12-2010

Evaluation of video based pedestrian and vehicle detection algorithms

Varun Bandarupalli
University of Nevada, Las Vegas

EVALUATION OF VIDEO BASED PEDESTRIAN AND VEHICLE DETECTION ALGORITHMS

by

Varun Bandarupalli

Bachelor of Technology in Electronics and Instrumentation Engineering
Jawaharlal Nehru Technological University, India
June 2007

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Electrical Engineering
Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2010

THE GRADUATE COLLEGE

We recommend that the thesis prepared under our supervision by

Varun Bandarupalli

entitled

Evaluation of Video Based Pedestrian and Vehicle Detection Algorithms

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

Muthukumar Venkatesan, Committee Chair

Emma E. Regentova, Committee Co-chair

Paulo Ginobbi, Committee Member

Laxmi Gewali, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

December 2010

ABSTRACT

Evaluation of Video Based Pedestrian and Vehicle Detection Algorithms

by

Varun Bandarupalli

Dr. Venkatesan Muthukumar, Examination Committee Chair
Associate Professor of Electrical and Computer Engineering,
University of Nevada Las Vegas

Video based detection systems rely on the ability to detect moving objects in video streams. Video based detection systems have applications in many fields like, intelligent transportation, automated surveillance etc. There are many approaches adopted for video based detection. Evaluation and selecting a suitable approach for pedestrian and vehicle detection is a challenging task. While evaluating the object detection algorithms, many factors should be considered in order to cope with unconstrained environments, non stationary background, different object motion patterns and the variation in types of object being detected.

In this thesis, we implement and evaluate different video based detection algorithms used for pedestrian and vehicle detection. Video based pedestrian and vehicle detection involves object detection through background foreground segmentation and object tracking. For background foreground segmentation, frame differencing, background averaging, mixture of Gaussians and codebook methods were implemented. For object tracking, Mean-Shift tracking and Lucas Kanade optical flow tracking algorithms were implemented.

The performance of each of these algorithms is evaluated by a comparative study; based on their performance such as ability to get good detection and tracking, CodeBook algorithm is selected as a candidate algorithm for background foreground segmentation and Mean-Shift tracking is used to track the detected objects for pedestrian and vehicle detection.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I take this opportunity to express sincere gratitude to Dr. Venkatesan Muthukumar my advisor, for his help and guidance. I would like to thank Dr. Emma Regentova for assisting and advising me throughout the research. I would also like to thank Ajay Kumar Mandava for helping and encouraging my work for the thesis.

Finally, I would like to thank family and all my friends for their support and constant encouragement.

CHAPTER 1

INTRODUCTION

The ability to reliably detect pedestrians from video data has very important applications in many fields like, intelligent transportation, automated surveillance and security, robotics, assistive technology for visually impaired, advanced human machine interfaces, automated driver assistance systems in vehicles etc. There are many technologies that are currently being used for pedestrian and vehicle detection such as ultrasonic sensors, Doppler radar sensors piezo-metric sensors etc. These sensors while being very effective have various drawbacks ranging from cost effectiveness to durability. Video based detection emerged as an important aspect of research, as proliferation high performance cameras and faster inexpensive computing systems became assessable. Video based detection provides fast accurate results at lower costs.

Pedestrian and vehicle detection in the fields intelligent transportation plays a vital role is various aspects such as pedestrian safety, retrieving pedestrian or traffic volume data. Accurately detecting pedestrians from a video is one of the most challenging tasks for object detection and there exists a lot of research in this area. Pedestrian are more vulnerable to accidents and collisions involving pedestrians often produce severe injuries. Each year in the United States, approximately 5,000 pedestrians are killed in traffic crashes, accounting for approximately 11% of all traffic fatality victims [2]. An accurate analysis of pedestrian statistics can help us to reinforce available safety measures for pedestrians.

The methodology and algorithms adopted for pedestrian detection can also be applied to people detection and automated surveillance systems. A very important application of people detection is in the field of automated surveillance. Intelligent video surveillance system has emerged as a very important topic of research in the field of Computer Vision in the recent years. The conventional approach in Video Surveillance involves a closed circuit camera installed in a public place capturing outdoor and indoor information and streaming the video information to the control center, where the information is monitored and analyzed by human observers and stored. An automated video detection system can obtain a description of events occurring in a monitored area and then to take appropriate action based on that interpretation, e.g., alert a human supervisor to reduce human involvement significantly and assist human operators for better monitoring [3].

Automated vehicle detection system has various applications in the fields of transportation which include, incident detection on a roadway or a cross-road, automating the process of ticketing the law offenders in matters such as speeding violation, red light running etc., simplifying the laborious tasks of counting and calculating volumes of vehicles [1].

In this thesis, we implement and evaluate video processing algorithms used for pedestrian and vehicle detection in real conditions and determine an algorithm suitable for both pedestrian and vehicle detection.

1.1 Thesis Outline

The structure of this thesis consists of five chapters.

Chapter 2 briefly discusses an overview of previous work in object detection, background subtraction, and object tracking algorithms.

Chapter 3 focuses on algorithms developed for object detection using background subtraction. This chapter explains the working of these algorithms. The results of the implemented algorithms are a presented in this Chapter 5.

Chapter 4 discusses the different tracking algorithms to localize the target object. This chapter also explains the working of the object tracking algorithms. The results of the implemented algorithms are a presented in this chapter 5.

Chapter 5 summarizes implementation details of the algorithms and discusses the results obtained from the use of the proposed algorithms implemented on real-time video sequences.

Finally, Chapter 6 summarizes the work done within the scope of this thesis and discusses the conclusions drawn from the work carried out.

CHAPTER 2

LITRETURE REVIEW

Pedestrian and vehicle detection from a stationary video is a very challenging task and the focus of lot of research topics as it has applications is many fields. A reliable pedestrian and vehicle detection system relies heavily on the system’s ability to detect and track objects of interest in a video.

2.1 Object Detection

Object detection in videos involves detecting the presence of an object in a sequence of images and location for precise recognition. Object tracking is to monitor object’s spatial and temporal changes during a video sequence, including its presence, position, size, shape, etc. The above two processes are interrelated because tracking needs the objects to detected, while detecting an object repeatedly in subsequent frames is necessary to help and verify the tracking.

There are three key steps involved in a video based detection systems: detection of interesting moving objects (object detection), tracking of such objects from frame to frame (object tracking), and analyze the results to recognize their behavior (objects recognition and pose estimation) [18].

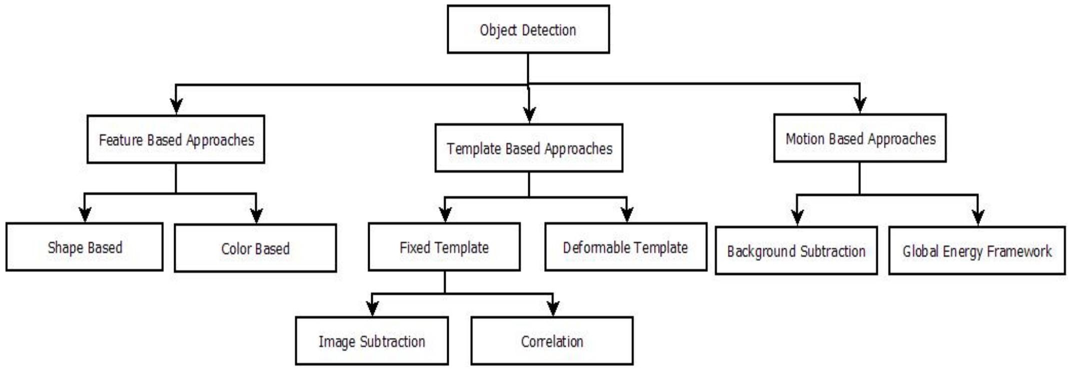


Figure 2.1 Object Detection Classification

There have been many approaches adopted for object detection as shown in figure 2.1. These methods can generally be classified into:

- Feature and based Methods
- Template based Methods
- Motion based Methods

2.1.1 Feature based object detection

Feature based detection is based on identifying the points of interest in an image such as edges, corners, color compositions, blobs, their points (corners) and ridges. Feature based methods are generally implemented on individual images rather than a sequence of images. The core algorithm in these methods being divided into two categories, 1) extract features 2) classify these features and trains a system for recognition and classification. Feature (specific structures such as points, edges, curves, boundaries etc.) selection is very important as the rest of the algorithm depends on how good the features are detected [4]. There are several approaches adopted for feature selection and learning methods for pedestrian and vehicle detection. Papageorgiou et. al. [5] applied Support Vector machine (SVM) and Haar Wavelet features to train a pedestrian detector. This

paper also introduces the usage of motion cues to improve detection accuracy in a video sequences. D. M. Garvilla [6] uses image matching using distance transforms involving the features extracted locally at various image locations such as edge points. Leibe et. al. [73] follows a two staged approach, first a codebook is created that contains information of local structures appear on the object (local shape feature information) and in the second step, an implicit shape model is trained to classify and recognize objects. In addition to static local features such as intensity, Viola et. al. [8] used local motion feature information to detect face and pedestrians.

Dalal et. al. [10] implemented locally normalized Histogram of Oriented Gradient (HOG) descriptors which use edge orientation histograms. This method proved to be robust and achieved promising results for pedestrian detection. Wu et. al. [9] have achieved similar detection results with discriminative local shape and contour fragments and edge-let features.

The goal of all these approaches is to build a robust and generalized object detection systems based on various features and different learning sets. Feature based object detection is very challenging task. The primary difficulty with these algorithms is selection of features, accurate prior information of the feature properties and limited extrapolation of the feature set properties. Different features have different drawbacks; for example color feature based approach have to deal with pedestrians wearing different colored clothing which sometimes are indistinguishable from the background. Shape feature based algorithms have to deal with different poses and positions of a pedestrian and also deal with the

situations such as pedestrian carrying bags or wearing a hat. Most of these approaches are complex searches for specific patterns or textures, and gathering a representative learning set for these algorithms are computationally exhaustive and expensive.

2.1.2 Template based object detection

Template based detection is the process of matching features between a template and the image under analysis. A simple version of template matching involves the image which is represented as a bi-dimensional array of intensity values, is compared using a suitable metric (typically the Euclidean distance) with a single template representing the object. In template-based object detection, the features of tracked templates are learned in the initialization phase of the detection process. The detection algorithm then searches the frame for these features. Occlusion is detected by the absence of the template features in the frame beyond a certain threshold. Objects in such algorithms are not detected during occlusion but after object reappearance. While such algorithms work well for tracking of single objects, they fail to robustly track multiple objects during occlusion. Split is not explicitly detected, however, if the object is split due to an obstacle, the minimization of the template's feature comparison function will choose to which portion of the split object the match is made, if any. Probabilistic models are being developed as templates to characterize different objects [77].

2.1.2 Motion based object detection

The capability of extracting moving objects from a video sequence is a typical first step in computer vision applications. The motion of the objects complicates

process by adding object’s temporal change requirements; on the other hand, it also provides additional information for detection and tracking. A common approach for discriminating moving objects from the background is detection by background subtraction. The basic idea of background subtraction is to subtract or difference the current image from a reference background model. The subtraction identifies non-stationary or new objects [36]. Background subtraction is a critical part of object detection systems as its outcome is fed to higher level processes such as object recognition and tracking and these processes rely heavily on the accuracy of background subtraction techniques. The performance of background subtraction methods hugely depend on the background model.

Background subtraction can be classified into non-recursive and recursive techniques as shown in the figure 2.2. A non-recursive technique uses a sliding-window approach for background estimation. It stores a buffer of the previous *'l'* video frames, and estimates the background image based on the temporal variation of each pixel within the buffer. Recursive techniques do not maintain a buffer for background estimation. Instead they recursively update a single background model based on each input frame. As a result, input frames from distant past could have an effect on the current background model.

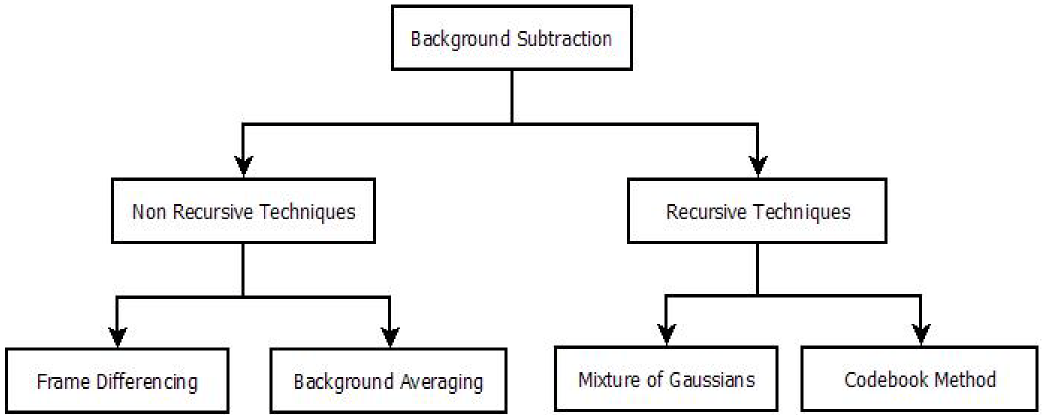


Figure 2.2 Background Subtraction Classification

The background scene, even when captured from a stationary camera, poses challenging demands such as illumination changes, the outdoor scenarios having changes in background geometry such as moving trees, rippling water, flickering monitors etc. A robust background modeling algorithm should also handle situations where new objects are introduced or old ones removed from the background. Furthermore, the shadows of the moving objects can cause problems. Even in a static scene, frame to frame changes can occur due to noise and camera jitter. Moreover, the background modeling algorithm should operate in real-time [11].

Frame differencing [33, 65] approach detects moving objects in video sequences. The basic idea is to subtract the current frame from a previous frame and to classify each pixel as foreground or background by comparing the difference with a threshold [33]. In practice, several difficulties arise such as selection of appropriate threshold, pixels interior to the foreground object not being detected (aperture problem) [35], fluttering objects, illumination changes, clouds, shadows etc. To deal with these difficulties several methods have been

proposed by R. Cucchiara et. al. [12]. There are several variants to the frame difference method; Xia et. al. [38] implemented double and triple difference, but the algorithm has an inherent flaw as it is completely dependent on the motion of the objects.

Background Averaging [35] is a straightforward background subtraction algorithm, where the background model is built by taking arithmetic average of pixels values of the last *'n'* frames [43] and the current frame is differenced from the model. The result is compared to a threshold to differentiate between foreground and background pixels. This method needs very low computational power and memory, but it is not accurate. Several methods have been proposed to improve the performance such as selective update model; Koller et. al. [13] i.e. to update pixels only the pixels identified as moving objects; Jabri [14] included edge information with background average method to achieve better results. Sigari et. al. [41] implemented a fuzzy running Gaussian average; this is also a case of selective update using a fuzzy logic and achieved 6% more accuracy than the previous method [43]. Although averaging background method is fast and requires less memory, it has some major drawbacks. Primarily, background model is not robust to sudden changes in the background. In the simplest form, a background image is a long term average image [15] as in equation 2.1.

$$B(x, y, t) = \frac{1}{t}\sum_{k=1}^{t} I(x, y, k) \qquad 2.1$$

where, x and y are pixel co-ordinates and t is the number of images.

The obvious error in this approach is that the lighting conditions change over time and to overcome this problem, moving window average is used. Each image contribution to the background is weighted to decrease exponentially.

$$B(x, y, t) = (1 - \alpha)\, B(x, y, t-1) + \alpha\, I(x, y, t) \qquad 2.2$$

Where 'α' is the time constant for weighted average in equation 2.2, and should be in the range (0, 1). Using exponential forgetting function is equivalent to using Kalman filtering to track the background image. Kalman filter is a widely-used recursive technique for tracking linear dynamical systems under Gaussian noise. Many different versions have been proposed for background modeling, the simplest version uses only the luminance intensity [66, 67]. Unlike Kalman filter which tracks the evolution of a single Gaussian, the Mixture of Gaussians (MoG) method [15] tracks multiple Gaussian distributions simultaneously. Mixture of Gaussians method has a superior analytical form and efficiency when compared to other previously described models. Similar to the non-parametric (background averaging) model, Mixture of Gaussians method maintains a density function for every pixel and is capable of handling multi modal backgrounds and it can be updated without having to store large number of frames in buffer hence reducing memory costs.

Mixture of Gaussians method works based on the persistence and the variance of each of the Gaussians. Pixel values that do not fit the background distributions are considered to be part of the foreground until there is a Gaussian that includes them with sufficient, consistent evidence in favor of its inclusion in the new background mixture [34].

The Mixture of Gaussians method describes each pixel $I(x) = I(x, y)$ as mixture of n Gaussian distributions as shown in equation 2.3.

$$P(X_t) = \sum_{i=1}^{k} \omega_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{I,t}), \qquad 2.3$$

where k is the number of Gaussians, $\eta(X_t, \mu_{i,t}, \Sigma_{I,t})$ is a multivariate normal distribution and w_k is the weight of k^{th} Gaussian. The background mixture model is dynamically updated, based on the criterion that the incoming pixel belongs to an existing distribution and pixel value occurs in the interval of $\pm$ 2.5 standard deviations.

Mixture of Gaussians method has some disadvantages where backgrounds having fast variations are not easily modeled with just a few Gaussians accurately and it may fail to provide sensitive detection [16]. In addition, depending on the learning rate to adapt to background changes, Mixture of Gaussians faces problems; for a low learning rate, it produces a wide model that has difficulty in detecting a sudden change to the background. If the model adapts too quickly, slowly moving foreground pixels will be absorbed into the background model, resulting in a high false negative rate. This is called the foreground aperture problem [17]. To overcome the foreground aperture problem, a technique estimating the probability density function at each pixel from many samples using kernel density estimation technique was developed which adapts very quickly to the changes in background process [16]. The non-parametric technique in kernel density estimation cannot be used when long time periods are needed to sufficiently sample the background. To overcome this, the

codebook [36] algorithm that constructs a highly compressed background model was proposed.

The Codebook Method [36] adopts a quantization/clustering technique, to construct a background model from long observation of image sequences. For each pixel, it builds a codebook consisting of one or more codewords. Samples at each pixel are clustered into the set of codewords based on a color distortion metric together with brightness bounds. Not all pixels have the same number of codewords. The background is encoded on a pixel-by-pixel basis. Detection involves testing the difference of the current image from the background model with respect to color and brightness differences. The incoming pixel is classified as background if the color distortion is less than the detection threshold and its brightness lies within the brightness range of that codeword otherwise it is classified as foreground.

Global energy frameworks: The motion detection problem is formulated to minimize a global objective function and is usually performed using stochastic (Mean-field, Simulated Annealing) or deterministic relaxation algorithms (Iterated Conditional Modes, Highest Confidence First).

In that direction, the spatial Markov Random Fields [76] have been widely used and motion detection has been considered as a statistical estimation problem. Although this estimation is a very powerful, usually it is very time consuming [75].

2.2 Object Tracking

The efficient tracking of visual features in complex environments is a challenging task for the computer vision applications. Real time applications such as surveillance and monitoring, perceptual user interfaces, smart rooms, and video compression all require the ability to track moving objects [46]. The primary goal of a object tracker is to find targets between consecutive frames in a sequence of images. The computational complexity of the object tracker is critical for most applications with only a small percentage of system resources being allocated for tracking, while the rest is assigned to preprocessing stages or to high-level tasks such as recognition, trajectory interpretation.

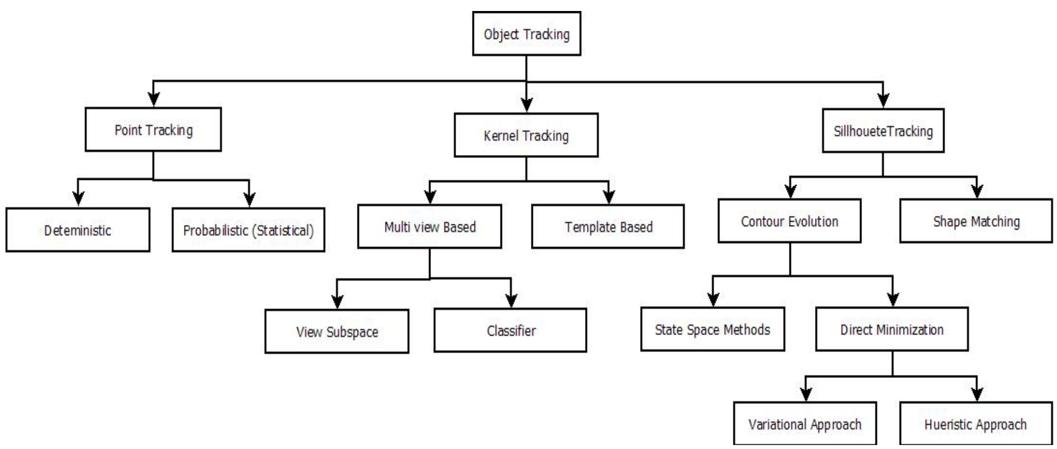


Figure 2.3 Object Tracking Classification

The aim of an object tracker is to generate the trajectory of an object over time by locating its position in every frame of the video. Object tracker also provides the complete region in the image that is occupied by the object at every time instant. In various object tracking approaches, the objects are represented

using the shape and/or appearance models [18]. Figure 2.3 shows the classification of various tracking algorithms.

Object tracking can be classified into three types:

- Point Tracking
- Kernel Tracking
- Silhouette Tracking

2.2.1 Point Tracking

In this approach, objects being tracked are represented in terms of points and association of the points, based on previous object state which includes object position and motion. A multi-point association is employed and an external mechanism is used to detect objects in every frame. These approaches are generally implemented when object sizes are small and have low velocity. Association of the points across the frames is a complicated problem and is affected even more by presence of occlusion and misdetections etc. Point tracking can be further classified in the deterministic and probabilistic approaches based on their association methods.

Many algorithms have been proposed for deterministic approaches. Sethi and Jain [19] proposed an algorithm that considered two consecutive frames initialized by the nearest neighbor criterion. The point associations are exchanged iteratively to minimize the cost. Veenman et. al. [20] extended the work of Sethi and Jain [19], and Rangarajan and Shah [21] by introducing the common motion constraint. The common motion constraint provides a strong constraint for coherent tracking of points that lie on the same object; however, it

is not suitable for points lying on isolated objects moving in different directions. The algorithm is initialized by generating the initial tracks using a two-pass algorithm and the cost function is minimized by Hungarian assignment algorithm in two consecutive frames. This approach can handle occlusions and misdetection errors. However, it is assumed that the number of objects is the same throughout the sequence i.e. no object enters or exits. The Kalman filter has been extensively used in the vision community for tracking. Broida et. al. [22] used the Kalman filter to track points in noisy images. In stereo camera-based object tracking, Beymer and Konolige [74] use the Kalman filter for predicting the object's 18 positions and speeds in x-y-z dimensions. Rosales and Sclaroff [23] use the extended Kalman filter to estimate 3D trajectory of an object from 2D motion.

2.2.2 Kernel Tracking

Kernel refers to the object shape and appearance. For example, the kernel can be a rectangular shaped or an elliptical shaped template with an associated histogram. Objects are tracked by computing the motion of the kernel in consecutive frames [18]. Kernel tracking is typically performed by computing the motion of the object, which is represented by a primitive object region, from one frame to the next. The object motion is generally in the form of parametric motion (translation, conformal, affine, etc.) or the dense flow field computed in subsequent frames. These algorithms differ in terms of the appearance representation used, the number of objects tracked and the method used to estimate the object motion.

The most common and primitive approach of kernel tracking is template matching, Birchfeild et. al. [70] used image illumination and image gradients feature in template matching. A major limitation of template matching is high computational cost as the algorithm sums up to brute force search. Comaniciu [46, 71] used a weighted histogram computed from a circular region to represent the object. Instead of performing a brute force search for locating the object, they use the mean-shift procedure. The mean shift algorithm was originally invented by Fukunaga and Hostetler [24] for data clustering, which they called a “valley-seeking procedure”. It was first introduced into the image processing community several years ago by Cheng [48]. Comaniciu et. al. successfully applied it to image segmentation and tracking.

The mean shift tracking algorithm uses a color histogram to describe the target region. The tracker maximizes the appearance similarity iteratively by comparing the histograms of the object '*Q'*, and the window around the hypothesized object location, '*P'* [47]. The Kullback-Leibler divergence, Bhattacharyya coefficient and other information-theoretic similarity measures are commonly employed to measure the similarity between the template region and the current target region. At each iteration, the mean-shift vector is computed such that the histogram similarity is increased. This process is repeated until convergence is achieved, which usually takes five to six iterations [18].

Comaniciu et. al. extended the mean-shift tracking approach by using a joint spatial-color histogram instead of just a color histogram. An obvious advantage of the mean-shift tracker over the standard template matching is the elimination

of a brute force search, and the computation of the translation of the object patch in a smaller number of iterations. To track objects in video frame sequences, the color image data has to be represented as a probability distribution. Color histograms are used to accomplish this task. Color distributions derived from video image sequences change over time, so the mean shift algorithm has to be modified to adapt dynamically to the probability distribution it is tracking. Bradski [25] implemented CAMshift (Continuously Adaptive Mean shift) algorithm to meet these requirements.

Jepson et. al. [72], propose an object tracker that tracks an object as a three component mixture, consisting of the stable appearance features, transient features and noise process. Another kernel based approach to track a region defined by a primitive shape is to compute its translation by use of an optical flow method. Optical flow methods are used for generating dense flow fields by computing the flow vector of each pixel under the brightness constancy constraint [26] [19],

$$I\,(x, y, t) - I\,(x + dx, y + dy, t + dt) = 0 \qquad 2.4$$

This computation is always carried out in the neighborhood of the pixel either algebraically [27] or geometrically [26]. Extending optical flow methods to compute the translation of a rectangular region is trivial. Shi and Tomasi [28] proposed the Lucas Kanade optical flow object tracker which iteratively computes the translation (du, dv) of a region centered on an interest point.

2.2.3 Silhouette Tracking

Silhouette based methods provide an accurate shape description for the objects tracked. The goal of a silhouette-based object tracker is to find the object region in each frame by means of an object model generated using the previous frames. This model can be in the form of a color histogram, object edges or the object contour. Tracking is performed by estimating the object region in each frame. Silhouette tracking methods use the information encoded inside the object region. This information can be in the form of appearance density and shape models which are usually in the form of edge maps. Given the object models, silhouettes are tracked by either shape matching or contour evolution. The representations chosen by the silhouette-based object trackers can be in the form of motion models (similar to point trackers), appearance models (similar to kernel trackers), or shape models or a combination of these.

Object appearance is usually modeled by parametric or nonparametric density functions such as mixture of Gaussians or histograms. Object shape can be modeled in the form of contour subspace where a subspace is generated from a set of possible object contours obtained from different object poses [32]. Additionally, object shape can be implicitly modeled via a level set function where the grid positions are assigned at the distance generated from different level set functions corresponding to different object poses [29].

Appearance-based shape representations are also commonly used by researchers who employ a brute force silhouette search. For edge-based shape representation, Hausdorff distance is the most widely used measure. However,

Hausdorff measure is known for its sensitivity to noise. Hence instead of using the maximum of distances, researchers have considered using an average of the distances [30]. Occlusion handling is another important aspect of silhouette tracking methods. Usually methods do not address the occlusion problem explicitly. A common approach is to assume constant motion or constant acceleration where, during occlusion, the object silhouette from the previous frame is translated to its hypothetical new position. Few methods explicitly handle object occlusions by enforcing shape constraints [31] [29].

Based on the literature review, motion based algorithms were implemented to obtain pedestrian and vehicle detection and point tracking algorithms were implemented for tracking.

CHAPTER 3

OBJECT DETECTION

Many computer vision applications depend heavily on the ability to detect moving objects in a video stream and extract information. Images from the video stream are analyzed and processed by various video processing techniques in a reliable and effective way taking into account problems like unconstrained environments, non-stationary background and different motion patterns of objects. Furthermore different types of objects such as pedestrians, vehicles etc. pose various problems in object detection [40].

Objects in the pedestrian and vehicle detection primarily focus on extracting foreground objects information and classifying the foreground objects into pedestrian or vehicles or any other objects. One of the primary advantages of this model is a stationary camera which provides an opportunity to model a steady background to detect the foreground objects.

The basic steps involved are extracting foreground object information using background subtraction techniques, applying connected component analysis and foreground clean up algorithms and classifying the foreground objects into pedestrians or vehicles.

3.1 Background Foreground Segmentation

Background foreground segmentation is achieved by background subtraction from an image leaving the non-stationary foreground components. Background Subtraction is a process to detect a movement or significant differences inside of

the video frame, when compared to a reference, and to remove all the non-significant components (background).

Background Subtraction Algorithms:

- Frame differencing [33, 65]
- Background Averaging [35] [41]
- Mixture of Gaussians method [34]
- Codebook Method [36]

3.1.1 Frame Differencing

Frame difference method is a basic background subtraction method. Frame difference method uses of the difference between the two consecutive frames in a video sequences or the difference between current frame and a reference background frame to extract motion region of an image creating a difference image. In the difference image, the pixels with same intensity i.e. background pixels are eliminated while the pixels with changed intensities of the foreground remains as the foreground. This change is caused by movement, but all the pixel intensities are not the same, minor variations in the intensities give a difference value and are considered as foreground pixel. To avoid this, a binary process such as thresholding is applied on the difference image to distinguish the moving foreground objects and the stationary background [38]. Each pixel value in the difference image larger than the threshold i.e. the difference is large enough to be classified as foreground is assigned as a foreground object and the rest background.

$$FG(i,j) = \begin{cases} 1 & \text{if } |I_t(i,j) - I_{t-1}(i,j)| > threshold \\ 0 & \text{otherwise} \end{cases} \quad 3.1$$

where, FG(i,j) is the foreground image, i and j are pixel coordinates

$I_t(i,j)$ and $I_{t-1}(i,j)$ are current and previous images

Frame difference method is computationally fast and inexpensive but has some drawbacks. Selecting threshold value is a very important aspect of the Frame Difference Algorithm. Foreground detection is very sensitive to threshold value. Selecting a low threshold value leads to false detection as minor changes in illumination cause a difference in pixel value leading to false detection of foreground. Selecting a high threshold leads to detection failure of foreground objects, as even if there is difference in pixel value, as the threshold is high the pixels are discarded.

Frame difference method is dependent on the movement of foreground objects, majority of pixels interior to foreground objects occupy the same locations in the consecutive frames. In the difference image, these pixels are considered as background as the pixels belonging to same object have same intensities and the difference of these pixel values does not cross the threshold causing holes of background inside the detected foreground objects. The effects the above problem can be reduced by subtracting every 3rd frame or every 5th frame in the video stream instead of consecutive frames.

Frame difference is completely dependent on the motion of the foreground objects. If the object becomes stationary, the algorithm cannot detect the objects.

This is a drawback for pedestrian detection as the algorithm cannot detect stationary people at crosswalks.

Frame Difference Pseudo Code

Initialization values of variable used for the algorithm:

Threshold = 30

Algorithm

Step 1: Grab a frame I_{t-1}

Step 2: Convert to grey scale single channel image g_{t-1}

Step 3: Grab the next frame I_t

Step 4: Convert the second frame to single channel g_t

Subtract the second frame from the first frame in each pixel value respectively to give a difference image.

Step 5: IdiffImg = $g_t - g_{t-1}$

Step 6: If (IdiffImg(x, y) > Threshold) then

Step 6a: foreground Image (x, y) = 1

Step 7: Else

Step 7a: foreground Image (x, y) = 0

Step 8: End

The difference image under ideal conditions should consist of only foreground objects that are moving but due to illumination changes and noise some pixels have a positive values and if the value is greater than the threshold, they are considered as foreground.

3.1.2 Averaging Background Method

The Averaging Background Method is also known as Gaussian Average Method. In this method, background model is built by arithmetic average of pixel values in a sequence images and frame difference is applied on next image and the background model. This algorithm is memory efficient and fast, but has a shortcoming of being not very accurate [41].

$$\mathrm{BG}\ (\mathrm{x}, \mathrm{y}) = \sum_{k=1}^{n} I(x, y, k) \qquad 3.2$$

where, BG(x,y) is the background model

n is the number if images to learn the background

I(x, y, k) is the Current image. x, y are pixel coordinates

In Averaging Background Method algorithm, the background is modeled based on ideally fitting a Gaussian probability density on the last '*n*' pixel value. The averaging method basically learns the average and standard deviation of each pixel as its model of the background. A difference image is derived by subtracting the average model from the current frame and the new image is subject to threshold like in frame difference [39].

Background averaging has some drawbacks. It is not robust to scenes with slow moving objects. It cannot handle backgrounds with multiple stages such as moving trees and recovers slowly when the background is changed.

Averaging Background Pseudo Code

Initialization values of variable used for the algorithm:

No. of frames to learn Background = 30

Ihi = high threshold = 30, Ilow = low threshold = 9

Ihi, Ihi1, Ihi2, Ihi3: Images to store higher threshold bound channel wise

Ilow, Ilow1, Ilow2, Ilow3: Images to store lower threshold bound

Igray1, Igray2, Igray3: Grayscale values of current image to compare with the threshold values

Iavg: Average of pixel values; Idiff: difference image

Step 1: If (Current frame count < No. of frames to learn Background)

Step 2: Accumulate background

Step 2a: Add image to Iavg

Step 2b: Subtract image from previous Image

Step 2c: Add difference image to Idiff

Step 3: Else if (Current frame count = No. of frames to learn Background) then Create Models Statistics

Step 3a: Scale Idiff to high threshold and add Iavg = Ihi

Step 3b: Split image channel wise into Ihi1, Ihi2 and Ihi3

Step 3c: Scale Idiff to low threshold and add Iavg = ILow

Step 3d: Split image channel wise into Ilow1, Ilow2 and Ilow3

Step 4: Else backgroundDiff

> *Step 4a: Split the current image Igray1, Igray2, Igray3*
>
> *Step 4b: If (Ilow < Igray < Ihi) then Pixels are foreground*
>
> *Step 4c: Else Background*

Step 5: End

3.1.3 Mixture of Gaussians

In the mixture of Gaussians model, values of a pixel are modeled as a mixture of Gaussians. Based on the persistence and the variance of each of the Gaussians of the mixture, it is determined which Gaussians correspond to background colors. Pixel values that do not fit the background distributions are considered foreground until there is a Gaussian that includes them with sufficient, consistent evidence supporting it. Each pixel of the background is modeled by a separate mixture of '*K*' Gaussians as

$$P\,(X_t) = \sum_{i=1}^{k} \omega_{\,i,\,t} * \eta\,(X_t,\ \mu_{i,\,t},\ \textstyle\sum_{I,\,t}) \qquad 3.3$$

where *K* is the number of Gaussians (*K* = 3 to 5). X_t is the current pixel value vector, which consists of red, green, blue component intensity. $W_{i,t}$ is an estimate of the weight of the i^{th} Gaussian in the mixture at time '*t*';

$\mu_{i,\,t}$ and $\sum_{I,\,t}$ are respectively the mean value and the covariance matrix of the i^{th} Gaussian in the mixture at time '*t*' (This assumes that the red, green, blue pixel components are independent) , and 'η' is a Gaussian probability density function [42]

$$X_t = (\,x_t^r,\ x_t^g,\ x_t^b)$$

$$\mu_{i,\,t} = (\mu_{i,\,t}^{\ r}\,,\ \mu_{i,\,t}^{\ g},\ \mu_{i,\,t}^{\ b}) \qquad 3.4$$

$$\frac{1}{(2\pi)n/2\ |\Sigma|1/2}\ e^{-1/2\ (Xt\,-\,\mu t)T*\ \Sigma-1(Xt\,-\,\mu t)} \qquad 3.5$$

Foreground segmentation consists of two independent problems: 1) estimating the parameters of k Gaussians and 2) evaluating the likelihood of each Gaussian to represent the background.

1) Estimating Parameters of K-Gaussian Distributions

The weights and means are initialized to 0. Variances are set to a large value 'V_0'. Then at time 't', every new pixel value 'X_t' is checked against the existing 'K' Gaussian distributions, until a match is found. A match is defined as a pixel value 'X_t' within 2.5 standard deviations of a distribution. The 'μ' and ∑' parameters of the unmatched Gaussian distributions remain the same, and the parameters of Gaussian 'G_i' in the mixture that matches 'X_t' is updated as follows

$$\mu_t = (1-\rho) * \mu_{t-1} + \rho * X_t$$

$$\textstyle\sum_{i,t} = (1-\rho) * \sum_{i,t-1} + \rho * diag[(X_t - \mu_{i,t})^T * (X_t - \mu_{i,t})] \qquad 3.5$$

where, $\rho = \alpha * \eta (X_t \mid \mu_{i,t-1} , \sum_{i,t-1})$, '$\alpha$' is the learning rate, *diag* [x] produces a diagonal matrix from matrix 'X_t'. If none of the 'K' Gaussians matches the current pixel value 'X_t', the least probable distribution G_j is reassigned, where $j = argmin \{\omega_{i, t-1}\}$, $\omega_{j, t-1} = W_0$, $\mu_{j,t} = X_t$,

$$\textstyle\sum_{j, t-1} = V_0 , I \qquad 3.6$$

Where, 'W_0' is a small prior weight; 'I' is a 3 x 3 identity matrix. Then the weight of all 'k' Gaussian distributions at time 't',' $\omega_{i, t}$ ' are updated as:

$$\omega_{i,t} = (1-\alpha) * \omega_{i,t-1} + \alpha * M_{I,t}$$

$$\omega_{i,t} = \omega_{i,t} \;/\; \textstyle\sum_{m=1}^{k} \omega_{i,t} \qquad 3.7$$

where, '$M_{i, t}$' = 1 for the Gaussian distribution, 'G_i' which matched the 'X_t', and 0 for the unmatched Gaussians.

2) Background Model Estimation

After the parameters of each pixel model are updated, the Gaussians which are most likely to be produced by background processes are determined. First, the Gaussians are ordered by the value of $\omega / |\sum|$, so the most likely background

distributions will remain on top and the less probable transient background distributions will move towards the bottom and eventually be replaced by new distributions. Then, the first '*B*' distributions are chosen as the background model and expressed as shown in equation 3.8

$$B = \mathrm{argmin}_b \left(\sum_{m=1}^{k} \omega_k > T\right) \qquad 3.8$$

where '*T*' is a threshold (0.5 < T < 1), the first '*B*' components of the sorted mixture Gaussians are responsible for the background. If the pixel 'X_t' is best modeled by one of the background components (the pixel value 'X_t' matches one of the '*B*' distributions), it is marked as background, otherwise it is classified as foreground.

Mixture of Gaussians Pseudo code

Initialization values of variable used for the algorithm:

No. of Components = k = 3, Learning rate α = 0.01, Background Threshold = BgThr = 0.9, Standard Deviation threshold = StdDevThr= 3.5

Initial Weights = 0.05, Initial Standard Deviation = InitStdDev = 6

Mean(i, j, k), Weight (i, j, k) , SD (i, j, k) : Mean, weight, standard deviation matrices. i, j pixel locations, k number of gaussians.

rank(i, j, k): store rank values i.e. likelihood of the pixel belonging to foreground or the background

Step 1: Initialize Weight, Mean, Standard Deviation Matrices

Step 1a: Mean (i, j, k) = random in range (0 – 255)

Step 1b: Weight (i, j, k) = 0.05

Step 1c: SD (i, j, k) = InitStdDev

Step 1d: diffImg = Current frame - Mean

*Step 2: if (diffImg (i, j, k) < StdDevThr * SD (i, j, k))   [Match]*

*Step 2a: Update Weight (i, j, k) = (1-α) * Weight (i, j, k) + α*

Step 2b: P = α / Weight (i, j, k)

*Step 2c: Update Mean (i, j, k) = (1-P)*Mean (i, j, k) + P*Current frame (i, j)*

Step 2d: Update SD (i, j, k) = $[(1-P) * (SD(i, j, k))^2 +$

$P * (Current\ Frame\ (i, j) - (Mean(i, j, k))^2]^{1/2}$

Step 3: Else [No Match, create new Gaussian]

*Step 3a: Update Weight (i, j, k) = (1-α) * Weight (i, j, k)*

Step 3b: Min (i, j, x) = minimum [Weight (i, j, x)]

Step 3c: Mean (i, j, x) = frame(i, j)

Step 3d: SD(i, j, x) = initStdDev

Step 4: Normalize Weight (i, j, k) = Weight (i, j, k) / Sum [Weight (i, j, k)]

*Step 5: Update bgImg (i, j, :) = bgImg(i, j, k) + Mean(i, j, k) * Weight (i, j, k)*

Step 6: Update rank (i, j, :) = Weight (i, j, :) / SD (i, j, :)

Step 7: Extract Foreground if (Weight (i, j, :) >= threshold) then

*Step 7a: if (diff (i, j, :) <= StdDevThr * SD (i, j, :) then*

Step 7b: fg (i, j) = 0 (Background)

Step 7c: Else fg (i, j) = 1(Foreground)

Step 8: End

3.1.4 The Codebook Method

In the codebook method, background model is built considering color and brightness changes. For each pixel, a codebook consisting of one or more

codewords is built. A codebook is formed to represent significant states in the background. A new pixel value is compared to observed values. If the value is close to a prior value, then it is modeled as a perturbation on that color and is associated with that corresponding codebook. If it is not close, then it can seed a new group of colors to be associated with that pixel forming a new codebook. The result could be envisioned as a bunch of blobs floating in RGB space, each blob representing a separate volume considered likely to be background [35].

The codebook algorithm adopts a quantization/clustering technique, to construct a background model from long observation sequences. For each pixel, it builds a codebook consisting of one or more codewords. Samples at each pixel are clustered into the set of codewords based on a color distortion metric together with brightness bounds. Not all pixels have the same number of codewords. The clusters represented by codewords do not necessarily correspond to single Gaussian or other parametric distributions. The background is encoded on a pixel-by-pixel basis. Detection involves testing the difference of the current image from the background model with respect to color and brightness differences.

Construction of initial codebook:

Let X be a training sequence for a single pixel consisting of N RGB-vectors:

$X = \{X_1, X_2 \ldots\ldots\ldots X_n\}$

Let $C = \{C_1, C_2 \ldots\ldots\ldots C_l\}$ represent the codebook for the pixel consisting of L codewords. Each pixel has a different codebook size based on its sample variation.

Each codeword C_i, i = 1 . . . L; consists of an RGB vector $V_i = (R_i, G_i, B_i)$ and a 6-tuple $aux_i = \{ i_i, J_i, f_i, \lambda_i, p_i, q_i)$ The tuple aux_i contains intensity(brightness) values and temporal variables described below:

I, J: the minimum and maximum brightness, respectively, of all pixels assigned to this codeword

f: the frequency with which the codeword has occurred

λ: the maximum negative run-length (MNRL) defined as the longest interval during the training period that the codeword has NOT recurred

p; q the first and last access times, respectively, that the codeword has occurred.

The initial training period each value 'X_t' sampled at time 't' is compared to current codebook to determine which codeword 'C_m' matches it. The color distortion measure and brightness bounds are employed to determine which codewords matched best.

The codebooks are matched when pure colors of 'X_t' and 'C_m' are close enough and the brightness of 'X_t' lies between acceptable brightness bounds of 'C_m'.

In practice, the choice of RGB is not particularly optimal. It is better to use a color space aligned with brightness, such as the YUV color space. The reason for this is that, empirically, most of the variation in background tends to be along the brightness axis, not the color axis.

When we have an input pixel $X_t = (R, G, B)$ and a codeword C_i where $V_i = (R_i, G_i, B_i)$,the color distortion δ can be calculated as shown in equation 3.9

$$P^2 = || X_t ||^2 \ cos^2 \theta = (X_t , V_i)^2 / || V_i ||^2$$

$$Colordist\ (X_t, V_i) = \delta = \sqrt{(||\ Xt\ ||2 - p2\ )} \qquad 3.9$$

where, $||\ X_t\ ||^2 = R^2 + G^2 + B^2$, $||\ V_i\ ||^2 = R^2{}_i + G^2{}_i + B^2{}_i$, $(X_t, V_i)^2 = (R_iR + G_iG + B_iB)^2$. Color distortion measure can be interpreted as a brightness-weighted version in the normalized color space. This is equivalent to geometrically rescaling (normalizing) a codeword vector to the brightness of an input pixel. This way, the brightness is taken into consideration for measuring the color distortion, avoiding the instability of normalized colors.

For brightness changes in detection, I and J (minimum and maximum brightness) statistics are stored, which are the minimum and maximum brightness of all pixels assigned to a codeword. The brightness changes are allowed to vary in range [I_{low} , I_{high}]

$$I_{low} = \alpha J \qquad I_{high} = min\ \{\ \beta J,\ I\ /\ \alpha\ \} \qquad 3.10$$

where, α < 1 and β >1 typically the range [I_{low} , I_{high}]

Brightness function is defined as:

Brightness (I, (i,j))	=	true	if	$I_{low} < \|\| X_t \|\| < I_{high}$	
	=	false	otherwise		3.11

Foreground detection:

For an incoming pixel X foreground or background classification FG(x) (foreground image) is given as follows:

Step 1: x = (R, G, B), I $\leq \sqrt{(R2\ +\ G2\ +\ B2)}$

Step 2: For all codewords μ, find codeword C_m matching X based on:

- Colordist(X, C_m) < ε
- Brightness(I, (I, J)) = true

Step 3:

FG(x)	=	foreground	if there is no match (step 2)
	=	background	otherwise

ε is the detection threshold [36].

Codebook Method Pseudo code

x_t (δ): Current pixel value vector; C_m (ε): Codeword values; aux_m : tuple contains intensity(brightness) values and temporal variables

Step 1: If (Current frame count < No. of frames to learn Background) then

Step 1a: for t = 1 to N (N = No. of frames to learn Background)

Step 1b: $x_t = (R2 + G^2 + B^2)^{1/2}$

Step 1c: If (δ < ε and bright [I, {I, J}] = True) (match the codewords) then

Step1d: Update the matched codeword C_m

$V_m = (f_m(R_m), f_m(G_m), f_m (B_m))$

$aux_m = \{ I_m, J_m, f_m, \lambda_m, p_m, q_m\}$

Step 1e: Create a new codeword

$V_m = (R_m, G_m, B_m)$

$aux_m = \{ I, I, 1, t\text{-}1, t, t\}$

Step 2: Else (Background Subtraction)

Step 2a: $x_t = (R^2 + G^2 + B^2)^{1/2}$

Step 2b: If (δ < ε and bright[I, {I, J}] = True) then (Match the codewords)

Goto Step 1d (Update the codeword)

Pixel is Background

Step 2c: Else

Pixel is Foreground

Step 3: End

Where $\delta = colordist(x_t, v_m) = \sqrt{(||x||2 - \rho)}$;

$\rho = (R_m R + G_m G + B_m B)^2 / (R^2_m + G^2_m + B^2_m)$

bright [I, {I, J}] = true If($I_{low} < ||x_t|| < I_{high}$ *)*

= *false otherwise;*

3.2 Foreground Clean-up and connected components

The outcome of the background-foreground segmentation step is a single channel grayscale image consisting of binary values. The foreground has only two values for every pixel i.e. whether it belongs to a foreground (255) or background (0). This raw segmented image is noisy and has foreground pixels spread out around the image. All these pixels may not belong to foreground, some of them may be caused due to illumination variation etc. Generally the foreground pixels cluster around the area of a foreground object and the noise pixels are sparsely located and are not clustered. The noise pixels can be eliminated by applying morphological techniques such as erosion to get rid of scarcely placed noise pixel and dilation to rebuild the area of surviving components that was lost in erosion. After the initial cleanup, a connected component analysis is applied to the foreground mask to extract regions containing the foreground objects. Connected components labeling scans the image, pixel by pixel (from top to bottom and from left to right) to identify regions of adjacent pixels which share the same intensity in case of a binary image, the

pixels with intensity 255. These regions are retrieved as contours and are filtered considering factors such as relevance of the area of the contour to the foreground objects size.

3.2.1 Morphological Operations (Dilation and Erosion)

In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as dilation or erosion [43].

Erosion: The value of the output pixel is the minimum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to 0, the output pixel is set to 0. In mathematical terms, Let 'A' and '*B*' be sets in Z^d, d > 0. Let $(B)_x$ denote the translation of '*B*' by '*x*' and let '*B*' denote the reflection of '*B*' with respect to its origin [43]. The erosion of '*A*' by '*B*', A or B, is defined as

$$A \text{ or } B = \{x |\ (B)_x \subset A \} \qquad 3.12$$

Dilation: The value of the output pixel is the maximum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to the value 1, the output pixel is set to 1. In mathematical terms, Let '*A*' and '*B*' be sets in Z^d, d > 0. Let $(B)_x$ denote the translation of '*B*' by '*x*' and let '*B*' denote the reflection of '*B*' with respect to its origin [44]. The erosion of '*A*' by '*B*', A and B, is defined as

$$A \text{ and } B = \{ x |\ (B)_x \cap A \text{ not equal to } 0 \} \qquad 3.13$$

3.2.2 Connected Components

Connected component labeling works on binary or grayscale images and different measures of connectivity are possible such as 4 – connectivity, 8 – connectivity etc. The connected components labeling operator scans the image by moving along a row until it comes to a point '*p'* (where '*p'* denotes the pixel to be labeled at any stage in the scanning process) for which *Value =1.* When this is true, it examines the four neighbors of '*p'* which have already been encountered in the scan based on this information, the labeling of '*p'* occurs as follows:

- If all the neighbors of *'p'* are of the value 0 then assign a label '*q'*
- If all the neighbors of '*p'* are of the value 1 then assign a label '*p'*
- If more than one of the neighbors of the value *1,* assign one of the labels to '*p'* and make a note of the equivalences.

After the completion of the scan, a secondary scan is made in which each label is replaced by label assigned to its equivalence classes. The foreground components has only two labels '*p*' and '*q*' belonging to foreground and background respectively and all the nearby blobs are approximated and labeled as single foreground object [45]. After the connected component labeling, the boundaries of the blobs are approximated to polygon lines or to convex hulls to a clear boundary.

CHAPTER 4

OBJECT TRACKING

4.1 Mean Shift Tracking

The mean shift algorithm is a robust statistical algorithm which finds local extrema in the probability distribution. It works with a search window that is positioned over a section of the distribution. Within this search window the local maxima is determined by a simple average computation. The search window is moved to a new position and average computation is repeated again. This procedure is repeated until the algorithm finds a local maximum and converges. Every pixel in a frame has a probability value P (u, v), depending on its color/ intensity, '*P'* which indicates how likely is the pixel related to the target object. Using the probability values a frame can be represented as a 2D probability distribution and the mean shift algorithm can be applied. Mean shift is used in color-based object tracking because it is simple and robust but, is heavily dependent on the color of the object. Sudden illumination changes, occurrence of other objects with similar color proportions, similarity of the background color to the object color pose some problems to efficiency of the algorithm.

Mean Shift Algorithm

The mean shift algorithm iteratively shifts a data point to the average data points in its neighborhood similar to clustering. Consider a set S of '*n'* data points X_i in d-D Euclidean space '*X*'. Let K(x) denote a kernel function that indicates how much '*x*' contributes to the estimation of the mean. Then, the sample mean '*m'* at '*x'* with kernel '*K*' is given by

$$m(x) = \frac{\sum_{i=1}^{n} K\,(x - xi)\,xi}{\sum_{i=1}^{n} K\,(x - xi)} \qquad 4.1$$

Where, kernel K is a function of $||X||^2$

The difference m(x) - x is called mean shift. Mean shift algorithm iteratively moves data point to its mean, in each iteration, x← m(x) and the algorithm stops when m(x) = x. The sequence x, m(x), m (m(x))… is called the trajectory of x. If sample means are computed at multiple points, after each iteration, an update is made simultaneously on all these points [47].

Mean shift Tracking

Let X_i, i = 1…n, denote pixel locations of target model centered at 0. Let $b(x_i)$ denote the color bin of the color at X_i. Assume size of model is normalized; so, kernel radius '*h*' = 1. The probability of the color '*u'* in the target model is derived by employing a convex and monotonic decreasing kernel profile '*k'* which assigns a smaller weight to the locations that are farther from the center of the target. The weighting increases the robustness of the estimation, since the peripheral pixels are the least reliable, being often affected by occlusions (clutter) or background. The radius of the kernel profile = 1, by assuming that the generic coordinates '*x*' and '*y'* are normalized with *hx* and *hy*, respectively. The probability '*q'* of color '*u'* in the model is:

$$qu = \sum_{i=1}^{n} K\,(\,||\,xi\,||2\,)\,\delta\,(\,b(xi) - u\,) \qquad 4.2$$

where, *δ* is the Kronecker delta function. The normalization constant '*C'* is derived by imposing the condition$\sum_{u=1}^{m} qu = 1$, from where

$$C = \frac{1}{\sum_{i=1}^{n} k\,(\,||\,xi\,||2\,)\,]} \qquad 4.3$$

Since the summation of delta functions for u = 1…..m is equal to 1. Target Candidates: Let Yi, i = 1….n, denote pixel location of the targets centered at 'y' in the current frame. The probability of the color 'u' in the target candidate is given by:

$$p_u (y) = C_h \sum_{i=1}^{nh} k \left(\left|\left| \frac{y - yi}{h} \right|\right| 2 \right) \delta \left(b(yi) - u \right) \qquad 4.4$$

Where, C_h is the normalization constant. The radius of the kernel profile determines the number of pixels (i.e., the scale) of the target candidate.

Tracking Algorithm:

Given the distribution { q_u } of the target model and the estimated location y of the target in the previous frame:

Step 1: Initialize the location of the target in the current frame to y, compute the distribution $\{ p_u (y) \}$ and $\rho (p(y), q)$ where, ρ is the Bhattacharya coefficient:

$$\rho (p(y), q) = \sum_{u=1}^{m} \sqrt{pu\ (y)\ qu} \qquad 4.5$$

Step 2: Apply mean shift and calculate the new location z

$$z = \frac{\sum_{i=1}^{nh} g (|| y - yi\ / h ||2)\, yi}{\sum_{i=1}^{nh} g (|| y - yi\ / h ||2)} \qquad 4.6$$

Calculate $\{ p_u (z) \}$ and $\rho (p(z), q)$

Step 3: while $\rho (p(z), q) < \rho (p(y), q)$, do z ← (y+z)/2

If || z – y || is small enough, stop. Else set y ← z and go to step 1.

In practice, a window of pixels are considered for y_i of size h. Step 3 validates the target's new location and it can stop when y and z round off to the same pixel.

Mean Shift tracking Algorithm

The main steps in the mean-shift algorithm are as follows [50]:

1. Initialize the size and position of the search window

2. Find the centre of gravity of the search window

3. Calculate the distance vector between the centre of the search window and centre of gravity and shift the search window equal to the distance vector

4. Repeat from step 2 until convergence

pseudo code

Step1: Calculate current candidate histogram, and Bhattacharyya distance between model & candidate histogram

Step2: New weights histogram with each bin = $\sqrt{\frac{value(model)}{value\ (candidate)}}$

Step3: Compute

Step 3a: m00 = sum of all weights

*Step 3b: m01 = sum of (weight * x value of pixel with this color)*

*Step 3c: m10 = sum of (weight * y value of pixel with this color)*

Step 3d: mean shift in direction x = m10 / m00 - width / 2

Step 3e: mean shift in direction y = m01 / m00 - height / 2

Step 4: Shift candidate rectangle in computed direction

Step 5: Compute histogram of shifted rectangle and Bhattacharyya distance between model and shifted histogram

Step 6: While (distance from step 1 > just computed distance)

Step 6a: Shift candidate rectangle with half mean shift

Step 6b: Compute histogram of shifted ellipse, and

Step 6c: Compute Bhattacharyya distance between model and shifted histogram

Step 7: If ((mean shift in direction x)^2 *+ (mean shift in direction y)*^{2}*) < ε then*

Step 7a: Goto Step 9

Step 8: Else

Step 8a: New candidate = shifted rectangle

Step 8b: Goto Step 1

Step 9: End

4.2 Lucas Kanade Tracking

The Lucas-Kanade (LK) method is often used to compute optical flow. The optical flow is a velocity field associated with image changes. This effect generally appears due to the relative movement between object and camera or by moving the light sources that illuminates the scene [49]. A velocity or displacement can be associated with pixels from previous frame to current frame. By measuring the associated velocity and displacement one can track the point of interest in successive frames. The LK algorithm relies only on local information that is derived from small window surrounding each of the points of interest but the disadvantage of using a small window is that large motions can move outside the local window. To overcome this, a pyramidal Lucas Kanade method was implemented, which tracks starting from highest level of an image pyramid (lowest detail) and working down to lower levels (finer detail). Tracking over image pyramids allows large motions to be caught by local windows [35].

The basic idea of Lucas Kanade algorithm rests on three assumptions:

- Brightness Constancy: A pixel from the image of an object in the scene does not change in appearance as it moves from frame to frame.

For grayscale images, this means we assume that the brightness of a pixel does not change as it is tracked from frame to frame.

- Temporal persistence: The image motion of a surface patch changes slowly in time. In practice, this means the temporal increments are fast enough relative to the scale of motion in the image that the object does not move much from frame to frame.
- Spatial coherence: Neighboring points in a scene belong to the same surface, have similar motion, and project to nearby points on the image plane.

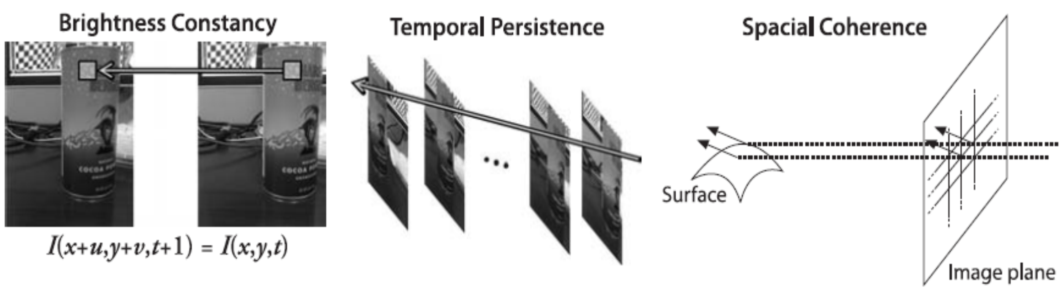


Figure 4.1 Assumption of Lucas Kanade Optical flow method

The first requirement, brightness constancy, is just the requirement that pixels in one tracked patch look the same over time:

$$f(x, t) = I(x(t), t) = I(x(t+dt), t+dt) \qquad 4.7$$

Implies that change in intensity of the pixel over time is 0 as in 4.8a and from the second assumption, change between current frame to the next frame is differentially small. By applying chain rule of partial differentiation 4.8b, *Ix* is the spatial derivative across the first image, *It* is the derivative between images over

time and V is the velocity. Therefore the associated velocity can be calculated by 4.8c.

$$\text{a. } \frac{df(x)}{dt} = 0 \quad \text{b. } \frac{\partial I}{\partial x} \mid \left(\frac{\partial x}{\partial t}\right) + \frac{\partial I}{\partial t} \mid \quad \text{c. } V = \frac{Ix}{It} \qquad 4.8$$

Consider Figure 4.2, which shows an edge moving to the right along the *x*-axis. The velocity Vat which the edge is moving is the rise over run, where the rise is over time and the run is the slope (spatial derivative).

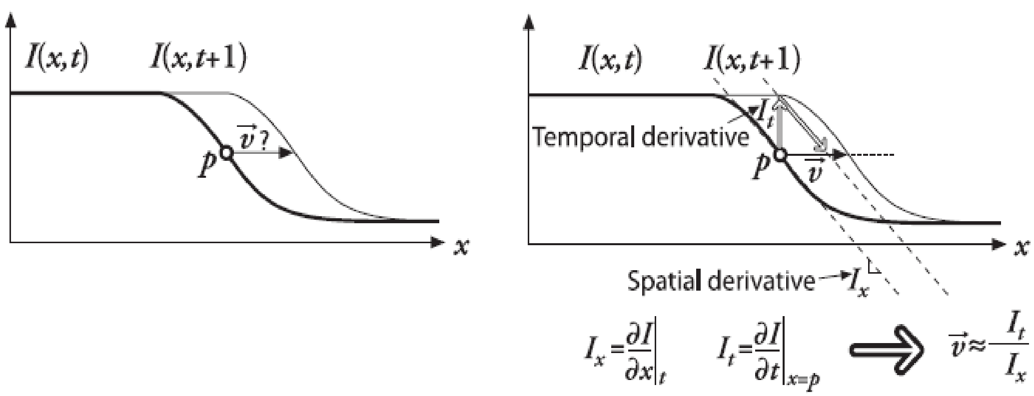


Figure 4.2 Estimate of velocity of edge

The initial assumption about brightness constancy is always not true as the brightness is not stable and time steps are often not as fast relative to the motion. This means that our solution for the velocity is not exact. But if the solution is close enough. Newton's method can be used to iterate to a solution with initial estimate of velocity as the starting point. for the next iteration and then repeated to converge to a solution. If the initial estimate is not close enough, the outcome will diverge.

Now, for a two dimensional solution, the brightness constancy assumption:

$$I_x u + I_y v + I_t = 0 \tag{4.9}$$

In this equation there are two unknowns, and hence a unique solution cannot be obtained for a 2-d motion at that point. This can be solved for the motion component only normal to the line described by the equation. The Normal optical flow leads to aperture problem, which occurs as the flow component only in the gradient direction can be determined. The motion parallel to the edge cannot be determined. To overcome this problem, more constraints are required such as optical flow changes smoothly locally, that (u, v) is constant within small neighborhood of a pixel i.e. if a local patch of pixels move coherently, motion of the central pixel can be calculated using a system of equation of the surrounding pixels. The Lucas Kanade tracking algorithm gives a good performance when the tracking window is centered over a corner region of an object [35]. The algorithms cannot track large motions. To overcome this problem, objects are tracked over large spatial scales using image pyramids followed by refining the initial motion velocity assumptions by working down the levels of the image pyramid.

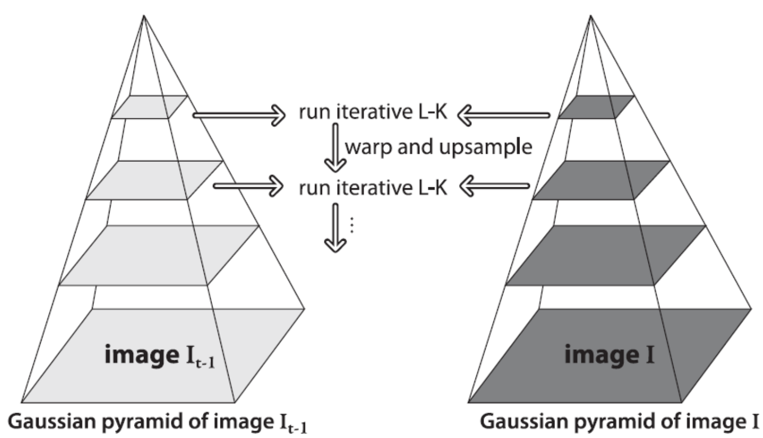


Figure 4.3 Pyramid Lucas Kanade Optical flow

The Lucas-Kanade optical flow tracking method provides a good tracking for objects for which the assumptions apply, such as considerable brightness constancy which can be overcome by Newton's method and small motion, which can be overcome by using image pyramids. This method gives a good performance when used to track corners, and hence it is used in conjunction with corner detection algorithms.

Lucas Kanade pyramidal Optical flow tracking pseudo code

The optical tracking component uses the pyramidal implementation of the Lucas-Kanade optical flow algorithm, which first identifies and then tracks features in an image. These features are pixels whose spatial gradient matrices have a large enough minimum Eigen value.

Step 1: Pre-compute the spatial derivatives Ix and Iy

Step 2: For each point i

Step 2a: Compute gradient covariance matrix, Z_i

Step 2b: Initialize $u_i = (0, 0)$

Step 2c: Repeat until convergence

Step 3: Compute I_t *from first image and shifted second image,* $I_t = I(x_i) - J(x_i+u_i)$

Step 4: Compute e_i

Step 5: Find the estimate of displacement, $v_i = Z_i^{-1} ei$

Step 6: $u_i = u_i + v_i$

Step 7: if $|| v_i || < \varepsilon_{lk}$ *(minimum displacement threshold) then Exit*

Step 8: End

Lucas Kanade Pyramidal Method

Step 1: For each feature i,

Step 1a: Initialize $u_i \leftarrow (0, 0)^T$

Step 1b: Initialize λ_l

Step 2: For pyramid level $n - 1$ *to 0 step* -1,

Step 2a: For each feature i, compute Z_i

Step 3: Repeat until convergence:

Step 3a: For each feature i,

Step 3b: Determine v_i

Step 3c: Difference Image $I_t(x, y) = I_1(x, y) - I_2(x + u_i, y + v_i)$

Step 3d: Compute e_i

Step 3e: Solve $Z_i V_i = E_i$ *for incremental motion* v_i

Step 3f: Add incremental motion to overall estimate: $u_i \leftarrow u_i + v_i$

Step 4: Expand to the next level: ui ← kui, where k is the pyramid scale factor

Step 5: End

CHAPTER 5

RESULTS

This chapter presents the experimental results of the algorithms implemented for pedestrian and vehicle detection. These experiments are conducted on a series of real videos. All the video processing techniques used for pedestrian and vehicle detection are implemented using OpenCV, a C-language based library. OpenCV supports major formats for video and images and the codecs are easily available online. A major part of processing power used in video detection and tracking is wasted due to focusing on the entire image where as the objects of interest are present in a particular area of the image. Figure 5.1 shows a typical surveillance scene and its region of interest.

Figure 5.1 Region of Interest

The objective of the system is to detect objects on the cross walk. From the scene, it is clear that the majority of the video provides no information to the task. A function is created to select region of interest manually using a mouse. Every frame is extracted from the video and is cropped to the required dimensions and is written to a new video file.

5.1 Object Detection Results

There are many methods traditionally used for comparing background subtraction algorithms such as methods in [51] [52]. In scheme presented in [51], background subtraction algorithms are compared with images annotated by hand and the result is analyzed using detection theory techniques. Jacinto et.al. [52] presented standardized algorithms to evaluate background subtraction algorithms.

Object Detection Algorithms implemented:

- Frame differencing
- Averaging Background
- Mixture of Gaussians
- Codebook Method

Object Detection Algorithms implemented:

- Mean Shift Algorithm
- Lucas Kanade Optical flow tracking

5.1.1 Frame Difference Results

In frame differencing algorithm, threshold is a key value to control the amount of noise and good detection rate. For comparison, one particular frame is displayed with various threshold values ranging from 10 to 100. When the threshold is low, considerable value from the background pixels are considered as foreground and if the threshold is high, information from the foreground objects are not detected.

Figure 5.2, displays images with different threshold values. Figure 5.2 (i - 10, ii - 20, iii - 30) shows the low threshold values, where considerable noise appears on the foreground. Figure 5.2 (iv - 50, v - 60, vi - 100) displays images with high thresholds where the object is not detected properly. Figure 5.2 iii shows an optimum threshold value has a detection and low noise and hence is chosen as a candidate for comparison with other background subtraction algorithms.

As discussed in Chapter 3, occurrences of holes within the foreground object is prominent and even with good threshold values, holes are present. To overcome this problem, foreground clean up and connected component analysis is applied on the foreground mask. Variation in threshold values does not affect the time needed for execution and memory utilized.

Figure 5.3 ii shows the output of foreground clean up and connected component analysis of foreground image, Figure 5.3i is input and 5.3iii is the segmented image. From observation of different videos, threshold values between 30 and 50 showed good object detection results.

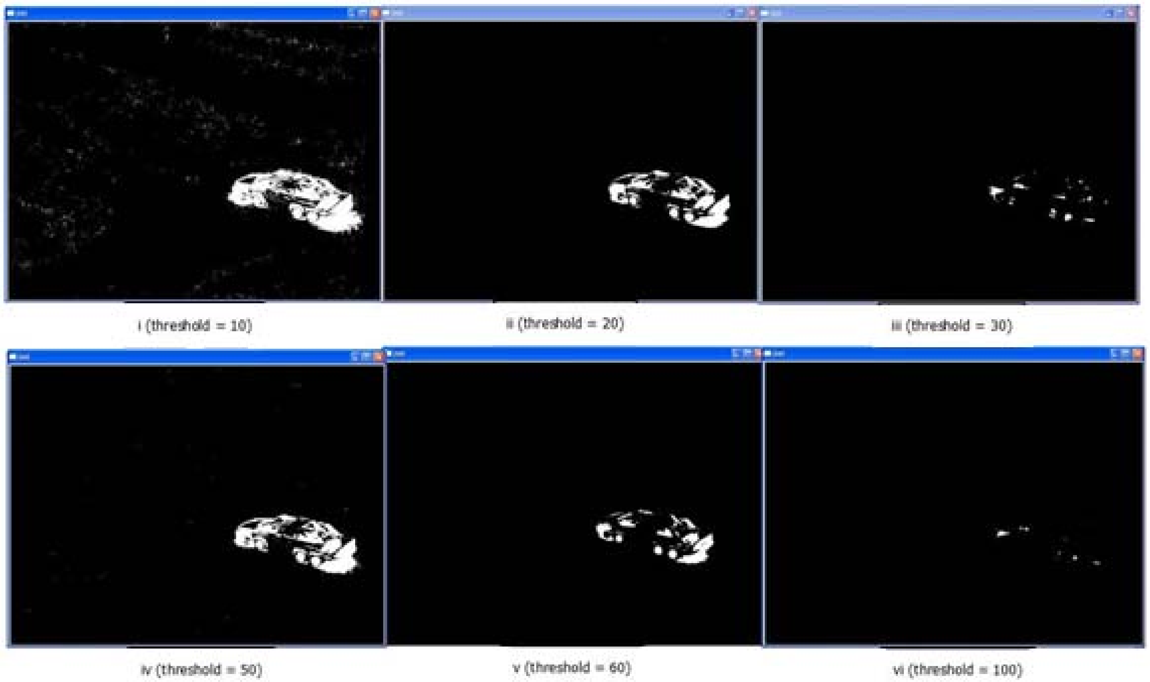


Figure 5.2 Frame Difference Threshold Values

Table 5.1 shows the system time utilized by the frame difference algorithm. The algorithm takes high initialization time and low time to segment background and foreground. Figure 5.4 shows object detection by frame difference algorithm on four different videos.

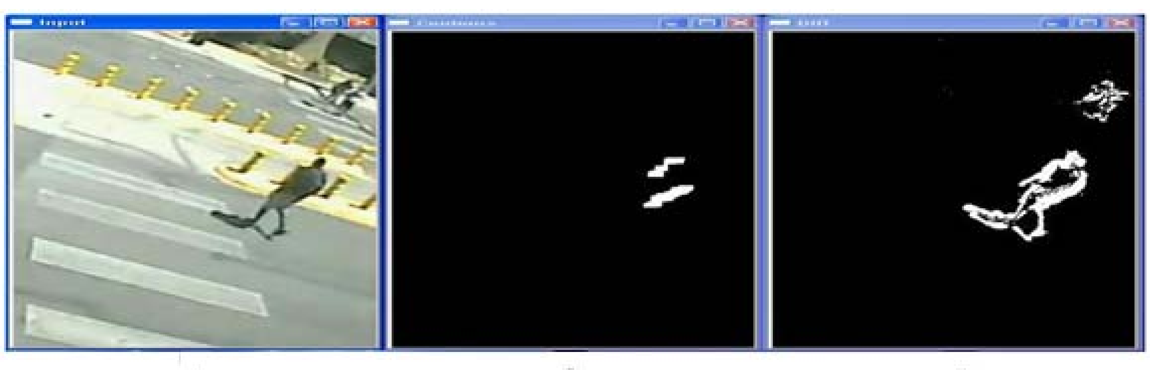


Figure 5.3 Frame Difference Connected Components

Table 5.1 Frame Difference timing results

	Video 1 Resolution 252 x 188	Video 2 Resolution 768 x 576	Video 3 Resolution 720 x 576	Video 4 Resolution 352 x 288	Average timing
Initialization time (Sec)	0.125	0.125	0.157	0.063	0.117
Time to Learn Background (Sec)	0	0	0	0	0
Time for segment Background and Foreground (Sec)	0.031	0.031	0.047	0.016	0.031
Foreground Cleanup and Connected Components Analysis (Sec)	0.016	0.047	0.062	0.016	0.035

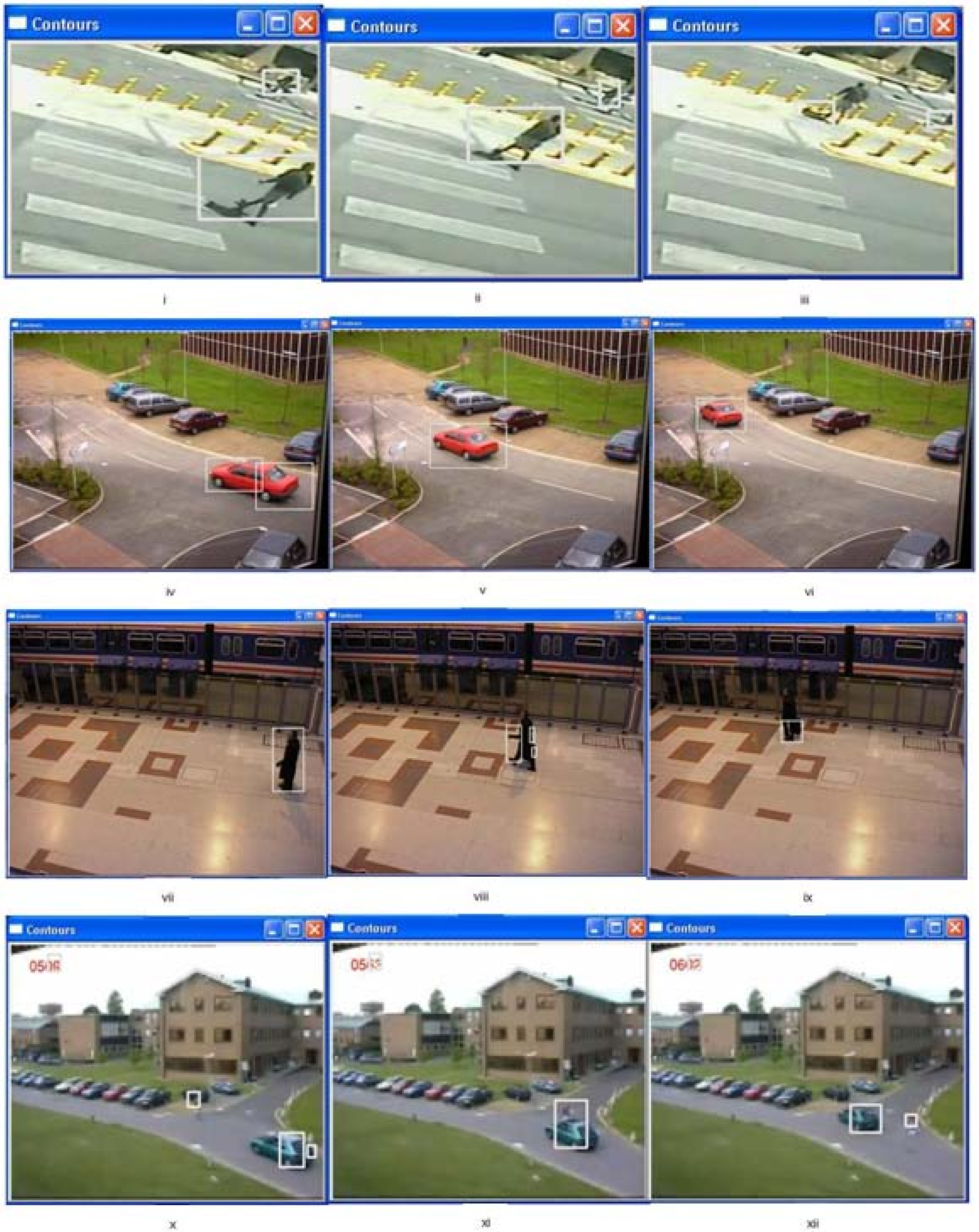


Figure 5.4: Frame Difference Object Detection

5.1.2 Background Averaging Results

Background Averaging also deals with thresholding average of pixel values over a number of frames. A technique similar to frame difference was implemented, the threshold values varying threshold and the results were as shown in Figure 5.5

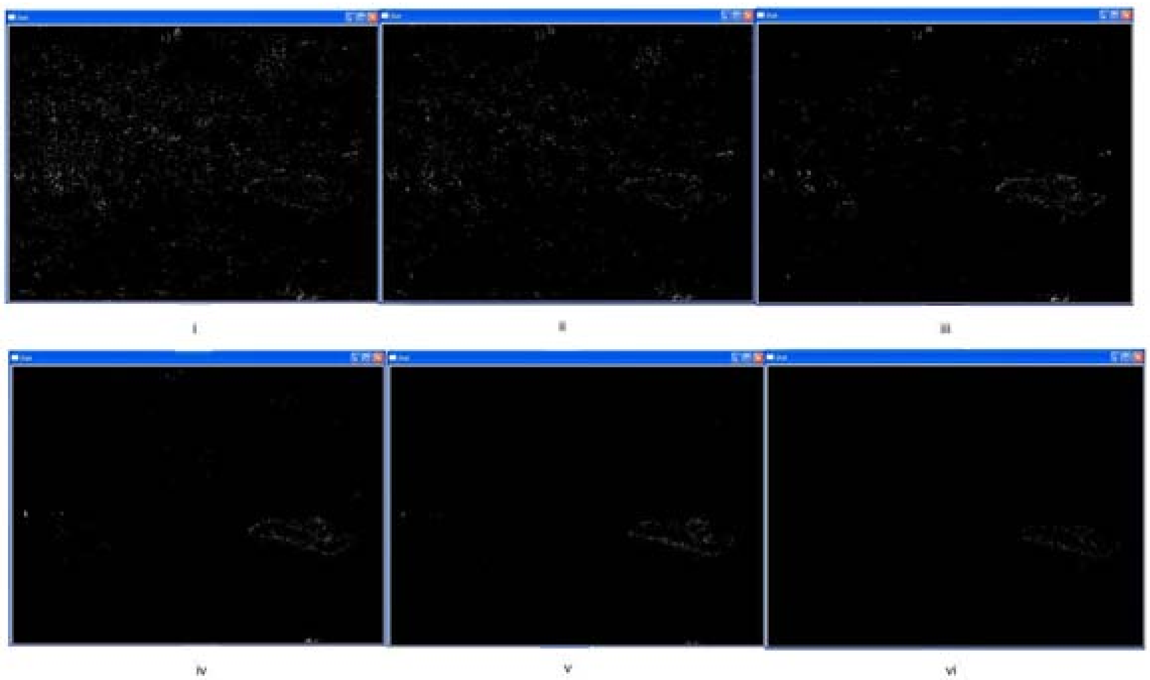
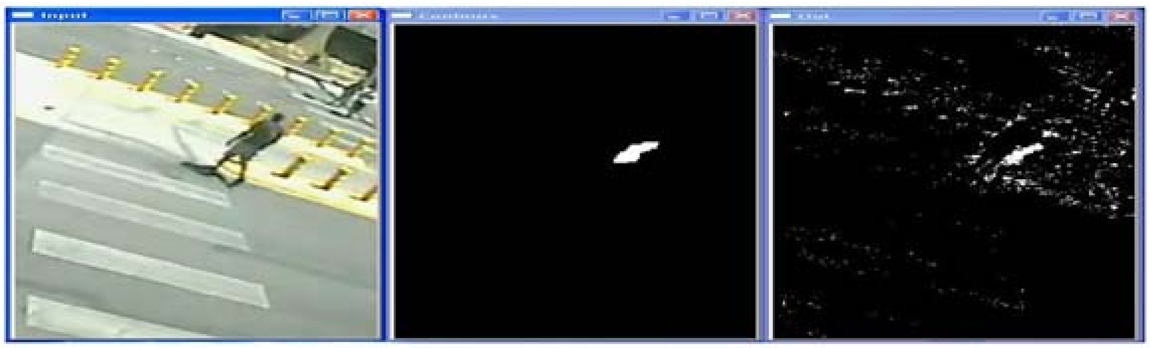


Figure 5. 5 Background Averaging Threshold values

Figure 5.5 shows images with varying threshold values. Figure 5.5 (i, ii, iii) shows images with low threshold values where considerable amount of noise appears and the figure 5.5 iv, v, vi shows high threshold values, even in high threshold, less noise from the background appears in the foreground and the object is also not detected properly. Since, object detection is of the primary

importance, and noise can be removed up by applying morphological operations and connected components analysis.



i ii iii

Figure 5.6 Background Averaging Connected Components

Figure 5.6ii shows connected component output of background averaging algorithm. From observation, a threshold of 45 was selected for background averaging method. Table 5.2 shows the system time utilized by the background averaging algorithm.

Figure 5.7 shows object detection results of background averaging method.

Table 5.2 Background Averaging Results

	Video 1 Resolution 252 x 188	Video 2 Resolution 768 x 576	Video 3 Resolution 720 x 576	Video 4 Resolution 352 x 288	Average timing
Initialization time (Sec)	0.063	0.172	0.095	0.078	0.102
Time to Learn Background (Sec)	0.016	0.047	0.047	0.015	0.031
Time of Create Background Model (Sec)	0.047	0.141	0.163	0.063	0.103
Time for segment Background and Foreground (Sec)	0.032	0.094	0.141	0.031	0.074
Foreground Cleanup and Connected Components Analysis (Sec)	0.016	0.047	0.047	0.016	0.031

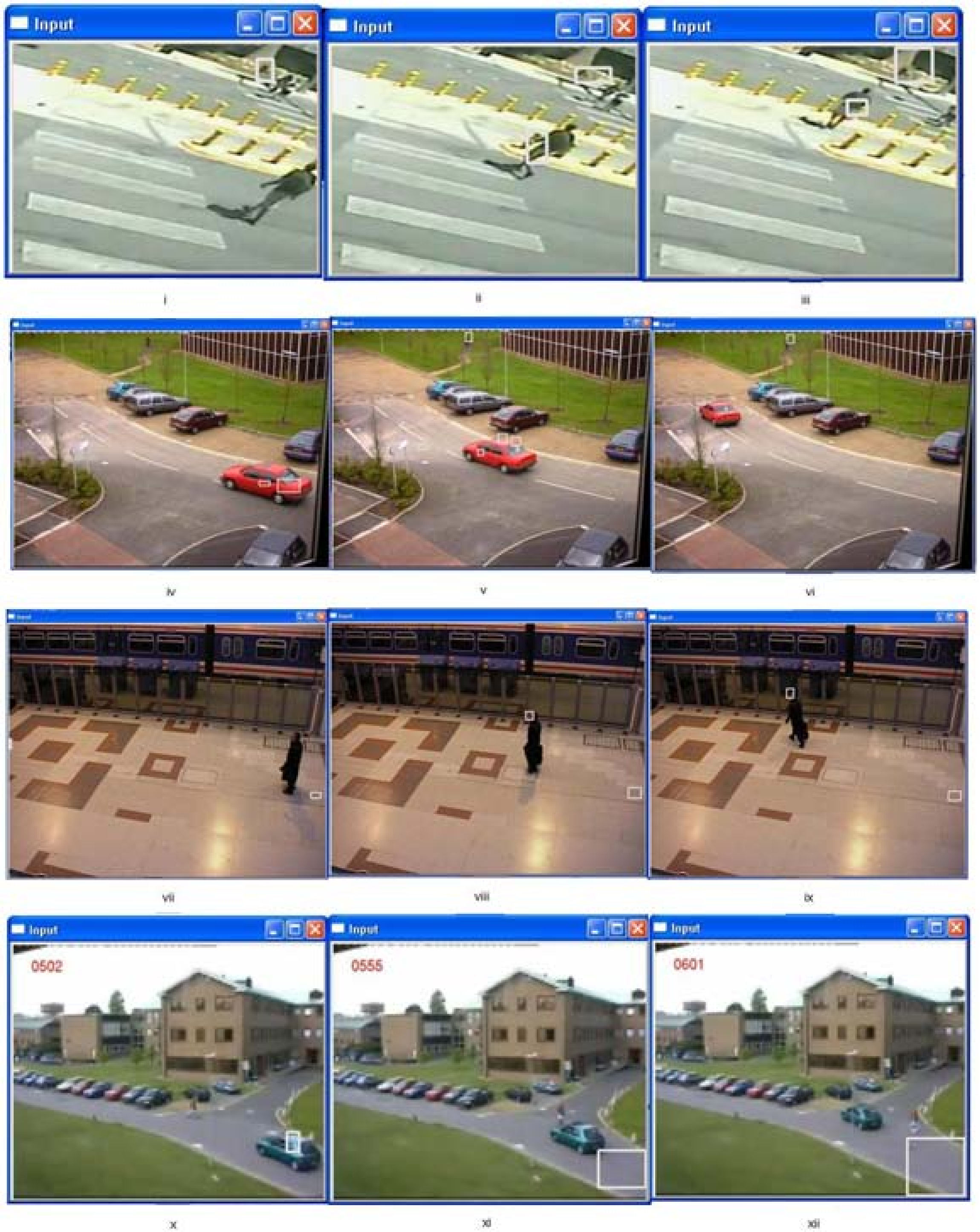


Figure 5.7 Background Averaging Object Detection

5.1.3 Mixture of Gaussians Results

The mixture of Gaussians method has many parameters such as number of Gaussians (k), window size (mxm), background threshold (BgThr) and standard deviation threshold (StdDevThr). Figure 5.8 displays variation of different parameters that affect the foreground of the algorithm.

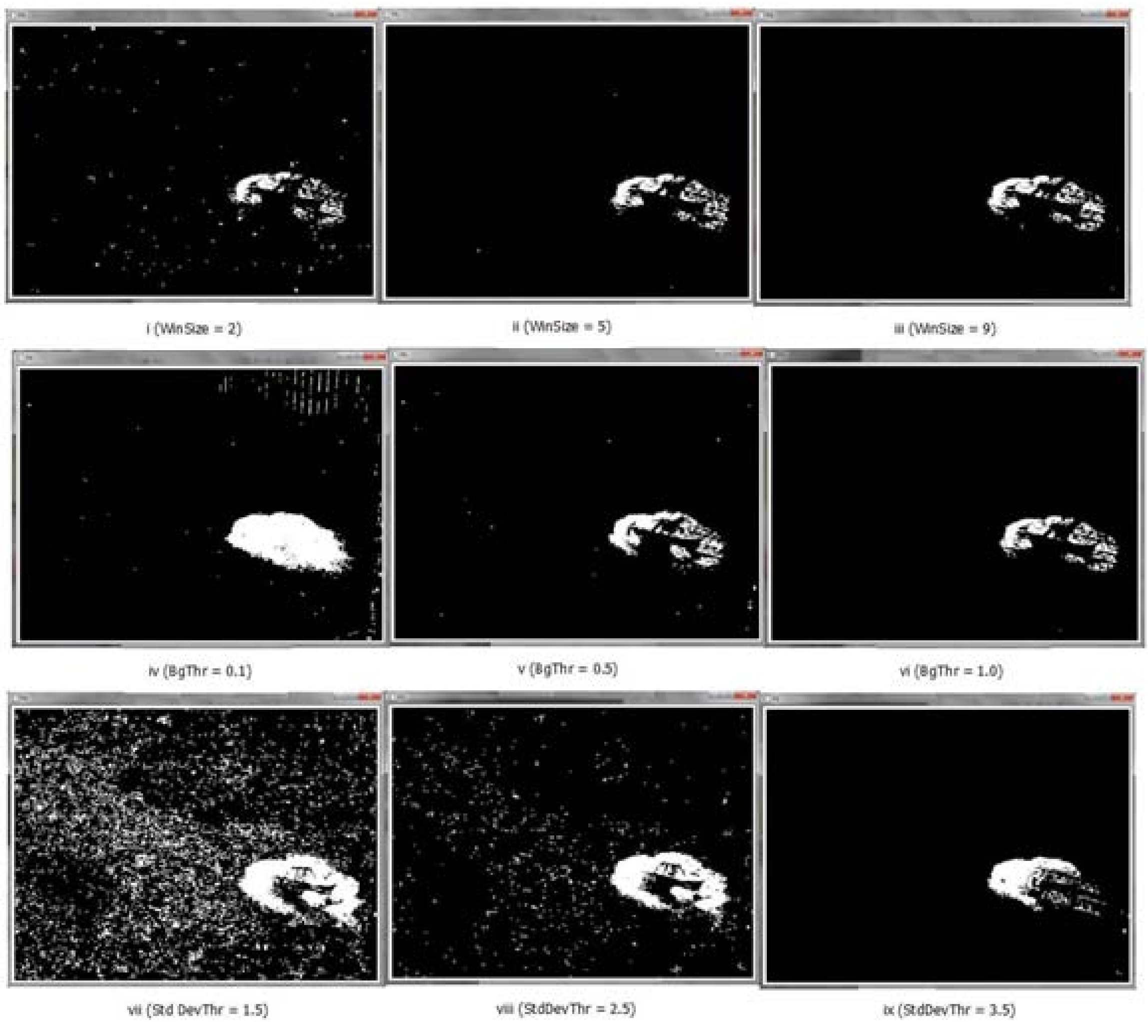


Figure 5.8 Mixture of Gaussians Background Foreground Segmentation

Figure 5.8 shows results of variation of window size, background threshold, standard deviation for mixture of Gaussians foreground detection algorithms. Figure 5.8 i, ii, iii show changes in window sizes of 2, 5 and 9. Figure 5.8 iv, v, vi shows variations of background threshold (BgThr) figure 5.8 iv has a threshold 0.1 which gives the full object, but has large amount of noise, figure 5.8 v has a threshold 0.5 which has less noise but the object is not completely detected and the figure 5.8 vi has a threshold of 1.0 which has low noise but most part of the object is lost. Figure 5.8 vii, viii, ix show changes in standard deviation threshold (StdDevThr) from 1.5, 2.5 and 3.5.

Figure 5.9ii shows connected component analysis and foreground cleanup results for mixture of Gaussians:

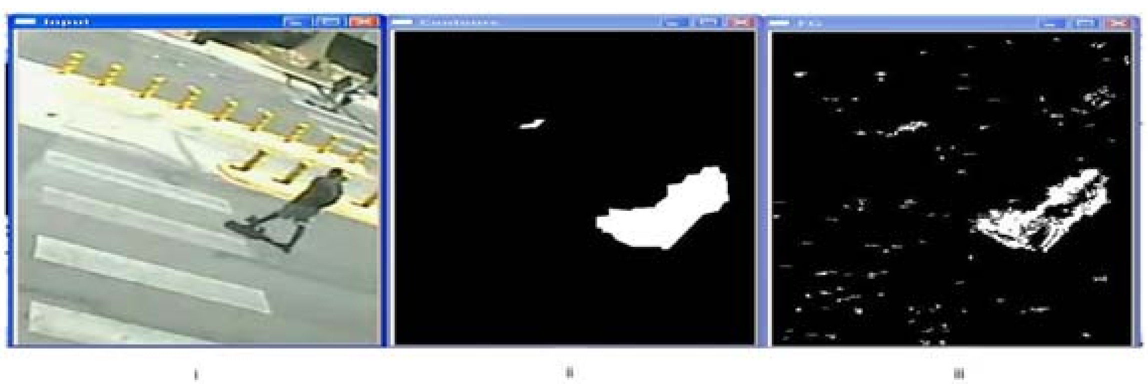


Figure 5.9 Mixture of Gaussians Connected Components

From observation, window size of 3, background threshold (BgThr) of 0.4 and standard deviation threshold (StdDevThr) of 3.5 are used to get good detection.

Table 5.3 shows Mixture of Gaussians timing results, mixture of Gaussians method takes 0.179 seconds of initialization time, and 0.226 seconds to segment

background and foreground on an image. Figure 5.10 shows the output mixture of Gaussians object detection on four different videos.

Table 5.3 Mixture of Gaussians Results

	Video 1 Resolution 252 x 188	Video 2 Resolution 768 x 576	Video 3 Resolution 720 x 576	Video 4 Resolution 352 x 288	Average timing
Initialization time (Sec)	0.125	0.25	0.265	0.078	0.179
Time to Learn Background (Sec)	0	0	0	0	0
Time for segment Background and Foreground (Sec)	0.156	0.328	0.344	0.078	0.226
Foreground Cleanup and Connected Components Analysis (Sec)	0.016	0.047	0.047	0.016	0.031

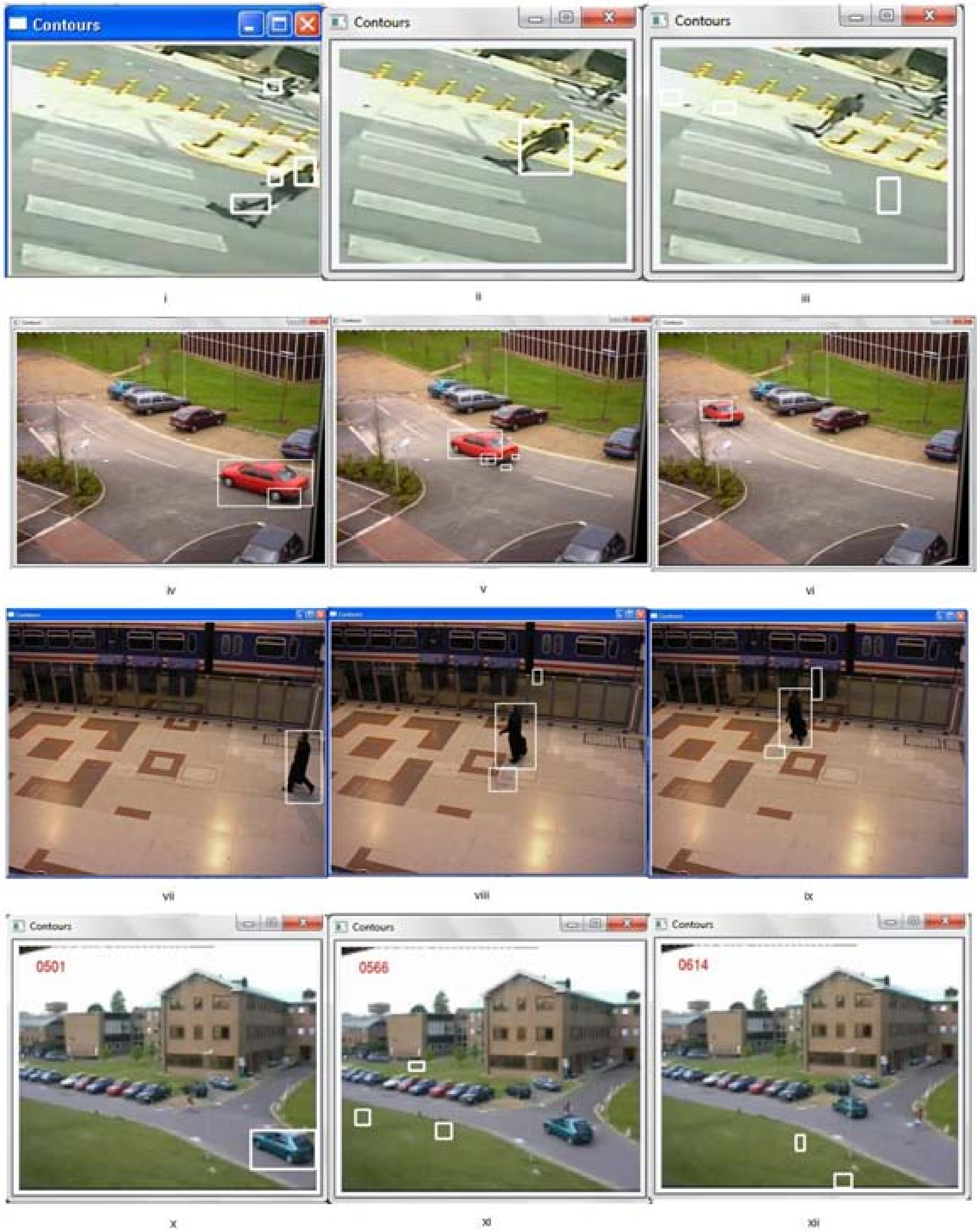
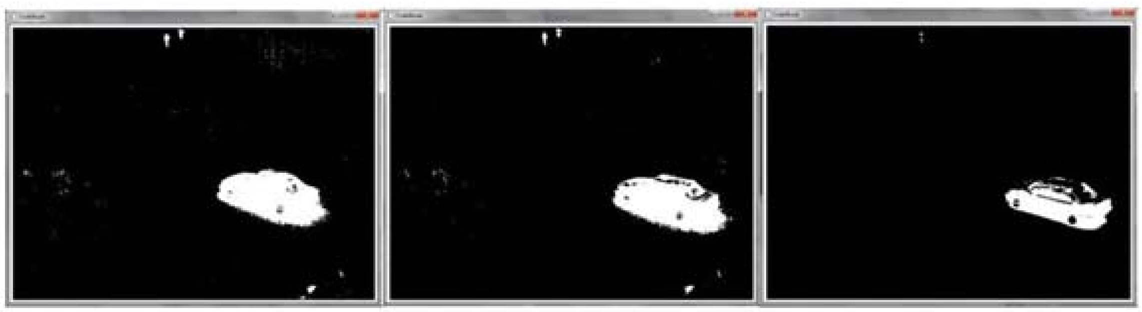


Figure 5.10 Mixture of Gaussians Object Detection

5.1.4 Codebook Method Results

The codebook method as explained in Chapter 3 depends on high and low thresholds over each color axes for the codebook selection. If each new pixel value falls within this range of learning threshold the codebooks. The variation of the threshold values is shown in the figure 5.8:



i ii iii

Figure 5.11 Codebook Background Foreground Segmentation

The parameters are varied and figure 5.8 shows changes in threshold values for the codebook algorithm. Figure 5.8 i shows low minimum and maximum threshold for each codebook, where some noise comes into the image but the objects are detected completely. Figure 5.8 ii shows low minimum and high maximum thresholds, which also has some noise but some of the object is lost. Figure 5.8 iii shows high minimum and maximum thresholds which has minimal noise, but some part of the object is lost.

Figure 5.12 shows foreground cleanup and connected component output of codebook foreground image.

i ii iii

Figure 5.12 Codebook Method Connected Components

In the codebook algorithm, a minimum threshold of 15 and maximum threshold of 30 were used to get good detection. Table 5.4 shows Codebook Method timing performance results. The codebook algorithm takes 0.136 seconds for initialization and 0.023 seconds to segment background and foreground.

Figure 5.13 shows codebook method object detection results on four different videos.

Table 5.4 Codebook Method Results

	Video 1 Resolution 252 x 188	Video 2 Resolution 768 x 576	Video 3 Resolution 720 x 576	Video 4 Resolution 352 x 288	Average timing
Initialization time (Sec)	0.125	0.25	0.125	0.047	0.136
Time to Learn Background (Sec)	0.015	0.047	0.031	0.016	0.027
Time of Create Background Model (Sec)	0.016	0.016	0.016	0.015	0.016
Time for segment Background and Foreground (Sec)	0.016	0.031	0.031	0.015	0.023
Foreground Cleanup and Connected Components Analysis (Sec)	0.016	0.047	0.063	0.016	0.031

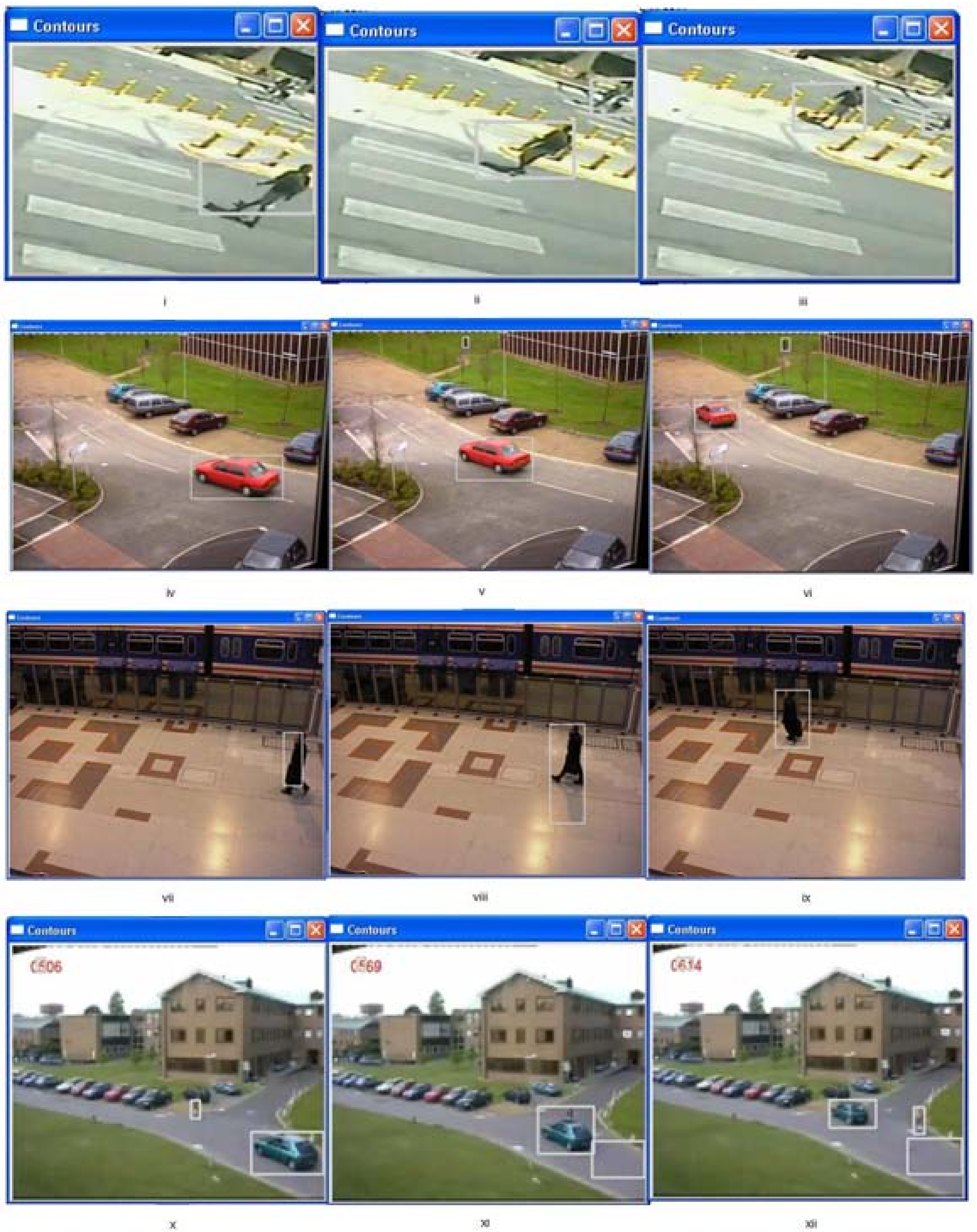


Figure 5.13 Codebook Method Object Detection

5.2 Object Tracking Results

In this thesis, mean shift tracking and Lucas Kannade pyramidal object tracking methods were implemented to track the objects on the same set of videos used for object detection. Object tracking is used to keep track of objects moving in the consequent frames and used to validate the object detection.

5.2.1 MeanShift Tracking Results

Mean shift tracking is a histogram based tracking method that estimates the position of the object based on matching the histograms of the target object in previous and the current frame. A rectangular window is defined on the object being tracked in the initial frame. In the consequent frames, the rectangular object is tracked using Mean Shift algorithm. Figure 5.14 shows the object being tracked in consequent frames. Screen shots were taken of object being tracked for every 10 frames. Table 5.5 shows Mean Shift Tracking timing performance results. The algorithm takes 0.098 seconds for initialization and 0.042 seconds on an average to track the objects.

Table 5.5 Mean Shift Tracking Results

	Video 1 Resolution 252 x 188	Video 2 Resolution 768 x 576	Video 3 Resolution 720 x 576	Video 4 Resolution 352 x 288	Average timing
Initialization time (Sec)	0.047	0.141	0.141	0.063	0.098
Time to Track (Sec)	0.031	0.062	0.062	0.016	0.042

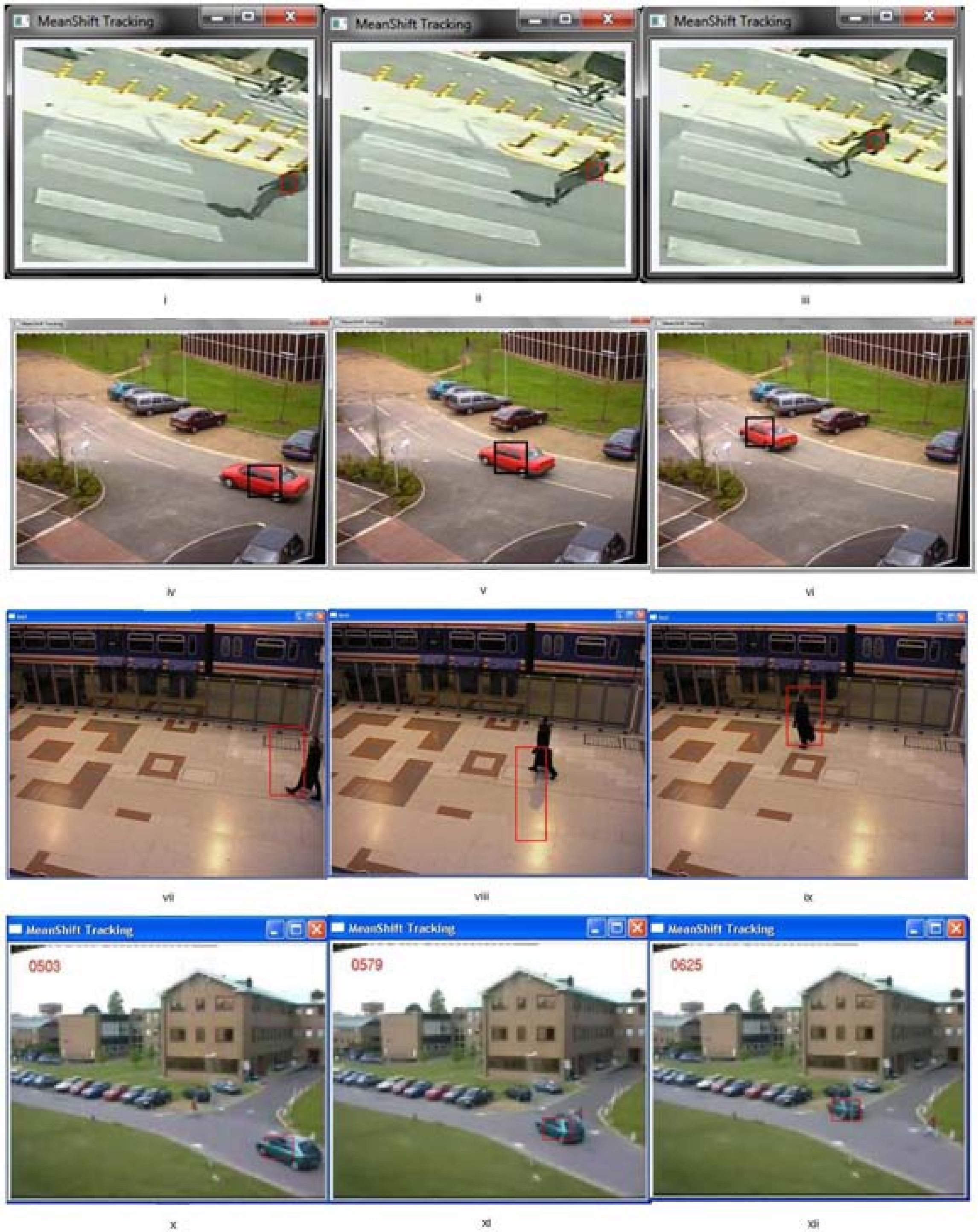


Figure 5.14 Mean Shift Object Tracking

5.2.2 Lucas Kannade Pyramidal Optical flow Tracking Results

Figure 5.11 shows the object tracking of lucas kannade optical flow tracking. The object is detected and good features (corners) to track are calculated using algorithm in [28] and these features are tracked in the consequent frames. Figure 5.15 shows the object being tracked every 10 frames. Table 5.6 shows Lucas Kanade Optical Flow Tracking timing results. The algorithm takes 0.558 seconds for initialization and 0.082 seconds to track the objects

Table 5.6 Lucas Kanade Optical Flow Tracking Results

	Video 1 Resolution 252 / 188	Video 2 Resolution 768 / 576	Video 3 Resolution 720 / 576	Video 4 Resolution 352 / 288	Average timing
Initialization time (Sec)	0.375	0.688	0.738	0.431	0.558
Time to Track (Sec)	0.063	0.094	0.121	0.063	0.082

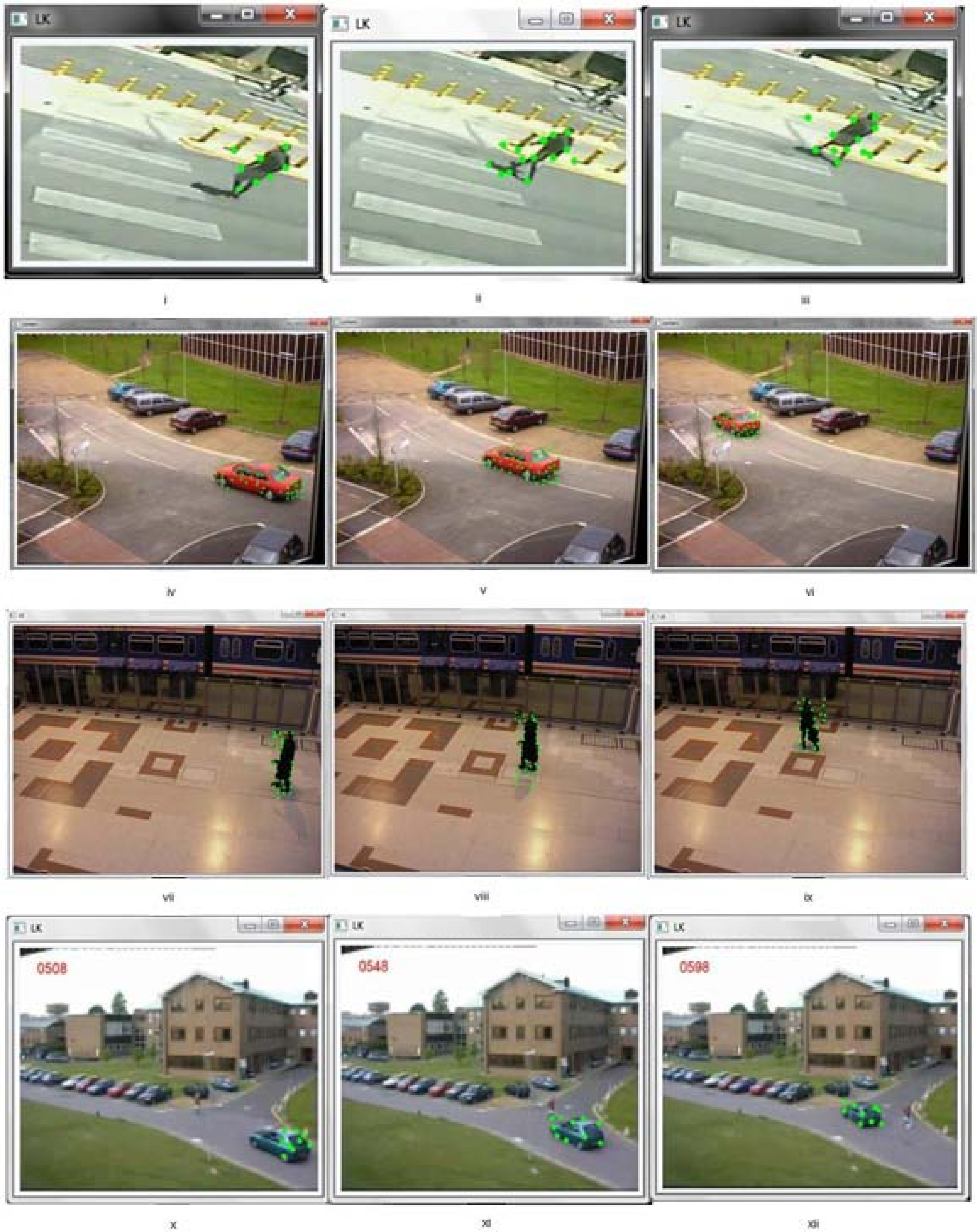


Figure 5.15 Lucas Kanade Optical flow Object Tracking

5.3 Object Detection Performance Evaluation and Analysis

Many approaches have been considered to evaluate performance of foreground detection algorithms. The objective metrics proposed in [52] consider all types of errors and are compared with manually calculated ground truth. The evaluation considers the following parameters [52]:

- Correct Detection (CD): the detected region matches one and only one region.
- False Alarm (FA): the detected region has no correspondence.
- Detection Failure (DF): the test region has no correspondence.
- Merge Region (M): the detected region is associated to several test regions.
- Split Region (S): the test region is associated to several detected regions.
- Split-Merge Region (SM): when the conditions merge and splits are simultaneously satisfied.

The object detection algorithms were evaluated for the above mentioned metrics on four videos and the results are presented in tables 5.7 and 5.8:

Table 5.7 Object Detection Results on Video 1

%	Correct Detection	False Alarm	Detection Failure	Merge	Split	Split and Merge
Frame Difference	100	25	0	37	37	37
Background Averaging	66	75	37	12	12	0
Mixture of Gaussians	100	87	0	25	50	37
Codebook Method	100	0	0	37	0	0

Table 5.8 Object Detection Results on Video 2

%	Correct Detection	False Alarm	Detection Failure	Merge	Split	Split and Merge
Frame Difference	100	33	50	16	66	33
Background Averaging	50	66	50	33	16	0
Mixture of Gaussians	83	83	16	33	66	66
Codebook Method	100	33	0	33	0	0

Table 5.9 Object Detection Results on Video3

%	Correct Detection	False Alarm	Detection Failure	Merge	Split	Split and Merge
Frame Difference	100	10	0	20	40	40
Background Averaging	50	90	50	30	40	40
Mixture of Gaussians	70	140	30	20	30	30
Codebook Method	90	10	10	30	10	10

Table 5.10 Object Detection Results on Video 4

%	Correct Detection	False Alarm	Detection Failure	Merge	Split	Split and Merge
Frame Difference	100	25	0	25	0	0
Background Averaging	50	100	50	75	0	25
Mixture of Gaussians	75	125	25	50	0	25
Codebook Method	100	24	0	25	0	0

Analyzing the output results of the various background detection algorithms, frame difference is takes less time and has good detection correct detection but has high splits and merges.

Background averaging has a low correct detection rate and high detection failure. Mixture of Gaussians method also has good detection rate but has a high false detection rate.

Codebook method has good correct detection rate, low detection failures and low merges and splits. The table 5.11 shows the memory used by each algorithm in mega bytes.

Table 5.11 Memory Utilized

	Video 1 Resolution 252 / 188	Video 2 Resolution 768 / 576	Video 3 Resolution 720 / 576	Video 4 Resolution 352 / 188	Average Memory Utilized
Frame Difference	24	30	26	14	24
Background Averaging	35	74	75	30	54
Mixture of Gaussians	54	119	108	36	80
Codebook Method	30	48	44	18	35
Mean-Shift Tracking	24	32	24	13	24
LK Optical Flow Tracking	23	32	29	18	26

From the above results, taking memory usage, speed and accuracy into account, codebook algorithm gives good object detection and Mean Shift tracking

is considered to give the good performance for video based pedestrian and vehicle detection.

CHAPTER 6

CONCLUSION

6.1 Conclusion

In this thesis, various video processing algorithms used for pedestrian and vehicle detection were implemented and evaluated. The study provides object detection by background subtraction and object tracking from a video sequence.

For background subtraction, frame differencing, background averaging, Mixture of Gaussians and Codebook method were evaluated. These algorithms were compared for accuracy, timing and memory requirements. Considering all the performance parameters, frame differencing algorithm provides very fast processing speeds taking less memory but lacks high correct detection (aperture problem), background averaging takes considerable amount of memory and has a high false detection rate. Mixture of Gaussians method is slow and takes high memory and provides many options to optimize the output, but has high rate of false detections. Codebook method gives good accuracy in segmenting the foreground and also takes less memory and time to process the image. Hence, a good trade-off is attained with the Codebook method when implemented on videos in real condition.

Two tracking algorithms have been evaluated: mean shift tracking and lucas kannade optical flow tracking. Compared to mean shift, Lucas kannade optical flow tracking is computationally expensive as it takes twice as much time to track. Thus, for object tracking the mean shift algorithm is considered for tracking pedestrians and vehicles.

Considering the performance analysis, CodeBook algorithm is considered as candidate background foreground algorithm for object detection and MeanShift algorithm is considered as candidate algorithm for object tracking. Using the two algorithms, we implemented a system that detects pedestrians and vehicles efficiently in real time.

.

REFERENCES

1. P. Dollar, B. Babenko, S. Belongie, P. Perona, and Z. Tu. “Multiple component learning for object detection” Proceedings of the 10th European Conference on Computer Vision: Part II, 2008 ISBN:978-3-540-88685-3.

2. National Highway Traffic Safety Administration. Traffic Safety Facts 2007 Data: Pedestrians, DOT-HS-810-994, 2008.

3. Weiming Hu, Tieniu Tan, Liang Wang, and S. Maybank, “A survey on visual surveillance of object motion and behaviors” IEEE Transaction on Systems, man, and cybernetics part c: Applications and Reviews, VOL. 34, NO. 3, AUGUST 2004.

4. C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz, “Efficient and Effective Querying by Image Content" Journal of Intelligent Information Systems Vol 3 Issue 3-4 1994.

5. Constantine P. Papageorgiou, Michael Oren, Tomaso Poggio, "A General Framework for Object Detection," iccv, pp.555, Sixth International Conference on Computer Vision (ICCV'98), 1998.

6. D M Gavrila, "Multi-Feature Hierarchical Template Matching Using Distance Transforms," icpr, vol. 1, pp.439, 14th International Conference on Pattern Recognition (ICPR'98) - Volume 1, 1998.

7. David Crandall, Pedro Felzenszwalb, Daniel Huttenlocher, "Spatial Priors for Part-Based Recognition Using Statistical Models," cvpr, vol. 1, pp.10-17, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1, 2005.

8. Paul Viola, Michael J. Jones, Daniel Snow, "Detecting Pedestrians Using Patterns of Motion and Appearance," iccv, vol. 2, pp.734, Ninth IEEE International Conference on Computer Vision (ICCV'03) - Volume 2, 2003.

9. Bo Wu, Ram Nevatia, "Simultaneous Object Detection and Segmentation by Boosting Local Shape Feature based Classifier," cvpr, pp.1-8, 2007 IEEE Conference on Computer Vision and Pattern Recognition, 2007.

10 Navneet Dalal, Bill Triggs, "Histograms of Oriented Gradients for Human Detection," cvpr, vol. 1, pp.886-893, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1, 2005.

11. A. K. Mandava, (2009) "On Board 3D Object Tracking: software and hardware solutions" unpublished Master's thesis, UNLV.

12. Rita Cucchiara, Costantino Grana, Massimo Piccardi, Andrea Prati, "Detecting Moving Objects, Ghosts, and Shadows in Video Streams," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 10, pp. 1337-1342, Oct. 2003, doi:10.1109/TPAMI.2003.1233909.

13. D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, S. Russell, " Towards Robust Automatic Traffic Scene Analysis in Real-time". Proceedings of the 33rd Conference on Decision and Control December 1994.

14. Sumer Jabri, Zoran Duric, Harry Wechsler, Azriel Rosenfeld, "Detection and Location of People in Video Images Using Adaptive Fusion of Color and Edge Information," icpr, vol. 4, pp.4627, 15th International Conference on Pattern Recognition (ICPR'00) - Volume 4, 2000.

15. Nir Friedman, Stuart Russell, "Image Segmentation in video sequences: A probalistic approach" Proc. Thirteenth Conf. on Uncertainty in Artificial Intelligence (UAI 97), 1997.

16. Ahmed Elgammal and David Harwood and Larry Davis, "Non-parametric model for background subtraction". FRAME-RATE Workshop, IEEE, 2000, 751-761.

17. Kentaro Toyama, John Krumm, Barry Brumitt, Brian Meyers, "Wallflower: Principles and Practice of Background Maintenance," iccv, vol. 1, pp.255, Seventh International Conference on Computer Vision (ICCV'99) - Volume 1, 1999.

18. Alper Yilmaz, Omar Javed, Mubarak Shah, "Object Tracking a Survey" ACM Computing Surveys, Vol. 38, No. 4, Article 13, Publication date: December 2006.

19. I. K. Sethi, Ramesh Jain, "Finding trajectories of feature points in a monocular image sequence" IEEE Transactions on Pattern Analysis And Machine Itelligence, vol. pami-9, no. 1, January 1987.

20. Cor J. Veenman, Marcel J.T. Reinders, Eric Backer, "Resolving Motion Correspondence for Densely Moving Points," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 1, pp. 54-72, Jan. 2001, doi:10.1109/34.899946.

21. Krishnan Rangarajan, Mubarak shah, “Establishing motion correspondence” Proceedings CVPR '91., IEEE Computer Society Conference 1991.

22. G.S.J. Young, R. Chellappa, "3-D Motion Estimation Using a Sequence of Noisy Stereo Images: Models, Estimation, and Uniqueness Results," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 8, pp. 735-759, Aug. 1990, doi:10.1109/34.57666.

23. Romer Rosales, Stan Sclaroff, "3D Trajectory Recovery for Tracking Multiple Objects and Trajectory Guided Recognition of Actions," cvpr, vol. 2, pp.2117, 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99) - Volume 2, 1999.

24. K. Fukunaga, L. D. Hostetler, “The estimation of gradient of a density function, with applications in pattern recognition" IEEE Transactions on Information Theory, vol. it-21, no. 1, January 1975.

25. Gary R. Bradski, "Real Time Face and Object Tracking as a Component of a Perceptual User Interface," wacv, pp.214, Fourth IEEE Workshop on Applications of Computer Vision (WACV'98), 1998.

26. B. Horn, B. Schunk, “Determining optical flow” ARTIFICAL INTELLIGENCE, 1981, 17, 185--203.

27. Bruce D. Lucas and Takeo Kanade, “An iterative image registration technique with an application in stereo vision” 1981, 674-679.

28. Jianbo Shi, Carlos Tomasi, “Good features to track” Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994, 593 - 600.

29. Alper Yilmaz, Xin Li, Mubarak Shah, "Contour-Based Object Tracking with Occlusion Handling in Video Acquired Using Mobile Cameras," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 11, pp. 1531-1536, Nov. 2004, doi:10.1109/TPAMI.2004.96.

30. A. Baddeley, “Errors in binary images and an L p version of the Hausdorff metric", 1992.

31. J. Maccormick, A. Blake, "Probalistic exclusion and partitioned sampling for multiple object tracking", International Journal of Computer Vision 39(1), 57–71, 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

32. Michael Isard, Andrew Blake, “Contour tracking by stochastic approach” Contour Tracking By Stochastic Propagation of Conditional Density, 1996, pages343--356.

33. A. M. McIvor. "Background subtraction techniques". In Proc. of Image and Vision Computing, Auckland, New Zealand, 2000.

34. N. Friedman and S. Russell, “Image segmentation in video sequences: A probabilistic approach," in Proceedings of the Thirteenth Annual 161Conference on Uncertainty in Artificial Intelligence (UAI-97), pp. 175 to 181, Morgan Kaufmann Publishers, Inc., (San Francisco, CA), 1997.

35. Gary Bradski, Adrian Kaehler, Learing OpenCV Computer Vision with OpenCV Library, O' Reilly Media Publishers, ISBN: 978-0-596-51613-0.

36. Kyungnam Kim, Thanarat H Chalidabhondgse, David Harwood , Larry Davis, "Real-time foreground–background segmentationusing codebook model", doi:10.1016/j.rti.2004.12.004.

37. Z. Hou, C. Han, "A background reconstruction algorithm based on pixel intensity classification in remote video surveillance system," Proc. of the Seventh International Conference on Information Fusion, pp. 754-759, June, 2004.

38. Jie Xia, Kian Wu, Haitao Zhai, Zhiming Cui, “Moving Vehicle Tracking Based on Double Difference and CAMShift”, Proceedings of the 2009 International Symposium on Information Processing.

39. Zhen Tang, Zhenjiang Miao, Yanli Wan, “Background Subtraction Using Running Gaussian Average and Frame Difference”, Proceedings of the 2009 International Symposium on Information Processing (ISIP’09), Huangshan, P. R. China, August 21-23, 2009, pp. 029-032.

40. Jacinto Nascimento, Jorge Margues, “Performance evaluation of object detection algorithms for video surveillance”. IEEE Transactions on Multimedia, Vol. 8, No. 4, August 2006.

41. Mohamad Hoseyn Sigari, Naser Mozayani, Hamid Reza Pourreza, “Fuzzy running average and fuzzy background subtraction: Concepts and Application” IJCSNS International Journal of Computer Science and Network S 138 ecurity, VOL.8 No.2, February 2008.

42. Chris Stauffer, W .E .L Grimson, “Adaptive background mixture models for real time tracking”, 0-7695-0I 49-4/99 1999.

43. Image Processing Tool box <http://matlab.izmiran.ru/help/toolbox/images/morph3.html>

44. J. Rodrigues, D. Ayala “Erosion and dilation on 2D and 3D Digital images: A new size independent approach”, VMV 2001 Stuttgart, Germany, November 21–23, 2001.

45. Connected Component Labeling<http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm>

46. Comaniciu, D.; Ramesh, V.; Meer, P.; , "Real-time tracking of non-rigid objects using mean shift," Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on , vol.2, no., pp.142-149 vol.2, 2000. doi: 10.1109/CVPR.2000.854761.

47. Changjiang Yang, Ramani Duraiswami, Larry Davis, “Efficient Mean shift tracking via a new similarity measure”. Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) 1063-6919/05 2005.

48. Yizong Cheng “ Mean shift, mode seeking and clustering”. IEEE Transaction on pattern analysis and machine intelligence, vol. 17, NO. 8, AUGUST 1995.

49. Kelson R. T. Aires, Andre M. Santana, Adelardo A. D. Medeiros “ Optical flow using color information preliminary results”. SAC’08 March 1620, 2008, Fortaleza, Cear´a, Brazil Copyright 2008 ACM 9781595937537/08/0003.

50. F. Dadgostar, A. Sarrafzadeh, and S. P. Overmyer, "Face Tracking Using Mean-Shift Algorithm: A Fuzzy Approach for Boundary Detection," presented at Affective Computing and Intelligent Interaction, Beiging, China, 2005.

51. Abdou, I.E.; Pratt, W.K.; , "Quantitative design and evaluation of enhancement / thresholding edge detectors," Proceedings of the IEEE , vol.67, no.5, pp. 753- 763, May 1979.

52. J. Nascimento and J. S. Marques. “New Performance Evaluation Metrics for Object Detection Algorithms”. In IEEE Workshop on Performance Analysis of Video Surveillance and Tracking (PETS’2004), May 2004.

54. D. Gonzales Ortega, F. J. Diaz Pernas, M. Zarzuela, M. Anton Rodriguez, J. F. Diez Higuera and D. Boto Giralda, “Real time Nose Detection and Tracking based on Adaboost and optical flow algorithms”. Lecture Notes in Computer Science, 2009, Volume 5788/2009, 142-150, DOI: 10.1007/978-3-642-04394-9_18.

55. S.S. Beuchemin, J.L. Barron, The computation of optical flow, ACM Computing Surveys 27 (1995) 433–467.

56. Jean Yves Bouguet, “Pyramidal Implementation of Lucas Kanade Feature Tracker Description of the Algorithm” Intel Corporation Microprocessor Research Labs, 2000.

57. Lihi Zelnik Manor, “The Optical Flow Field” CalTech October 2004.

58. Carlo Tomasi, Takeo Kanade, “Detection and Tracking of Point Features” International Journal of Computer Vision,1991.

59. Radgui, A., Demonceaux, C., Mouaddib, E., Aboutajdine, D., and Rziza, M. (2008). An adapted lucas-kanade's method for optical flow estimation in catadioptric images. In Proceedings of the eighth workshop on Omnidirectional Vision.

60. D. Comaniciu, P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 603-619, May 2002, doi:10.1109/34.1000236.

61. Leow Wee Kheng, “Mean shift Tracking – Computer vision and pattern recognition” CS 4243, Department of Computer Science, School of Computing, National University of Singapore.

62. Gary R. Bradski, "Real Time Face and Object Tracking as a Component of a Perceptual User Interface," wacv, pp.214, Fourth IEEE Workshop on Applications of Computer Vision (WACV'98), 1998”.

63. Yunchu Zhang, Zize Liang, Zengguang Hou, Hongming Wang, Min Tan, “An Adaptive Gaussian Background Model with Onlne background Reconstruction and Adjustable Foreground Mergence Time for Motion Segmentation” Industrial Technology, ICIT 2005. IEEE International Conference 2005, page(s): 23 - 27 ISBN: 0-7803-9484-4.

64. Liyuan Li, Weimin Huang, Irene Y. H. Gu, Qi Tian, “Foreground Object Detection from videos containing Complex Background” Proceedings of the eleventh ACM international conference on Multimedia, Pages: 2 - 10 , 2003 ISBN:1-58113-722-2.

65. Z. Hou, C. Han, A background reconstruction algorithm based on pixel intensity classification in remote video surveillance system, Proceedings of the 7th International Conference on Information Fusion (Fusion 2004), Stockholm, Sweden, June 28 - July 1, 2004, International Society of Information Fusion (ISIF), 754-759.

66. Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, Alex Paul Pentland, "Pfinder: Real-Time Tracking of the Human Body," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 780-785, July 1997, doi:10.1109/34.598236.

67. Janne Heikkilä, Olli Silvén, "A Real-Time System for Monitoring of Cyclists and Pedestrians," vs, pp.74, Second IEEE Workshop on Visual Surveillance, 1999.

70. Neeraj K. Kanhere, Shrinivas J. Pundlik, Stanley T. Birchfield, "Vehicle Segmentation and Tracking from a Low-Angle Off-Axis Camera," cvpr, vol. 2, pp.1152-1157, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2, 2005.

71. D. Comaniciu, P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 603-619, May 2002, doi:10.1109/34.1000236.

72. Allan D. Jepson, David J. Fleet, Thomas F. El-Maraghi, "Robust Online Appearance Models for Visual Tracking," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 10, pp. 1296-1311, Oct. 2003, doi:10.1109/TPAMI.2003.1233903.

73. Leibe, B., Leonardis, A., & Schiele, B. Combined object categorization and segmentation with an implicit shape model. In ECCV'04 workshop on statistical learning in computer vision.

74. David Beyemer and Kurt Konolige, "Real Time Tracking of Multiple People Using Stereo" In Proc. Of IEE International Workshop on Intelligent Environments ,2005.

75. Zhong Guo, "Object Detection and Tracking in Video", Department of Computer Science, Kent State University, November 2001.

76. N. Paragios and G. Tziritas,Adaptive Detection and Localization of Moving Objects in Image Sequences.Signal Processing: Image Comm., vol. 14, pp. 277-296, 1999.

77. Carlos Vazquez, Mohammed Ghazal, Aishy Amer, “Feature-based detection and correction of occlusions and split of video objects” 2008, SIViP (2009) 3:13–25 DOI 0.1007/s11760-008-0055-6.

VITA

Graduate College
University of Nevada Las Vegas

Varun Bandarupalli

Degree:
Bachelor of Technology, Electronics and Instrumentation Engineering, 2007,
Jawaharlal Nehru Technological University, India

Thesis Title: Evaluation of Pedestrian and Vehicle detection Algorithms

Thesis Examination Committee:
Chairperson, Dr. Muthukumar Venkatesan, Ph.D
Committee Member, Dr. Emma E. Regentova, Ph.D.
Committee Member, Dr. Paulo Ginobbi, Ph.D.
Graduate College Representative, Dr. Laxmi Gewali , Ph.D.