance videos. in *Proceedings of IEEE Conference on Advanced Video and Signal Based Surveillance*, 237 – 242, 2003

11. R. Chellappa, C. Wilson and S. Sirohey, Human and machine recognition of faces: A survey. *Proceedings of the IEEE 83* (5). 705-740, 1995

12. M. Yang, D. Kriegman, N. Ahuja, Detecting Faces in Images: A Survey. *IEEE Trans. PAMI*, *Vol. 24* (1), 2002

13. V. Govindaraju, Locating human faces in photographs. *Intl. Journal on Computer Vision*, *19* (2). 129 - 146, 1996

14. H. Graf, T. Chen, E. Petajan, E. Cosatto, Locating faces and facial parts. in *IEEE Proc. of Int. Workshop on Automatic Face-and Gesture-Recognition*, (Zurich, Switzerland), 41–45, 1995

15. T. Leung, M. Burl and P. Perona, Face Localization via Shape Statistics. in *Proc. First Intl. Workshop Automatic Face and Gesture Recognition*, 154-159, 1995

16. Y. Dai, Y. Nakano, Face-Texture Model Based on SGLD and its Application in Face Detection in a Color Scene. *Pattern Recognition*, *29* (6). 1007-1017, 1996

17. H. Rowley, S. Baluja, T. Kanade Neural Network-Based Face Detection. in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 203-208, 1996

18. K. Sung, T. Poggio. Example-Based Learning for View- Based Human Face Detection *Technical Report AI Memo 1521, Massachusetts Inst. of Technology AI Lab,* 1994

19. M. Turk, A. Pentland, Face Recognition Using Eigenfaces. *IEEE Conference on Computer Vision and Pattern Recognition*. 586-591, 1991

20. B. Menser, F. Muller, Face detection in color images using Principal Component Analysis. *IEE Conference on Image Processing and its Applications*, *465*. 620-624, 1999

21. FaceDatabase, CMU. (http://www.ri.cmu.edu/projects/project_418.html)

22. Rice University Face Database. (http://www.owlnet.rice.edu/~elec301/Projects99/faces/images.html)

23. M. Huang, A. Mahajan, D. DeMenthon. Automatic Performance Evaluation for Video Summarization, Technical Report LAMP-TR-114, University of Maryland, 2004

24. I. Haritaoglu, D.Harwood, L. Davis, W4:Real-time surveillance of people and their activities. *IEEE Trans. PAMI*, *22* (8). 809–830, 2000

25. H. Roh, S. Kang, S. Lee, Multiple people tracking using an appearance model based on temporal color. in *Proc. International Conference on Pattern Recognition*, 643–646, 2000

26. A. Senior, A. Hampapur, Y. Tian, L. Brown, S. Pankanti, R. Bolle, Appearance models for occlusion handling. *2nd IEEE Workshop on Performance Evaluation of Tracking and Surveillance,* 2001

27. D. Greenhill, J. Renno, J. Orwell, G. Jones, Occlusion Analysis: Learning and Utilising Depth Maps in Object Tracking. in *British Machine Vision Conference*, 467-476, 2004

28. G. Triantafyllidis, D. Tzovaras, M. Strintzis, Occlusion Detection in Stereopairs. *International Conference on Trends in Communications EUROCON 2001*, Vol. 1, 99-102, 2001

29. J. Canny, A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.8 . 679-698, 1986

30. Jain, A. K., *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

31. E. Rivlin, M. Rudzsky, R. Goldenberg, U. Bogomolov, S. Lepchev, A Real-Time System for Classification of Moving Objects. *International Conference on Pattern Recognition*, Vol. 3, 2002

32. O. Javed, M. Shah, Tracking and Object Classification for Automated Surveillance. in *Proceedings of the 7th European Conference on Computer Vision*, 343-357, 2002

33. A. Rosenfeld, D. Doremann, and D. Dementhon, *Video Mining*. Kluwer Academic, 2003.

34. J. Han, M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

35. F. Qureshi, D. Terzopoulos Surveillance camera scheduling: a virtual vision approach. in *Proc. 3rd ACM international workshop on Video surveillance & sensor networks*, 131 – 140, 2005

36. W. Shao, D. Terzopoulos, Autonomous pedestrians. *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 19–28, 2005

37. D. Terzopoulos, Perceptive agents and systems in virtual reality. in *Proc. 10th ACM Symposium on Virtual Reality Software and Technology*, 1–3, 2003

38. J. Black, T. Ellis, D. Makris, . Hierarchical Content-based Database of Surveillance Video Data (http://www.kingston.ac.uk/~ku32195/VideoDatabase.html)

39. N. Johnson, D. Hogg, Learning the distribution of object trajectories for event recognition. in *Proceedings of the 6th British conference on Machine vision*, 583 - 592, 1995

40. S. Rao, P.S. Sastry, Abnormal activity detection in video sequences using learnt probability densities. in *TENCON 2003, Conference on Convergent Technologies for Asia-Pacific Region*, 369- 372, 2003

41. N. P. Cuntoor, R. Chellappa, Key Frame-Based Activity Representation Using Antieigenvalues. in *Proc. Asian Conf. on Computer Vision*, 2006

42. M. Beynon, D. Van Hook, M. Seibert, A. Peacock, D. Dudgeon, , Detecting abandoned packages in a multi-camera video surveillance system. in *Proceedings. IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, 221- 228, 2003

43. D. Gibbins, G. Newsam, M. Brooks, Detecting suspicious background changes in video surveillance ofbusy scenes. in *Proceedings 3rd IEEE Workshop on Applications of Computer Vision*, 22-26, 1996