# Automatic Tracking of Moving Objects in Video for Surveillance Applications

*Manjunath Narayana*


Submitted to the Department of Electrical Engineering & Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master's of Science

**Thesis Committee:**

Dr. Donna Haverkamp: Chairperson

Dr. Arvin Agah

Dr. James Miller

**18/July/2007**

Date Defended



**07/18/2007**

The Thesis Committee for Manjunath Narayana certifies
that this is the approved version of the following thesis:

**Automatic Tracking of Moving Objects in Video for Surveillance Applications**

Committee:

Dr. Donna Haverkamp (Chair)

Dr. Arvin Agah

Dr. James Miller

**18/July/2007**

Date Approved

# Abstract

Automated surveillance systems are of critical importance for the field of security. The task of reliably detecting and tracking moving objects in surveillance video, which forms a basis for higher level intelligence applications, has many open questions. Our work focuses on developing a framework to detect moving objects and generate reliable tracks from real-world surveillance video. After setting up a basic system that can serve as a platform for further automatic tracking research, we tackle the question of variation in distances between the camera and the objects in different parts of the scene (object depth) in surveillance videos. A feedback-based solution to automatically learn the distance variation in static-camera video scenes is implemented, based on object motion in different parts of the scene. The solution, based on a concept that we call 'Vicinity Factor', is robust to noise and object segmentation problems. The Vicinity Factor can also be applied to estimate spatial thresholds and object size in different parts of the scene. Further, a new Bayesian algorithm to assign tracks to objects in the video is developed. The Bayesian method allows for probabilistic track assignment and can be the basis for future higher-level inference.

## Acknowledgements

My deepest appreciation and thanks to my mother, Girijamma, for her love and all the sacrifices she has made towards my life and education. If not for her efforts, I would not be who I am.

My most sincere thanks to my advisor, Dr. Donna Haverkamp, for all her guidance and efforts towards my research and thesis. I must thank her for painstakingly reviewing my thesis and publications, and offering invaluable suggestions towards improving them. My appreciation also for her support and encouragement in my future career plans.

Thanks to Dr. Arvin Agah for his guidance during the numerous occasions when I have approached him for advice on my future plans in the field of Artificial Intelligence and also for his inputs to this thesis as a committee member. Dr. James Miller, for his critical suggestions and valuable inputs in the thesis. Dr. Xue-wen Chen for first getting me started with research and helping me on the learning curve towards the field of Artificial Intelligence. Dr. Trevor Sorensen of the Aerospace department, who I worked with in the KUHABS research project, for his help and recommendation letters for my PhD admission process. Thanks to Dr. John Gauch for helping me with his KUIM image processing code that I used as the base for my research.

Thanks to my eldest brother, Bhanuprakash, for always being there when I needed help and support. At all the important stages in my education, in the most crucial of times, I can remember his presence by my side. My middle brother and my mentor, Gopinath, is a source of strength and motivation. My professional goals have always been shaped after discussions with him.

My appreciation for the love and care that my sisters-in-law, Savithri and

Jaya, have always showered. To the kids of my family - Suchi, Ved, Raksha, and Chaitu, for the unique unselfish love that only kids are capable of giving.

My deep respect and gratitude to my first *guru*, Shashi Jayaram, for guiding me early in life.

My sincere thanks to all my relatives in India who chipped in with support when I was preparing for my journey to the United States.

To all my school and college friends in India and in the United States whose company has made my formal education years very eventful and a lot of fun.

Finally, my biggest thanks to God for being the one with me through all the ups and downs of life. To Him I owe the most gratitude - for all the opportunities that have come my way, for all the successes and failures that have summed to my being here and now. *brhmaarpanam*

*To*

*Amma*

*and*

*Baba*

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

The area of automated surveillance systems is currently of immense interest due to its implications in the field of security. Surveillance of vehicular traffic and human activities offers a context for the extraction of significant information such as scene motion and traffic statistics, object classification, human identification, anomaly detection, as well as the analysis of interactions between vehicles, between humans, or between vehicles and humans. A wide range of research possibilities are open in relation to visual surveillance and tracking.

Some useful applications of tracking include recognition of interactions between humans [1], identification of people using gait analysis [2], [3], and recognition of specific human activities - for instance, the type of tennis stroke being played [4]. Some applications involve both humans and vehicles at the same time, such as simultaneous tracking of humans and vehicles [5]. The efficacy of object tracking in non-surveillance applications is demonstrated by applications such as animal detection and identification in videos [6]. A key step in all of the above is the

ability to track a particular object or objects in successive frames. In our research, the primary task is to track vehicles in static camera video as a precursor to future research in intelligent analysis of traffic patterns and detection of anomalous behavior by vehicles on the road.

## 1.2 Goals

Automatic tracking of objects can be the foundation for many interesting applications. An accurate and efficient tracking capability at the heart of such a system is essential for building higher level vision-based intelligence. Tracking is not a trivial task given the non-deterministic nature of the subjects, their motion, and the image capture process itself. The goal of the work documented in this thesis is two-fold: (a) to set up a system for automatic segmentation and tracking of moving objects in stationary camera video scenes at the University of Kansas, which may serve as a foundation for higher level reasoning tasks and applications, and (b) to make significant improvements in commonly used algorithms.

To achieve the first goal of setting up the tracking system, suitable datastructures and algorithms for object segmentation and tracking in image sequences have been designed in the existing KUIM image processing code. The second goal is to make the tracking system more reliable and to contribute to existing research in field of object tracking. This is achieved by (1) Use of velocity flow as an important factor in tracking (2) Automatic learning of spatial thresholds and depth information of objects in different parts of the scene and (3) Design of a new Bayesian algorithm for probabilistic tracking of objects. These additions to the basic system result in improved tracking as documented in the later sections of this thesis. The entire system has been tested on various videos and the seg-

mentation and tracking results are very encouraging. Though our current focus is on accurate automatic tracking of vehicles, the system is also capable of tracking humans. The major contribution of this thesis is establishing a baseline system for automatic tracking in image sequences by including some novel techniques unique to this thesis work.

## 1.3 System description

The process of automatic tracking of objects begins with the identification of moving objects. We use an improved background subtraction method in conjunction with a novel yet simple background model to achieve very good segmentation. Once the moving pixels are identified, it is necessary to cluster these pixels into regions, which we refer to as blobs, so that pixels belonging to a single object are grouped together. Single moving objects are often incorrectly separated into two or more subregions because of lack of connectivity between pixels, which usually occurs due to occlusion from other objects (e.g., trees). A blob merging module to merge close-by blobs is implemented. Having established individual moving objects, the next task in tracking is to achieve correspondence between the blobs in one frame and those in the next. Since we work with real-life data, the algorithm is designed to be robust to real-life tracking issues like occlusion, appearance and disappearance of objects, and abrupt change in speed, location, and orientation of objects. The robust tracking system has been satisfactorily tested on various static camera scenes involving both humans and vehicles.

A block diagram of the system is given in figure 1.1. An image sequence captured by a still camera is the input to the system. We first perform Motion Segmentation to extract moving blobs in the current frame. Some blobs that are

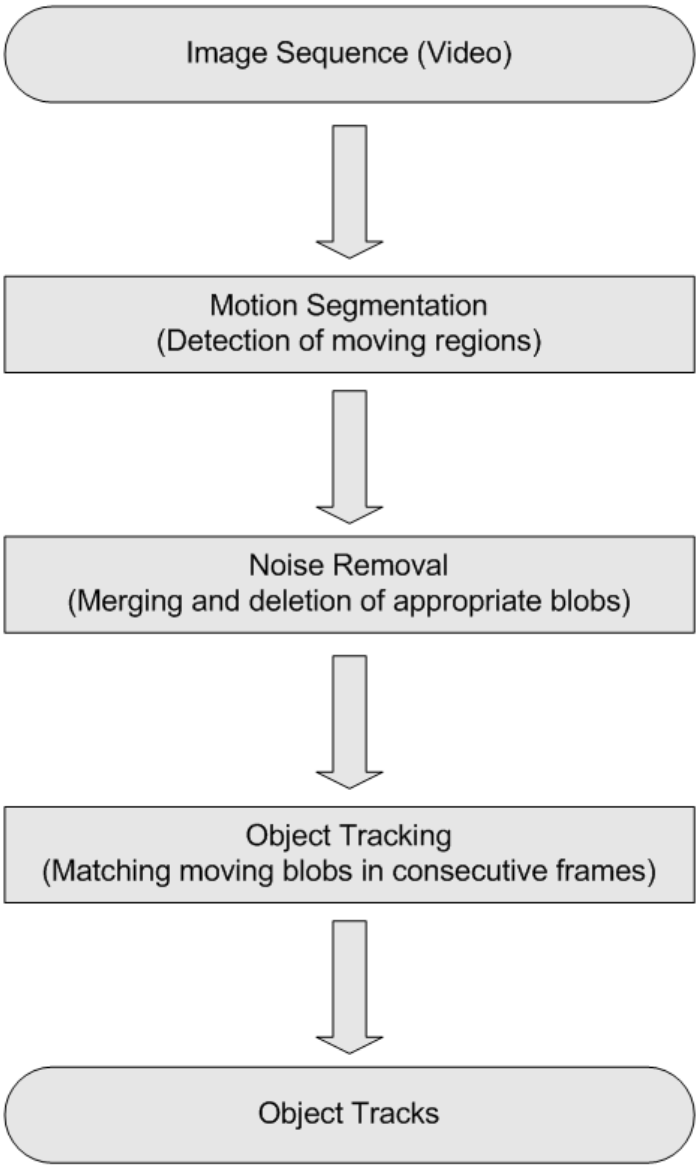



**Figure 1.1.** Block diagram of the system

very small and are likely to be noise are deleted. Due to partial occlusion, some moving objects get incorrectly segmented into two or more separate blobs. Using color and position information, such blobs are merged. The Object Tracking module tracks these moving objects over successive frames to generate Object Tracks.

Significant improvements are brought into the basic tracking system by using the velocity of moving objects as an important factor in the tracking algorithm. This is documented in section 4.2.

## 1.4 Contributions of this work

In our videos, there is significant change in object distance from the camera within the scene, which we will simply refer to as "object depth". Change in object depth affects the tracking system thresholds that are used to reflect physical distances between objects in the 3-D world. We present a simple method to make up for the effects of depth variation in scenes. The method uses feedback from the tracking stage and automatically estimates relative object distances in the real 3-D world. Observing how much an object track moves in different parts of the image can be the basis for a reliable method to estimate object depth and real world distances for tracking applications. This thesis develops a concept that we call the 'Vicinity Factor' to estimate object depth and size in different parts of the scene. We introduce the Vicinity Factor concept in chapter 6 and use feedback from the tracking stage to successfully learn the Vicinity Factor for different parts of the image. The efficacy and robustness of the Vicinity Factor makes it, in our view, a significant contribution to existing research in tracking algorithms.

Since there is a great deal of uncertainty reflected in the real-life data, a

probabilistic tracking algorithm is desirable for future applications. Thus, a new Bayesian framework to probabilistically assign track numbers to moving objects is implemented in the system. The Bayesian approach, discussed in chapter 8, is robust to occlusion, disappearance of objects, sudden changes in object velocity, and changes in object color profile and size. It can provide a strong basis for future research.

The Vicinity Factor and the Bayesian approach are the unique contributions of this thesis work. A description of our Bayesian algorithm and results from its use [7] were submitted to and accepted by the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007) Workshop on Object Tracking and Classification in and Beyond the Visual Spectrum (OTCBVS).

## 1.5 Organization of the thesis

Chapter 2 offers a look at current state of research in the field of automatic object tracking in videos. The chapter discusses object segmentation in image sequences, background modeling, and tracking approaches.

The architecture, block diagram, and basic results from our system are explained in chapter 3. In chapters 4 and 5, some shortcomings of the basic system, methods to overcome these shortcomings, and results from the improved algorithm are described. Chapter 6 discusses the problem of depth variation in the scene and the development of the Vicinity Factor to overcome the problem. Improved tracking resulting from use of the Vicinity Factor is shown in chapter 7. The Bayesian framework for tracking and results from its use are presented in chapters 8 and 9.

# Chapter 2

# Background

The common first step in the process of object tracking is to identify objects of interest in the video sequence and to cluster pixels of these objects. Hu et al. [8] offer a very useful survey of the entire process of automated video surveillance and the techniques involved. While color- and edge-based methods can be used to cluster regions on an image by image basis, surveillance video offers an additional temporal dimension that can be exploited to segment regions of interest in the sequence. Since moving objects are typically the primary source of information in surveillance video, most methods focus on the detection of such objects. Common methods for motion segmentation include average background modeling [9], Gaussian background modeling [10], [11], foreground pixel modeling [12], optical flow methods [13–15], gradient methods like the moving edge detector [16], and tensor-based motion segmentation [17].

Simple background subtraction is a widely used method for the detection of moving objects. If the background image is known beforehand, the task of detecting moving objects is trivial and can be done by subtracting the current frame from the background image. However, the background image is not always known

and often needs to be automatically generated by the system [9]. The background image can also change significantly with changes in lighting. In outdoor scenes, the background itself may be dynamic and may change rapidly (for instance, trees swaying in the breeze). Several methods have been used for the development of background and/or foreground models [10–12]. Stauffer and Grimson [11] use a Gaussian model of the background process for each pixel. Classification of pixels as foreground or background is carried out by calculating the probability that the current intensity level of the pixel can be explained by the background process model.

There are other approaches that do not rely on the estimation of background or foreground models. Biancardini et al. [16] discuss a method of segmenting moving objects by using moving edges and a hierarchical color segmentation method. Yet another paradigm in motion segmentation is to use computationally intensive optical flow calculations [15, 17]. In our method, we use an average background model along with a concept we call Secondary Background. This results in a very good segmentation without requiring the expensive computations of the above methods.

Once moving regions have been identified, the next task is to generate tracks of these objects over successive frames. This is essentially a correspondence task, the goal of which is to find a match for each blob or track in the previous frame to one of the blobs or objects in the current frame. A match matrix, usually based on Euclidean and Manhattan distances, is commonly used as a measure of the match between blobs of the previous frame and blobs of the current frame. For instance, in [18], the weighted sum of distances in color values (R,G, and B), position values (Y(column) and X(row) coordinate values), and velocity values between tracks of

previous frames and blobs of current frames is used as the measure for matching. We use a similar method, evaluating the measure of the match between blobs by calculating the Euclidean distance in color and position values between them. Kalman filters may be used to estimate the track position and velocity in the new frame [10, 19–21]. Variations of the Kalman filter, such as the modified Extended Kalman Filter [22], are in use and report improved performance in complex traffic environments. We use a linear prediction for track position based on its position in previous frame and its velocity.

An alternative to using motion segmentation involves generating a reference model of the object being tracked, then looking for the best match for the model in each frame by using predictive approaches like Kalman filters [20], [23], and sampling-based approaches like Condensation [24]. These methods have the disadvantages of having to develop a model of the object being tracked, and of complicated filter initialization and update procedures for tracking. Our system uses motion as the basis for object segmentation and tracking.

Yilmaz et al. [25] is a good survey of various tracking approaches being used currently. For tracking multiple objects, the Joint Probabilistic Data Association Filter(JPDAF) [23] [26] is the most widely used method. The JPDAF has the advantage of being able to probabilistically assign objects to tracks. Probabilistic track assignment can be very useful and in chapter 8, we develop a new approach for probabilistic track assignment based on the Bayesian principle, along with relevant background information.

In many surveillance videos, there is significant variation in object depth within the scene. Some objects may be very close to the camera, while others may be far away. Simple difference between pixel positions in Y(column) and X(row) values

is not a good indicator of how close two objects are in the real 3-D world. This variation in depth can affect the tracking algorithm, which is based on the difference between objects in pixel position values. Automatic learning of 3-D distance variation can be very useful in tracking applications. 3-D modeling of the scene, thus, becomes an important factor in automatic tracking algorithms. 3-D modeling is a difficult problem, and stereo image based solutions are most commonly used for depth determination like in [27] and [28]. Since we use monocular images, stereo methods cannot be used. Camera calibration is an important problem to be considered in 3-D modeling methods. 3-D modeling of traffic scenes by use of a fully calibrated camera and some prior knowledge about background and foreground regions in the scene is discussed in [29]. We do not use a calibrated camera approach for estimating object depth. A method for estimating 3-D structure from motion observed in monocular image sequences based on tracking some feature points, and models for translation and rotation are discussed in [30]. These methods are too complicated for the purpose of relative 3-D distance estimation in traffic videos from static cameras. In chapter 6, we discuss an alternative we have developed for distance estimation in the scene based on feedback from the tracking stage of the surveillance system.

Some of the existing automatic tracking systems discussed often in literature are VSAM [31], W4 [32], and Bramble [33]. VSAM [31] uses a statistical model for the background based on the average and deviation in intensity values at each pixel. A connected components approach is used to segment targets. The objects are tracked based on frame-to-frame matching of blobs using their position, velocity, appearance, size, and color histogram characteristics. Our system is most similar to VSAM. Both systems are designed to track subjects in outdoor videos

using a static camera, and they do so using somewhat similar segmentation and tracking methods. However, we also focus on the problem of depth variation of objects as they move through the scene. Our Bayesian approach for probabilistic track assignment is another significant addition.

The W4 [32] system is used to detect and track human targets. The background model in W4 consists of the maximum, minimum, and maximum difference observed in pixel intensity values for each pixel between consecutive frames during a training period. Thresholding, noise cleaning, and morphological filtering are used to detect moving pixels. Connected component analysis is applied to detect and label the moving objects. Medians (rather than centroids) of the tracked objects regions are predicted and tracked in consecutive frames. Further, binary edge correlation between current and previous silhouette edge profiles is computed over a $5 \times 3$ displacement window to determine position of the object more accurately. A model is used to determine position and motion of head, torso, hands, and feet. Our system currently does not use models for humans or vehicles. The use of models in the W4 system makes it possible to use some higher-level intelligence for tracking and analysis purposes, but comes with the disadvantage of model generation and initialization issues. Models can be useful when the application domain and target environment have been narrowed. The W4 system is designed to handle occlusion, though the paper [32] does not show specific results of objects being lost and recovered. Our system is designed to handle occlusion and results are presented to show how the feature works.

Bramble [33] detects people in indoor environments and uses explicit 3-D cylindrical modeling of target objects. The popular CONDENSATION algorithm [24] is used for prediction and tracking in Bramble. A Bayesian filter for tracking

multiple objects when the number of objects is unknown and variable over time is implemented in Bramble. A Gaussian model is used for background modeling, where the Gaussian model for each pixel is developed offline by observing a few hundred empty background images. Our system, like the VSAM and W4 systems, uses an online algorithm for background model generation, and works on objects in outdoor scenes, where background modeling, occlusion, non-static background regions (like swaying trees and leaves), and camera jitter are difficult challenges that have to be addressed.

# Chapter 3

# System architecture and description

This chapter discusses each of the steps involved in our solution to automatic moving target detection and tracking in detail. Fig. 3.1 shows the block diagram of the system and its constituent modules again.

An image sequence captured by a still camera is the input to the system. We first perform Motion Segmentation to extract moving blobs in the current frame. Some blobs that are very small and are likely to be noise are deleted. Due to partial occlusion, some moving objects get incorrectly segmented into two or more separate blobs. Using color and position information, such blobs are merged. The Object Tracking module tracks these moving objects over successive frames to generate Object Tracks.

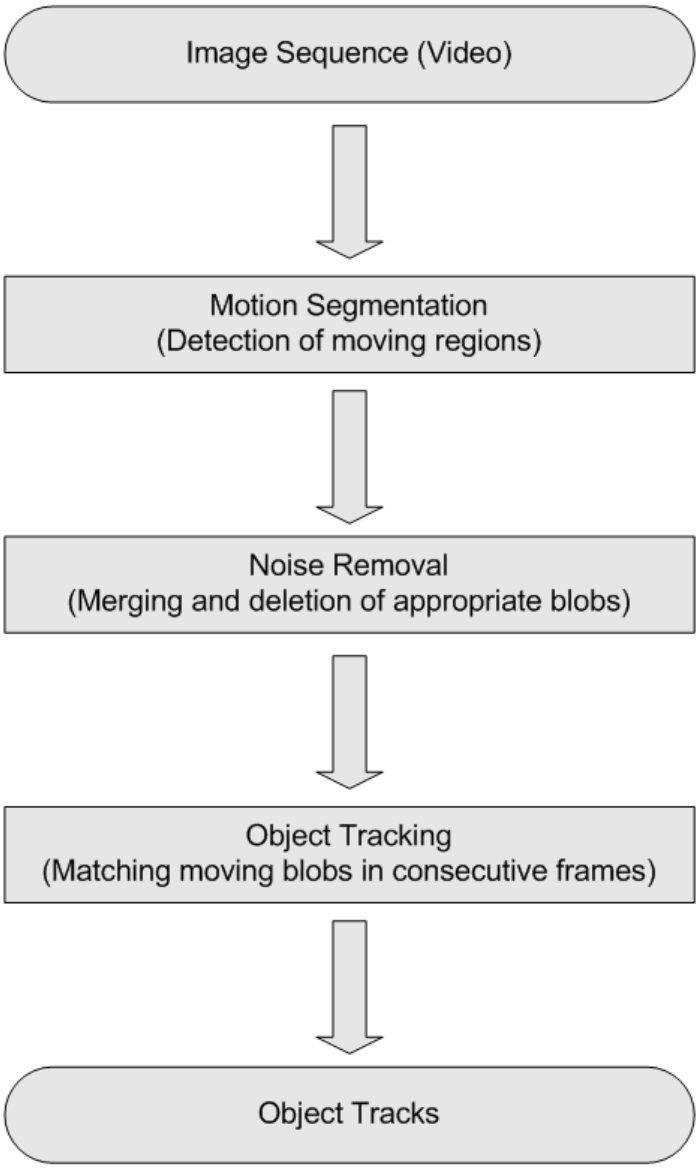



**Figure 3.1.** Block diagram of the system

## 3.1 Motion Segmentation

The first task in any surveillance application is to distinguish the target objects in the video frame. Most pixels in the frame belong to the background and static regions, and suitable algorithms are needed to detect individual targets in the scene. Since motion is the key indicator of target presence in surveillance videos, motion-based segmentation schemes are widely used. We utilize an effective yet simple method for approximating the background that enables the detection of individual moving objects in video frames.

### 3.1.1 Segmentation using moving average background model

A standard approach for finding moving objects in still-camera video data is to create a model for the background and compare that model with the frame in question to decide which pixels are likely to be the foreground (moving objects) and which the background. There are various methods for developing background models. Our approach is to use a simple average model where the average intensity values for each pixel over a window of N (typically 150) frames is regarded as the background model. The entire background modeling and segmentation process is carried out on grayscale versions of the images. Fast-moving objects do not contribute much to the average intensity value [9]; thus, the model becomes a reasonable estimate of the background.

If $I^k(y,x)$ is the intensity level at coordinates $Y = y$ and $X = x$ of the $k^{th}$ frame in the sequence and $bg^k(y,x)$ is the average background model value at $(y,x)$, then

$$bg^k(y,x) = \frac{1}{N} \sum_{j=k-(N/2)}^{k+(N/2)} I^j(y,x) \tag{3.1}$$

The segmented output is:

$$seg1^k(y,x) = \begin{cases} 1 & \text{, if } |bg^k(y,x) - I^k(y,x)| > T1 \\ 0 & \text{, otherwise} \end{cases} \tag{3.2}$$

where $T1$ is a predetermined threshold, usually 10.

Fig. 3.2 shows the grayscale version of a frame that is being processed. Fig. 3.3 is an example of how the moving average background model appears in frame 1010 of the video sequence. After subtracting this background model from the current frame and thresholding the difference image, Fig. 3.4 results. The white pixels in the image are the regions that have been segmented as moving pixels in the current frame.


**Figure 3.2.** Grayscale version of current frame being processed for background segmentation


**Figure 3.3.** Moving average background model for current frame 1010


**Figure 3.4.** Moving pixels segmented after subtraction from average background model and thresholding the difference

This base method does have a major drawback in that the segmented object regions have "tails" due to the effect of the window that is used for the average background model, as illustrated in Fig. 3.5(a), where an envelop has been established around the moving regions in the original color image. The difference image shows a high value in the region where the object earlier/later existed and where the object now exists, resulting in tail regions behind and before the object. The correct segmentation for the frame, achieved by improvements in the basic algorithm as discussed in section 3.1.2 is shown in Fig. 3.5(b).


**Figure 3.5.** (a)Tails due to basic segmentation (b) Corrected results after use of Secondary background(section 3.1.2)

### 3.1.2 Enhanced segmentation using a Secondary Background model

It was observed that the intensity level of a background pixel changes very little from frame to frame. The value of the background in the previous frame is actually a better estimate of the background in the current frame than is the average value over N frames. To take advantage of this, we use a secondary model of the background (called Secondary Background) which is the actual intensity

level of background pixels in the previous frame (or in the frame where the pixel was last classified as background). By combining the two models, a near perfect segmentation of the moving objects is obtained. The average model is used to identify moving regions and the Secondary Background is used to identify the valid moving object within these regions. The concept is illustrated in Fig. 3.6. The oval region around the car is the moving region. In frame 3, we can differentiate between the car region (D) and background regions (C and E) by using the Secondary Background for regions D (from frame 2), C (from frame 1), and E (from frame 2).

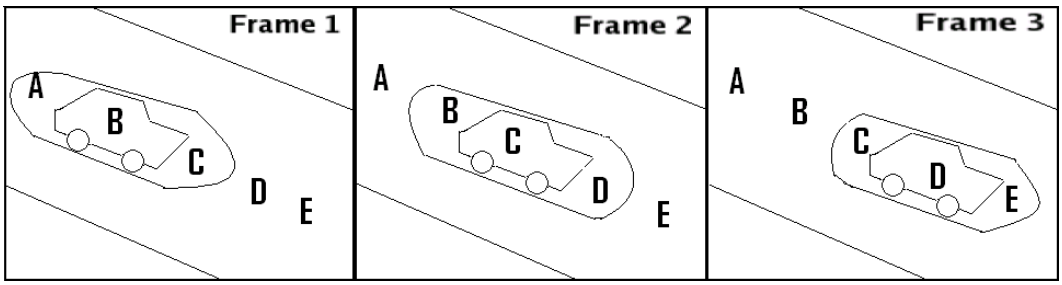



**Figure 3.6.** Illustration of the Secondary Background concept

After the segmentation, the background model is updated by calculating the average over N frames. The Secondary Background is updated by storing the intensity levels of pixels that were classified as background as the new Secondary Background. For pixels that were classified as foreground, the Secondary Background is left unaltered so that it holds the value of intensity from the frame where the pixel was last classified as background.

Secondary Background for frame $k$ is denoted by $sbg^k$. For initialization, we simply set $sbg$ to the background calculated for the first frame:

$$sbg^1 = bg^1(y, x) \tag{3.3}$$

where $sbg^1$ is the initial value for Secondary Background.

The segmented output based on the Secondary Background is denoted by:

$$seg2^k(y,x) = \begin{cases} 1 & \text{, if } (|sbg^k(y,x) - I^k(y,x)| > T2 \text{ and } seg1^k(y,x) = 1) \\ 0 & \text{, otherwise} \end{cases} \tag{3.4}$$

$T2$ is a threshold, usually greater than $T1$. In our application, $T2$ was set at 15.

We then update the Secondary Background for the next frame:

$$sbg^{k+1}(y,x) = \begin{cases} I^k & \text{, if } seg2^k(y,x) = 0 \\ sbg^k & \text{, otherwise} \end{cases} \tag{3.5}$$

As you can see, the segmentation and background modeling processes are interdependent, with the segmentation result feeding back to the Secondary Background model.

The Secondary Background model for the example frame 1010 are shown in Fig. 3.7. Subtraction from current frame and thresholding, as described in equation 3.4, results in the segmentation of the moving pixels as Fig.3.8. This improved segmentation algorithm gives the correct segmentation that was presented in Fig. 3.5(b). On comparing figures 3.3 and 3.7, we can see that the section of road in front of where the cars are found in the current frame is slightly darker in the basic method (Fig. 3.3). The improved background (Fig. 3.7) that uses the Secondary Background model does not exhibit this error.

Our background modeling method is much simpler than other adaptive background methods such as Stauffer and Grimson [11]. We found that while these complicated methods promise good results for complex and bimodal backgrounds,

**Figure 3.7.** Secondary Background model for current frame 1010

**Figure 3.8.** Moving pixels segmented after subtraction from Secondary Background model and thresholding the difference

they have very complicated procedures and are computationally expensive. In essence, our method is similar to Stauffer and Grimson's Gaussian background pixel model, but with a very high learning rate for the model. Our simple implementation was able to produce good results with little computational complexity. A sample of the segmentation results obtained using the Secondary Background model is shown in Fig. 3.5(b).

## 3.2 Noise removal

Following a segmentation of the image frame into moving and non-moving pixels, region growing is used to locate and identify each moving object. Each group of pixels that are connected to each other is labeled as one blob. An envelope is established around each object of interest by use of morphological erosion and dilation filters. The erosion-dilation operation also removes blobs that are very small and thus likely to be noise. Additional noise is filtered by ignoring blobs less than S pixels in size.

The envelop of the grayscale image is copied in the same locations on the color version of the current image. The color version of the frame is then analyzed to get the statistics for each blob. By averaging the location and color values for all pixels belonging to a blob, the centroid value of the blob(in Y and X coordinates), and mean R,G, and B values are obtained. These values are used as statistics for matching blobs across frames.

## 3.3 Representation of a blob

The variables and their description in the data structure used for representing an individual blob are given in Fig. 3.9. *BlobID* is the number that is given to a blob. The track number of the blob, as assigned by the tracking algorithm, is stored in this variable. The *meanR*, *meanG*, and *meanB* variables store the average intensity values of red, green, and blue colors of all pixels belonging to the blob. The centroid position of all pixel locations is stored in *yCenter* and *Xcenter*. The *size* variable keeps record of the size of the blob in pixels (number of pixels belonging to the blob).

The Blob class used for keeping record of all the individual blobs in the video is given in Fig. 3.10. An array of individual blobs is maintained in the *blobs* variable. *numBlobs* is an array that stores the number of blobs in each frame of the video. The *getBlobStats* function analyzes the current frame to generate all the statistics for all the blobs in the frame. *meanR*, *meanG*, *meanB*, *yCenter*, *Xcenter*, and *size* are the statistics that are calculated and stored for each blob in the frame. *connectBlobPixels* does a connected pixel based region growing to group together individual pixels into regions (blobs). *cleanBlobs* function removes any blobs that are smaller than a given threshold size. Thus, very small blobs that may be noise are deleted. *mergeBlobs* is a function that merges blobs that are very close to each other in position and color values. The need for *mergeBlobs* function and some issues involved in its implementation are discussed in chapter 4. *drawEnvelope* and *labelBlobs* functions are used to mark out the blobs in the original image. *drawEnvelope* function uses morphological erosion and dilation operations to draw an envelop around the detected blob in the image. *labelBlobs* is a function that is designed to write the *blobID* number near the centroid of all detected blobs in the image.

**Individual Blob structure- *blob***

| *Variable* | Description |
|---|---|
| *BlobID* | ID number of blob - usually the number of the track to which the blob belongs |
| *meanR* | Average Red value of all pixels in blob |
| *meanG* | Average Green value of all pixels in blob |
| *meanB* | Average Blue value of all pixels in blob |
| *yCenter* | Position of centroid of blob along Y axis (column) |
| *xCenter* | Position of centroid of blob along X axis (row) |
| *size* | Size of blob in pixels |

**Figure 3.9.** Variables in the individual *blob* data structure

**Blob class**

| | Description |
|---|---|
| ***Variables:*** | |
| *blobs* | Array of all individual *blob*s in the video |
| *numBlobs* | Array that stores the number of *blob*s in each frame of the video |
| **Methods:** | |
| getBlobStats | Analyzes current frame to generate *meanR*, *meanG*, *meanB*, *yCenter*, *xCenter*, and *size* of each *blob* in current frame. Updates *blobs* and *numBlobs* variables. |
| connectBlobPixels | Does connected pixel analysis to create blobs from pixels |
| cleanBlobs | Deletes *blob*s smaller in size than a pre-decided threshold |
| mergeBlobs | Merges blobs that are very similar in *Ycenter*, *Xcenter*, *meanR*, *meanG*, and *meanB* values |
| drawEnvelope | Draws an envelop around each blob in the image by using erosion and dilation morphological operations |
| labelBlobs | Writes the *blobID* number close to the centroid of the blob in the image |

**Figure 3.10.** Variables and methods in the Blob class

## 3.4 Representation of a track

The data structure used to represent an individual track is shown in Fig. 3.11. *trackID* stores the unique track number for the track. *startingFrame* and *endingFrame* variables store the first and last frame in which the track was seen, respectively. *lostCount* variable keeps count of how many consecutive frames for which a track has been 'lost'. This is kept so that a track that is lost for more than 5 frames can be deleted and closed. $y$ and $x$ keep record of the current position of the track in Y and X directions, respectively. *velocityY* and *velocityX* is the velocity of the track calculated over the last 5 frames in Y and X directions, respectively. *predictedY* and *predictedX* are the predicted positions for the track in the next frame, calculated based on the current velocity and position of the track.

The Track class variables and functions are presented in Fig. 3.12. *tracks* is an array of all individual tracks in the video. *numTracks* is an array that stores the number of tracks in each frame of the video. *openTracks* is an array that keeps a list of all tracks that were active in the previous frame (including tracks that were 'lost', but not deleted). *trackCount* is the count of total number of tracks in the video. The functions *createTrack* and *deleteTrack* create a new track and delete a track, respectively. *lostTrack* function is called when a track is not found in the current frame. The function increments the *lostCount* variable for the track. If the *lostCount* variable is greater than 5 (the track has been lost in more than 5 consecutive frames), then the *lostTrack* function calls the *deleteTrack* function and deletes the track. *updateTrack* function updates the position, velocity, and prediction variables of the track. *EuclideanTracker* is the tracking algorithm that finds correspondence between tracks of previous frame and blobs of current frame

**Individual Track structure- track**

| *Variable* | Description |
|---|---|
| *trackID* | ID number of track |
| *startingFrame* | The frame number when the track first appeared |
| *endingFrame* | The frame number when the track last appeared |
| *lostCount* | The number of frames for which the track has been "lost", this is 0 if the track was found in the previous frame |
| *y* | Location of track along Y direction |
| *x* | Location of track along X direction |
| *velocityY* | Velocity of track along Y direction |
| *velocityX* | Velocity of track along X direction |
| *predictedY* | The track's predicted position along Y direction in the next frame |
| *predictedX* | The track's predicted position along X direction in the next frame |

**Figure 3.11.** Variables in the individual *track* data structure

**Track class**

| | Description |
|---|---|
| ***Variables:*** | |
| *tracks* | Array of all individual *track*s in the video |
| *numTracks* | Array that stores the number of *track*s in each frame of video |
| *openTracks* | Array that stores a list of active tracks from the previous frame |
| *trackCount* | Total number of tracks in the video |
| **Methods:** | |
| createTrack | Creates new track |
| deleteTrack | Deletes a track |
| lostTrack | Declares a track as lost, increments *lostCount* for lost track<br>If *lostCount* is greater than 5, calls deleteTrack function |
| updateTrack | Updates position, velocity, and prediction values for a successfully matched track |
| EuclideanTracker | Algorithm to find track-to-blob correspondence in each frame based on Euclidean distance |

**Figure 3.12.** Variables and methods in the Track class

based on the Euclidean distance between blobs and tracks in order to achieve continuous tracking over consecutive frames.

## 3.5 Object Tracking

In our implementation, we perform object tracking by finding correspondences between tracks in the previous frame and blobs in the current frame.

### 3.5.1 Correspondence algorithm

After the moving objects have been segmented as blobs, the next task is to find correspondence between tracks of previous frame and blobs of current frame. Most commonly, a match matrix is calculated for the tracks and the blobs, and the best matches are calculated from the match matrix. Each row in the match matrix corresponds to a track from the previous frame and each column corresponds to a blob from the current frame. Each element in the matrix is, thus, the degree of match between a track and a blob. The values entered in the matrix are usually the distance (Euclidean or Manhattan) between a track and a blob. For instance, in [18] the match matrix consists of the weighted sum of Euclidean distance in position in Y and X coordinates, Manhattan distance in color histogram values, and distance in velocity values between a track and a blob. The smaller the distance, the better the match. In our system, each element in the match matrix is the sum of the Euclidean distance in position values (Y and X coordinate values) and the Euclidean distance in color values (R, G, and B values). The values for Y and X values are normalized against the maximum Y and X dimensions of the images, respectively. Similarly, R, G, and B values are normalized against their maximum possible value, which is 255.

Say $T^{k-1} = \{t_1^{k-1}, t_2^{k-1}, t_3^{k-1}, \ldots, t_{u^{k-1}}^{k-1}\}$ is the set of tracks from the previous frame $(k-1)$, where $t_j^{k-1}$ is the $j^{th}$ track and $u^{k-1}$ is the total number of tracks in frame $(k-1)$, and

$O^k = \{o_1^k, o_2^k, o_3^k, \ldots, o_{v^k}^k\}$ is the set of moving blobs in frame $k$, where $o_i^k$ is the $i^{th}$ blob in frame $k$ and $v^k$ is the total number of moving blobs in frame $k$.

The $j^{th}$ row and $i^{th}$ column in the match matrix, denoted by $MM_{j,i}$ is given by:

$$MM_{j,i} = \sqrt{(\Delta Y/Ydim)^2 + (\Delta X/Xdim)^2} + \sqrt{(\Delta R/255)^2 + (\Delta G/255)^2 + (\Delta B/255)^2} \tag{3.6}$$

where

$\Delta Y$ is the difference in Y position values,

$\Delta X$ is the difference in X position values,

$\Delta R$ is the difference in mean red values,

$\Delta G$ is the difference in mean green values, and

$\Delta B$ is the difference in mean blue values between track $j$ of the previous frame and blob $i$ of the current frame.

$Ydim$ and $Xdim$ are the image dimension values in Y and X coordinates, respectively (240 and 320, in our images).

The Y and X position values of a blob correspond to the location of the centroid of all pixels that have been segmented as belonging to the concerned blob. The mean red, green, and blue values for each blob are calculated by averaging the R, G, and B values for all pixels that have been segmented as belonging to the concerned blob.

### 3.5.2 Match matrix

Fig. 3.13 shows a sample match matrix for frame 0300 of video sequence 1. For simplicity, only tracks that are present in frame 0295 have been displayed.

(a) Previous frame - 0295 (b) Current frame - 0300

| | Blob 1 | Blob 2 | Blob 3 | Blob 4 |
|---|---|---|---|---|
| **Track 3** | 1.18 | 0.02 | 0.13 | 1.10 |
| **Track 13** | 1.21 | 0.12 | 0.03 | 1.10 |
| **Track 16** | 0.36 | 1.03 | 1.00 | 0.21 |
| **Track 19** | 0.11 | 1.28 | 1.28 | 0.33 |

(c) Match matrix for current frame 0300

**Figure 3.13.** Match matrix for example frame 0300

The matrix shows that the best matches for tracks 03, 13, 16, and 19 are blobs 2, 3, 4, and 1, respectively.

### 3.5.3 Factors of uncertainty in tracking

In the case of a noiseless environment and perfect segmentation of blobs, the task of tracking is straightforward. In reality, however, there is often a great deal of uncertainty in the final segmentation, as well as difficulties produced from imperfect knowledge of the scene context. Some common reasons for uncertainty are:

- Valid blobs may fail to be detected due to their small size
- Valid blobs may be occluded by another moving or static object

- Some moving objects such as tree branches, though valid moving blobs, are not of interest for our application
- Some blobs might be incorrectly merged by the blob merging module
- Objects entering (or leaving) the scene do not have corresponding blobs in preceding (or succeeding) frames
- An object may change in orientation or be temporarily occluded, leading to a change in its visible surfaces and thus its color statistics or even its size

All of the above, in addition to jitter and noise from the camera, make the correspondence a difficult problem to solve, particularly for videos that are shot outdoors. Our goal is to develop a tracking scheme that is robust to these sources of uncertainty.

### 3.5.4 Track assignment procedure

For each motion blob in the first frame, a new track is created. In subsequent frames, based on the match matrix values, matching blobs are sought in the current frame for each track in the previous frame. Matched blobs are assigned track numbers from previous frame. The track statistics are updated with position and color information from the matching blob. We allow tracks to be declared 'lost' if any of the factors mentioned above causes the algorithm to fail to detect a suitable match for a track. The track is not deleted but is 'kept alive' for a few frames in hopes that a match will be found. A track is deleted only if a suitable match is not found for L (typically 5) consecutive frames. If there is any blob in the new frame that cannot be matched to an existing track, then a new track is generated. A list of currently 'active' tracks is maintained for each time

step/frame. If there is a tie between multiple tracks for a single blob, then the older of the competing tracks is assigned to the blob. This is because a longer running track is less likely to be noise.

## 3.6 Results

Some results of the base system are shown below. In the figures where match matrices are presented, the assigned blob-track pairs are shaded differently in the matrix for easy reading.

Fig. 3.14 shows the match matrix for a case when a new track has to be initialized to account for a new unmatched blob that occurs in the current frame. Current frame 0290 has two new blobs compared to the previous frame 0285. The match matrix, when solved as explained earlier, results in assigning blobs numbered 2, 3, and 4 to tracks 03, 13, and 16, respectively. Since blobs numbered 1 and 5 are not matched to any active previous tracks, new tracks 19 and 20 are started to account for these, respectively.

Fig. 3.15 shows the match matrix when a previously active track gets lost. In this case, track 20, which was active in frame 0290 is now lost due to occlusion behind the tree in the foreground. The match matrix shows that two tracks (16 and 20) are good matches for the blob numbered 4. The algorithm is designed to choose the older of the two tracks. Since longer running track numbers are more reliable and less likely to be noise, we always choose the older numbered track in case of a tie between track numbers to be assigned to a blob. Hence, track 16 is assigned to the blob and track 20 is declared as 'lost'. The reason for choosing an older track over a newer one is clear in this example. As we see in the previous example (Fig. 3.14(b)), track 20 was created due to the erroneous breaking up of


(a) Previous frame - 0285 (b) Current frame - 0290

| | Blob 1 | Blob 2 | Blob 3 | Blob 4 | Blob 5 |
|---|---|---|---|---|---|
| Track 3 | 1.30 | 0.04 | 0.12 | 1.22 | 1.31 |
| Track 13 | 1.28 | 0.12 | 0.03 | 1.18 | 1.26 |
| Track 16 | 0.32 | 1.15 | 1.17 | 0.05 | 0.22 |

(c) Match matrix for current frame 0290

**Figure 3.14.** Match matrix when new track is initialized

track 16 because of occlusion. Choosing the older of the two tracks (16 and 20) ensures that the erroneous track is now deleted.

Fig. 3.16 shows a sequence of frames where the algorithm is able to allow for tracks to be 'lost' when objects disappear due to occlusion and then recovered when they resurface. Tracks 03 and 08 are being tracked in these frames. Frames 0165 and 0170 (Figs. 3.16(a) and (b)) show both tracks. In frame 0175 (Fig. 3.16(c)), track 08 is 'lost' due to the car being occluded by the tree. In frame 0180 (Fig. 3.16(d)), both tracks are 'lost' due to occlusion. In frame 0185 (Fig. 3.16(e)), track 08 is reassigned to the correct blob when the object re-emerges from the occluded region. Finally, in frame 0190 (Fig. 3.16(f)), when both objects have resurfaced, the algorithm recovers the correct track numbers.

(a) Previous frame - 0290 (b) Current frame - 0295

| | Blob 1 | Blob 2 | Blob 3 | Blob 4 |
|---|---|---|---|---|
| **Track 3** | 1.28 | **0.01** | 0.13 | 1.05 |
| **Track 13** | 1.31 | 0.13 | **0.01** | 1.06 |
| **Track 16** | 0.31 | 1.16 | 1.16 | **0.11** |
| **Track 19** | **0.04** | 1.27 | 1.30 | 0.40 |
| **Track 20** | 0.32 | 1.29 | 1.29 | 0.23 |

(c) Match matrix for current frame 0295

**Figure 3.15.** Match matrix when track is lost

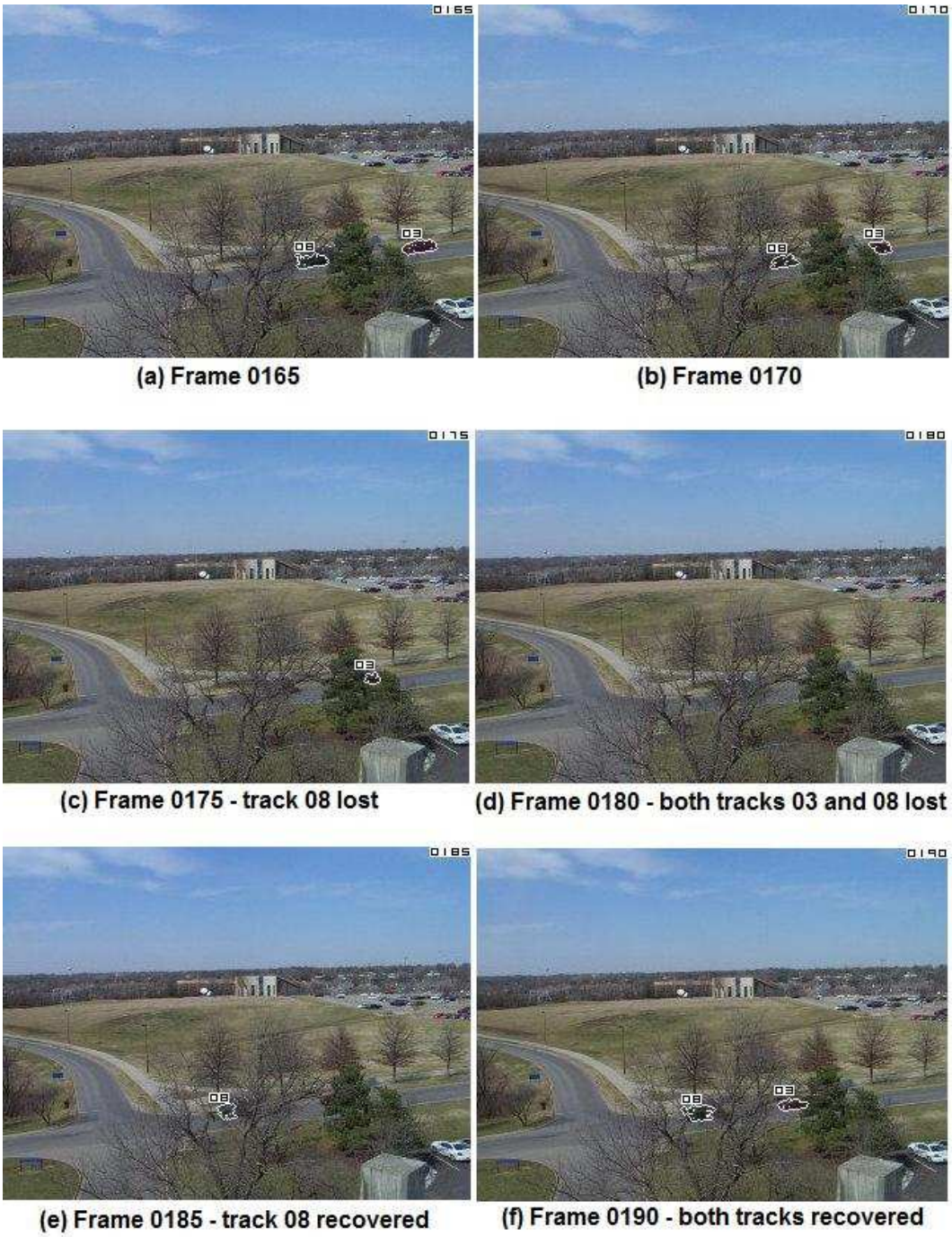



**Figure 3.16.** Tracks from subsequence in video 1

# Chapter 4

# Shortcomings in the basic algorithm

The basic system proved to be a good starting point from which to tackle some of the practical challenges in tracking objects in video. The results from the basic system highlight some of the problems associated with object segmentation and tracking. As figures 4.1 and 4.2 show, in some cases, a particular object can be broken up into two or more blobs because of discontinuities in the object, typically due to occlusion by some smaller object.

Another drawback is that the basic system may find a match for an object completely against the previous direction of velocity of the track (examples are discussed later in section 4.2). Thus, the velocity vector of the track should be an important factor in tracking objects.


**Figure 4.1.** Object broken in frame 0100


**Figure 4.2.** Object broken in frame 0195

## 4.1 Solution for broken objects: Merge module

After background segmentation, individual pixels that have been detected as moving pixels have to be clustered into objects. Each cluster of connected pixels is grouped as one moving object. This approach, however, often results in a single object in the image being broken down into two or more smaller objects. Often, this occurs due to occlusion by other objects in the scene. In the video sequence 1

that we have used so far in the examples, the section of the road that is occluded due to trees in the foreground often shows a single car being broken into smaller objects. This can add serious complications in the correspondence and tracking stage of the system. A module that merges blobs which are very close to each other in location and in color space is very effective at handling such cases. For all blobs in the image, pairwise comparison of their location (Y and X values) and color values (R, G, and B values) is done. If the difference in each of these values is less than predetermined thresholds, then the two blobs are merged and given the same blob number

The merge algorithm is:

For any two blobs $a$ and $b$ in current frame,

$$\begin{aligned}
&\text{If } \{ \quad \Delta Y \leqq Ythreshold \\
&\quad \text{and } \Delta X \leqq Xthreshold \\
&\quad \text{and } \Delta R \leqq Rthreshold \\
&\quad \text{and } \Delta G \leqq Gthreshold \\
&\quad \text{and } \Delta B \leqq Bthreshold \quad \} \\
&\text{then, relabel all pixels of blob } a \text{ with number } b
\end{aligned} \tag{4.1}$$

where $\Delta Y$, $\Delta X$, $\Delta R$, $\Delta G$, and $\Delta B$ are the difference in mean Y, X, R, G, and B values, respectively, between blobs $a$ and $b$. $Ythreshold$, $Xthreshold$, $Rthreshold$, $Gthreshold$, and $Bthreshold$ are predetermined thresholds.

Fig. 4.3(b) shows segmented frame 0195 as a result of using the blob merge algorithm. Fig. 4.3(a) shows the same frame when blob merging is not used. Occlusion from the tree causes object 7 of Fig. 4.3(b) to be mistaken as two separate objects 12 and 08 in Fig. 4.3(a). Thus, the blob merging module is able to merge blobs that are likely to belong to the same object. The effectiveness of the merging, however, depends strongly on the thresholds used in the algorithm. In this example, a threshold value of 20 pixels was used for $Ythreshold$ and $Xthreshold$ to look for suitable blobs to be merged.


**Figure 4.3.** Merge module succeeds in frame 0195

### 4.1.1 Dependence of merge module on position thresholds : Inadequacy of a single threshold for entire image

Fig. 4.4 shows how the same threshold of 20 for a different frame 0255 results in two separate cars being merged as single object 03 because of their proximity.

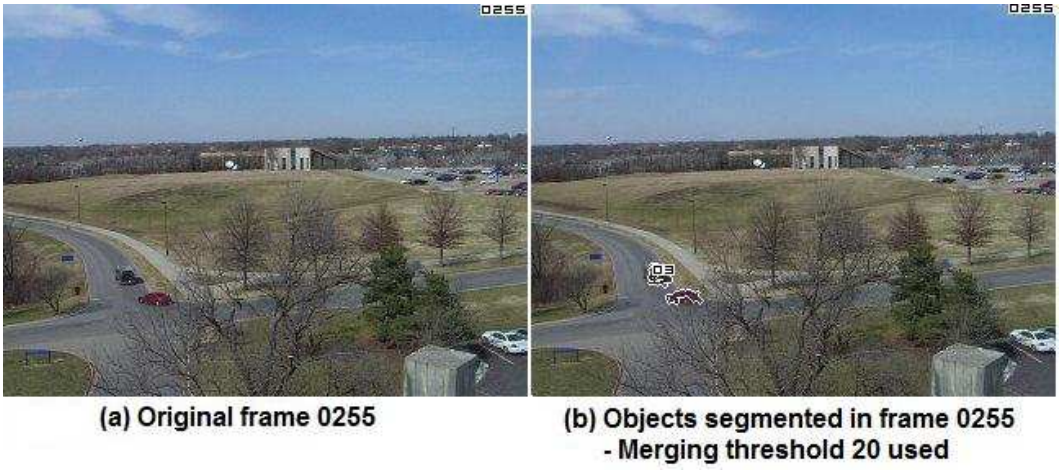



**Figure 4.4.** Merge module fails in frame 0195

Figures 4.5 to 4.9 further explain the dependence of the merging algorithm on the threshold used for blob merging. Fig. 4.5 shows frames 0255, 1000, and 1160 as they appear in the original unsegmented video sequence 1. The object segmentation results without use of blob merging can be seen in Fig. 4.6. While there is no need for blob merging in frames 0255 (Fig. 4.5(a)) and 1000 (Fig. 4.5(b)), frame 1160 (Fig. 4.5(c)) clearly calls for use of the blob merging module.

Results from the use of a position threshold value of 20 for $Ythreshold$ and $Xthreshold$ are shown in Fig. 4.7. Using a threshold of 20 implies that blobs which are within a range of 20 pixels in both Y and X directions of each other are considered as candidates for merging. If such candidate blobs are within the color thresholds as well, then they are merged and given a common blob number. As can be seen in Fig. 4.7, threshold value of 20 handles the blobs in frames 1160 (Fig. 4.7(c)) well. But the threshold proves to be very large in certain regions of the scene as can be seen in frames 0255 and 1000 (Figs. 4.7(a) and 4.7(b)). In both these frames, blobs which ought to be considered as separate cars are grouped together because they are within 20 pixels of each other and also very


(a) Original frame 0255


(b) Original frame 1000


(c) Original frame 1160

**Figure 4.5.** Frames used to explain merge module threshold problem


(a) No blob merging needed in frame 0255


(b) No blob merging needed in frame 1000


(c) Blob merging is needed in frame 1160

**Figure 4.6.** Results for frames without use of merge module

similar in their color statistics (R,G, and B values). Use of a smaller threshold 10 as shown in Fig. 4.8 fails to merge blobs 27 and 38 of frame 1160 (Fig. 4.8(c)). Use of an intermediate threshold of 15 (Fig. 4.9) exhibits problems similar to those that result from use of a threshold of 20. The blobs in frame 0255 (Fig. 4.9(a)) are treated correctly, but the blobs in frame 1000 (Fig. 4.9(b)) are erroneously merged because their positions are within 15 pixels of each other.

Thus, we observe that use of a smaller threshold fails to make some valid merges, while a higher threshold could result in erroneous merging. Fine tuning the thresholds is important to achieve best results. More importantly, even finely tuned position thresholds may not be useful because there is significant change in object depth in the scenes. This is evident in the example images shown previously. A single threshold for the entire image is not adequate for handling different merge cases effectively. There is a need to treat different regions of the scene with different thresholds. For example, in Fig. 4.6, the same position threshold values can not be used for the region around car 28 and region around car 41. The thresholds for proximity in position ($Ythreshold$ and $Xthreshold$) need to be different for different parts of the image because of the difference in object depth. The position thresholds used in the merging algorithm, thus, should be functions of the location (Y and X values) of the object in the image. Arriving at different values for thresholds for different regions in the scene can be a tedious task if done manually. Also, the optimum values for the thresholds will have to be recalculated for every new scene depending on the real-world 3-D setup of each scene. An automatic method to estimate these position thresholds can be very useful and is discussed in chapter 6, where the Vicinity Factor is introduced to tackle the problem of depth variation in scenes.


(a) Merge threshold 20 for frame 0255
- Erroneous merging of cars


(b) Merge threshold 20 for frame 1000
- Erroneous merging of cars


(c) Merge threshold 20 for frame 1160
- No errors in merging of cars

**Figure 4.7.** Merge module results with threshold=20


(a) Merge threshold 10 for frame 0255
- No errors in merging of cars


(b) Merge threshold 10 for frame 1000
- No errors in merging of cars


(c) Merge threshold 10 for frame 1160
- Error: Cars 27 and 38 not merged

**Figure 4.8.** Merge module results with threshold=10


(a) Merge threshold 15 for frame 0255
- No errors in merging of cars


(b) Merge threshold 15 for frame 1000
- Erroneous merging of cars


(c) Merge threshold 15 for frame 1160
- No errors in merging cars

**Figure 4.9.** Merge module results with threshold=15

## 4.2 Solution for velocity flow: Velocity factor

Figures 4.10 and 4.11 show examples where a match for a blob in previous frame is found in the current frame in a location that is not consistent with previous velocity flow of the concerned tracks. The original unsegmented frames of the sequence are also presented in both figures for reference. Fig. 4.10(a) shows two cars, both of which are correctly segmented and tracked in Fig. 4.10(c). In the current frame 0130, track 03 is completely occluded by the tree (Fig. 4.10(b)). The matching algorithm finds that the best match is as given in Fig. 4.10(d). Track 03 gets assigned to the car that should ideally be numbered 07.

In Fig. 4.11, four cars are being tracked. In frame 1635 (Fig. 4.11(c)), all four cars are clearly segmented and tracked. Occlusion causes track 37 to be lost behind the tree in Fig. 4.11(d), but the matching algorithm finds the best match by erroneously assigning track number 37 to the blob that should ideally be tracked as number 41. Both these errors are caused because velocity has not been considered as one of the factors in the tracking algorithm.


**Figure 4.10.** Error when velocity is not considered - frame 0130

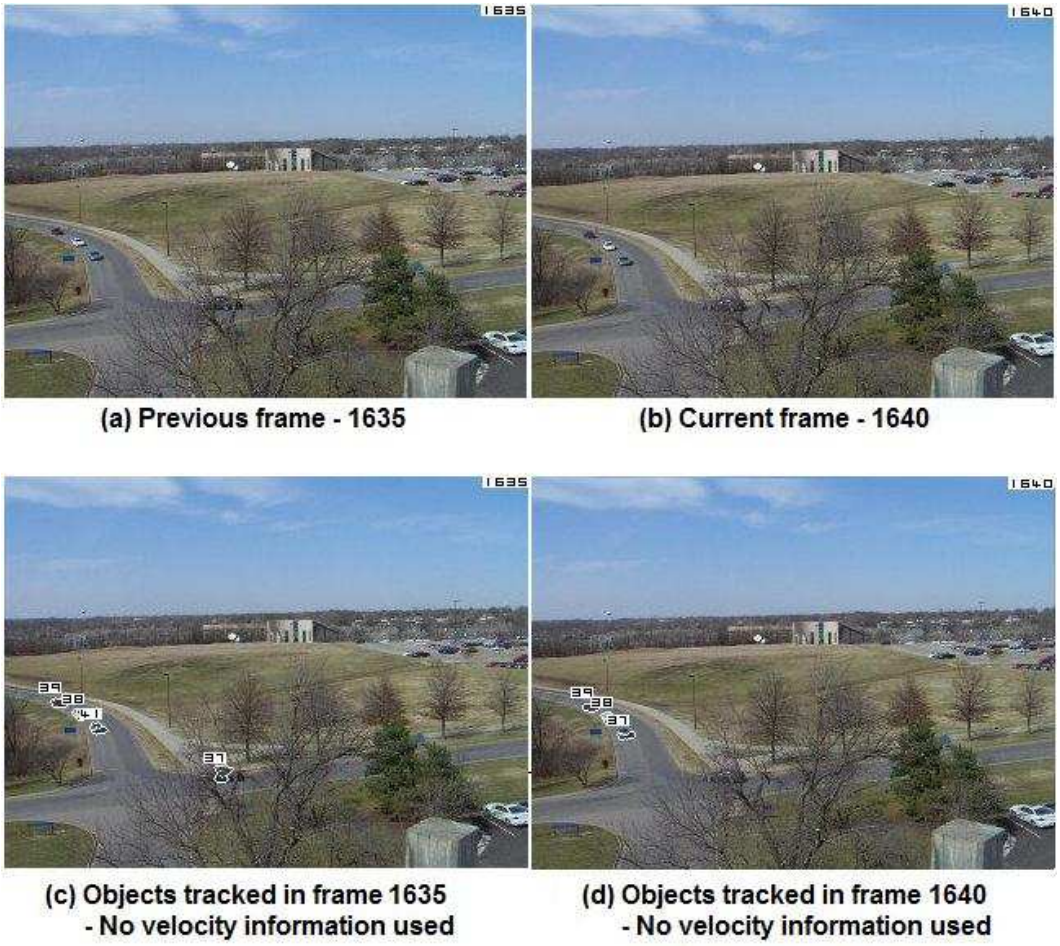



**Figure 4.11.** Error when velocity is not considered - frame 1640

One of the strongest visual clues that humans use for tracking objects is the idea of continuity and flow. Blobs are unlikely to suddenly change velocity in the video sequence. The closer the velocity vector of a candidate blob to the track velocity, the better the match between the candidate blob and the track should be. We incorporate a Velocity factor that calculates the congruity in velocity for each possible track-to-blob assignment. Depending on the closeness of the blob velocity vector to a track's most recent velocity vector estimate, we increase or decrease the match matrix value for that particular assignment using a multiplicative factor.

Use of velocity vector as a metric to decide closeness of match between candidate blobs is not a new concept. It is a common feature of many tracking algorithm implementations [18] [31]. In most applications, velocity difference between blobs is treated as another feature of the blob along with its position values (Y, X coordinates) and its color values (R, G, and B). A weighted sum of difference between these values is used as the measure of closeness between the concerned blobs. Rather than using velocity values as another feature in the distance calculation, we use velocity as a factor to scale the match matrix values down (or up), depending on how similar (or different) the velocities of the blob and track are.

The Velocity factor function that we used is plotted against the difference between blob and track velocities in Fig. 4.12. The Velocity factor values range from 0.5 to 5. The Velocity factor was arrived at by inverting a function linearly decreasing with velocity difference values (shown in inset). The linearly decreasing function weights smaller velocity differences more than larger velocity differences. Since we are using a distance measure in the match matrix, a smaller velocity difference should result in a smaller distance measure. Thus, the inverse of the linearly descending function is used as the Velocity factor. Multiplying by the

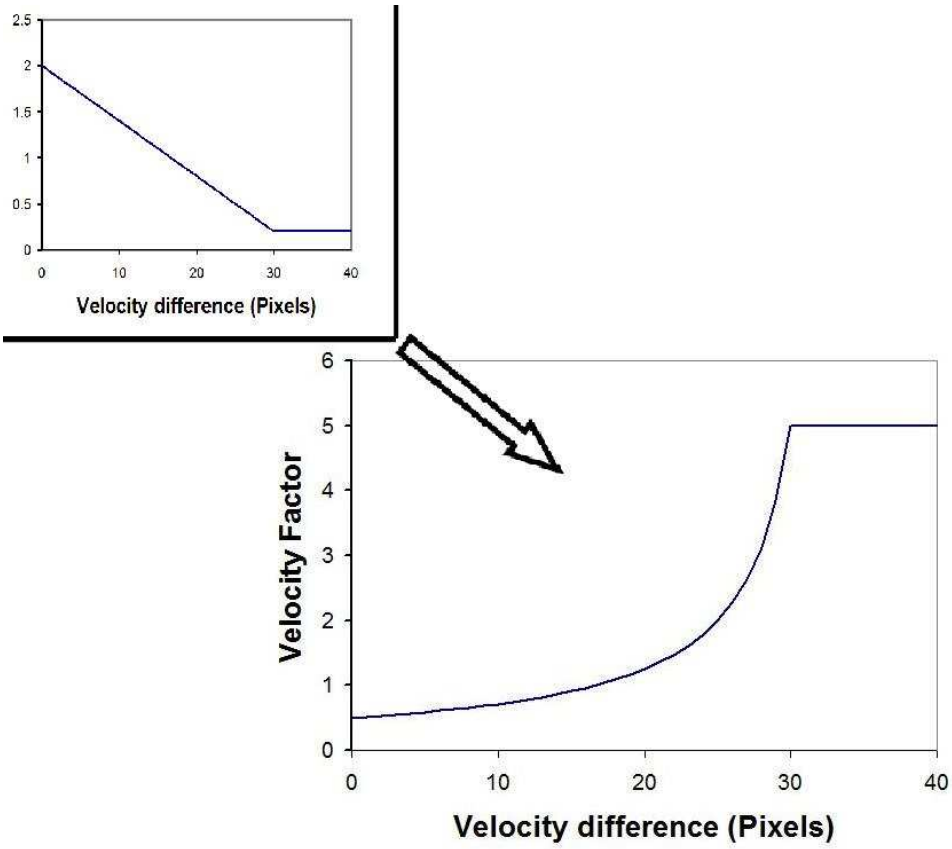



**Figure 4.12.** Velocity factor vs. observed difference in velocity

Velocity factor is equivalent to dividing by the linearly decreasing function in the inset.

For each active track from the previous frame, the velocity in Y and X directions is calculated by averaging the motion in Y and X directions over the previous 5 time steps. For each possible track-to-blob assignment in the current frame, the blob velocity vector that would result from the assignment is calculated. The absolute difference between the previous track velocity vector and the current blob velocity vector in Y and in X directions is used to decide the factor by which to multiply the match matrix value for the concerned assignment. The smaller the difference in velocity vector values, the smaller the multiplicative factor. Thus,

the Euclidean distance in the match matrix between a track-blob pair that are close to each other in velocity will get smaller due to the small multiplicative factor (close to 0.5). Conversely, the Euclidean distance in the match matrix for a track-blob pair that has a high difference in their velocity values will get multiplied by a higher multiplicative factor (greater than 1.5), and thus become larger. The match matrix, thus, gets adjusted to favor track-blob assignments that are closer in their velocity values.

The tracking results after use of the Velocity factor in the error cases that were explained earlier in figures 4.10 and 4.11 are shown in figures 4.13 and 4.14, respectively. It can be seen in Fig. 4.13(b) that car 07 is correctly labeled and car 03 is correctly declared as 'lost'. In Fig. 4.14(b), cars 55, 54, and 59 are correctly tracked and car 53 is declared as 'lost'.


**Figure 4.13.** No error in assignment when the Velocity factor is used - frame 0130


**Figure 4.14.** No error in assignment when the Velocity factor is used - frame 1640

# Chapter 5

# Results after improvements

In chapter 4, images were presented to show some cases where the merge module and the Velocity factor showed improved results in segmentation and tracking. There are several other cases in the video sequence where improvement in results are seen due to the use of the merge module and the Velocity factor. Qualitative improvements in the system were easily visible in the results. To arrive at a quantitative method to measure the results was much more difficult, given the nature of our data and the application. The data is raw surveillance video footage from which objects are to be segmented before they are tracked. Thus, there was no ground truth data that could be used for reference.

## 5.1 Ground truth data

Since the data used in our system is from real-life traffic scenes and not artificially synthesized, arriving at a comparison metric to measure accuracy and improvement was a difficult task. There are various stages in the tracking system, with each stage affecting the final tracking results. For instance, if the segmenta-

tion algorithm is modified slightly, it changes the number of blobs detected. This changes the track numbers of the blobs in the results. Similarly, blob merging algorithm, depending on the threshold, can merge different number of blobs. This can affect the track numbers, positions, and color statistics. To arrive at the ground truth for the final results was an important task.

Given the number of variables in the system, we decided that manual observation of the sequence and the results was the only reasonable method to arrive at a quantitative comparison metric to measure the improvements in the system. Video sequence 1 showing cars through an intersection, shown in Fig. 5.1, was used as the sequence for analysis. The sequence consists of 1780 frames. Since it would be extremely time consuming to manually observe the entire sequence for different runs of the algorithm, smaller subsequences from the whole sequence were chosen for analysis. Three subsequences in the video sequence were observed manually to arrive at the ground truth.

Subsequence 1, a frame of which is shown in Fig. 5.2, involves 4 cars being tracked between frames 0200 and 0400. This is a relatively easy section in the video as far as tracking is concerned because there is no crowding of cars in this section. Subsequence 2, whose example frame is Fig. 5.3, is a very easy section of the video where a single car is tracked from frame 0550 through frame 0830. Finally, subsequence 3 is a section of video between frames 1600 and 1750 where 4 cars pass through the intersection at the same time. This is a difficult section of the video for tracking purposes because of the crowding at the intersection, as can be seen in Fig. 5.4. The sequences were chosen to represent different situations that may occur during tracking. While subsequence 1 is a normal section at the beginning of the video sequence, subsequence 2 is an easy section for tracking in


**Figure 5.1.** Scene from video sequence 1 used for analysis


**Figure 5.2.** Subsequence 1 from video sequence used for analysis

the middle section of the video sequence, and subsequence 3 is a difficult section at the end of the video.

A spreadsheet was maintained that kept record of whether a human was able to detect and identify the cars in the three subsequences of the raw surveillance video.


**Figure 5.3.** Subsequence 2 from video sequence used for analysis


**Figure 5.4.** Subsequence 3 from video sequence used for analysis

## 5.2 Comparison metric

Blob merging impacts the system by merging blobs that may have been incorrectly separated by the segmentation algorithm. The merging algorithm affects the number of blobs a single object is broken into. Thus, to measure the accuracy of the merging algorithm, the number of errors in blob detection is a reasonable metric. Hence, one of the manual observations made in each frame of the results is

the number of blobs that an object in the video is segmented as. If an object is detected as more than one blob, then a '*number-of-blobs error*' count is incremented. For example, if an object is broken down into $k$ blobs, then the *number-of-blobs error* count is incremented by $(k-1)$. Sometimes, though a human might be able to detect the object from the image, the automatic background subtraction algorithm may fail to do so. These cases occur when the object has stopped at the intersection or has been merged with another object. In such cases, the error count is incremented by 1, because an object that had to be detected as 1 blob was not detected at all.

The Velocity factor affects the tracking stage of the algorithm. To measure the errors in the tracking stage, the track number assigned to each object in the frame is observed over the entire subsequence. A good tracking algorithm should (a) maintain the same index for the object throughout the sequence, and (b) in case the track number changes for some reason, the new track number should persist for the remainder of the sequence. A single metric cannot capture both these cases, thus we decided to use two metrics. One metric measures the number of errors in the index assignment for an object, while another measures the number of times the track number for an object changes from one frame to the next. We call these errors the '*index error*' and the '*swap error*', respectively. *index error* counts the number of instances in which an object is given a track number that is different from the best track number for it in the sequence. The best track number for an object is the track number that was assigned most number of times to the object in the subsequence. Every time the object number changes from one frame to the next, the *swap error* is incremented. Ideally, both these numbers should be small.

Fig. 5.5 illustrates how the errors would be calculated for car 1 in subsequence 1. The *number-of-blobs error* count is incremented whenever the car is segmented as more than or less than one blob. In frame 0280, the car is not assigned to any track, implying that it was not detected. Thus, the *number-of-blobs error* count is incremented by 1. In frames 0295 and 0315, the car is segmented as more than one blob and each blob is assigned to different track numbers. Thus, the *number-of-blobs error* count is incremented. In case of frame 0315, the error count is incremented by a value of 2 because the single car was segmented as three separate blobs.

| | **Car # 1** | | | |
|---|---|---|---|---|
| | | Errors | | |
| Frame | Track number assigned | number of blobs | index | swap |
| 0275 | 6 | 0 | 1 | 0 |
| 0280 | 0 | 1 | 0 | 0 |
| 0285 | 7 | 0 | 0 | 1 |
| 0290 | 7 | 0 | 0 | 0 |
| 0295 | 6,7 | 1 | 0 | 0 |
| 0300 | 7 | 0 | 0 | 0 |
| 0305 | 6 | 0 | 1 | 1 |
| 0310 | 7 | 0 | 0 | 1 |
| 0315 | 7,8,9 | 2 | 0 | 0 |
| | Total errors | 4 | 2 | 3 |

**Figure 5.5.** Illustration: calculating errors for car number 1 in subsequence 1

By observing the track numbers assigned, we can see that the best number for car 1 is 7 because it appears most number of times as the track number for the car. The *index error* count is incremented in each frame that the car is assigned to a number other than 7. The *swap error* is incremented in frames 0285, 0305, and 0310 because the track number assigned to the car in the frames changes from 6 to 7, 7 to 6, and 6 back to 7 in these frames, respectively.

## 5.3 Results after merge module

The resulting number of segmentation errors (*number-of-blobs error*) for different cases of the merging module are given in table 5.1. The errors as a percentage of number of car instances in each subsequence are given in table 5.2. The table 5.1 shows that by use of blob merging, the number of errors is reduced in all three subsequences. The least number of errors in segmentation errors for subsequence 1 is achieved by using blob merging with threshold of 10 (23/116 or 19.8% error). For subsequence 2, blob merging with threshold of 10 and 20 show best results (2/55 or 3.6% error). The most improvement is seen in subsequence 3, where blob merging with a threshold of 15 results in error count of 19 (or 15.7% error). This is a reduction of 10 errors from an original no-blob-merging case of 29 errors, an improvement of 34.5%. In total numbers, all the blob merging cases show improvement over the no-blob-merging case, the best results being a reduction of 10 errors in total by use of a threshold of 15 (reduction from original count of 58, which is a 17.2% improvement). In case of subsequence 2, however, blob merging with thresholds of 15 and 20 result in increase in number of errors. This is due to the shortcomings of a single threshold being used for the entire image as discussed in section 4.1.1.

It may be noted that the improvement in segmentation is despite the serious shortcomings of the position thresholds. By use of the Vicinity Factor, explained in chapter, 6, even better results can be expected.

| number-of-blobs errors in segmentation and merging | | | | |
|---|---|---|---|---|
| [Frames] (Number of instances of cars) | Subsequence 1 $[0200-0400]$ (116) | Subsequence 2 $[0550-0830]$ (55) | Subsequence 3 $[1600-1750]$ (121) | Total $1+2+3$ (292) |
| No Blob merging | 24 | 5 | 29 | 58 |
| Blob merging (threshold= 10) | **23** | 5 | 23 | 51 |
| Blob merging (threshold= 15) | 27 | **2** | **19** | **48** |
| Blob merging (threshold= 20) | 34 | **2** | 20 | 56 |

**Table 5.1.** Segmentation errors in video sequence 1

| Percentage errors (against number of car instances) in segmentation and merging | | | | |
|---|---|---|---|---|
| [Frames] (Number of instances of cars) | Subsequence 1 $[0200-0400]$ (116) | Subsequence 2 $[0550-0830]$ (55) | Subsequence 3 $[1600-1750]$ (121) | Total $1+2+3$ (292) |
| No Blob merging | 20.7 | 9.1 | 24.0 | 17.9 |
| Blob merging (threshold= 10) | **19.8** | 9.1 | 19.0 | 16.0 |
| Blob merging (threshold= 15) | 23.3 | **3.6** | **15.7** | **14.2** |
| Blob merging (threshold= 20) | 29.3 | **3.6** | 16.5 | 16.5 |

**Table 5.2.** Segmentation errors as percentage of total car instances in video sequence 1

## 5.4 Results after the Velocity factor

This section discusses quantitative improvements in tracking results brought about by the use of the Velocity factor. The two metrics - *index error* and *swap error* are presented separately in tables 5.3 and 5.5, respectively. The errors as a percentage of number of cars in the subsequences are tabulated in tables 5.4 and 5.6.

Tables 5.3 and 5.4 show that by use of the Velocity factor, in subsequence 1, slight improvement (2/116 or 1.7%) in *index error* counts is seen. In subsequence 2, the use of the Velocity factor produces an error while no *index errors* result when it is not used. This is a small drop in accuracy (1.8%). In subsequence 3, however, there is a significant improvement in *index error* counts. The number of errors falls from 32 when the Velocity factor is not used, to 13 when it is used. This is an improvement of 59.4%. When all three cases are summed, the number of errors when the Velocity factor is used reduces from 45 to 25, an improvement of 44.4%. Thus, the use of the Velocity factor adds significant improvements to the system.

| Errors in index assignment | | | | |
|---|---|---|---|---|
| | Subsequence 1 | Subsequence 2 | Subsequence 3 | Total |
| [Frames] | $[0200-0400]$ | $[0550-0830]$ | $[1600-1750]$ | $1+2+3$ |
| (Number of instances of cars) | (116) | (55) | (121) | (292) |
| Without Velocity factor | 13 | **0** | 32 | 45 |
| With Velocity factor | **11** | 1 | **13** | **25** |

**Table 5.3.** Index errors in video sequence 1

| Percentage Errors (against number of car instances) in index assignment | | | | |
|---|---|---|---|---|
| | Subsequence 1 | Subsequence 2 | Subsequence 3 | Total |
| [Frames] | $[0200-0400]$ | $[0550-0830]$ | $[1600-1750]$ | $1+2+3$ |
| (Number of instances of cars) | (116) | (55) | (121) | (292) |
| Without Velocity factor | 11.2 | **0.0** | 26.4 | 12.6 |
| With Velocity factor | **9.5** | 1.8 | **10.7** | **7.3** |

**Table 5.4.** Index errors as percentage of total car instances in video sequence 1

Counts and percentages of *swap error* when the Velocity factor is used are tabulated in tables 5.5 and 5.6, respectively. There is no improvement when the Velocity factor is used in subsequence 1. In subsequence 2, use of the Velocity factor results in 2 errors in an otherwise errorless tracking result. In subsequence 3, as with the *index error*, the *swap error* count reduces significantly - 6 errors less than the original error count of 13 which is a 46.2% improvement. Total errors in all three subsequences reduces from 18 without use of the Velocity factor to 14 with its use, an improvement of 22.2%.

| Number of Swap errors in assigning track numbers to blobs in consecutive frames | | | | |
|---|---|---|---|---|
| | Subsequence 1 | Subsequence 2 | Subsequence 3 | Total |
| [Frames] | $[0200-0400]$ | $[0550-0830]$ | $[1600-1750]$ | $1+2+3$ |
| (Number of instances of cars) | (116) | (55) | (121) | (292) |
| Without Velocity factor | **5** | **0** | 13 | 18 |
| With Velocity factor | **5** | 2 | **7** | **14** |

**Table 5.5.** Swap errors in video sequence 1

| Percentage Swap errors (against number of car instances) in blob assignment | | | | |
|---|---|---|---|---|
| | Subsequence 1 | Subsequence 2 | Subsequence 3 | Total |
| [Frames] | $[0200-0400]$ | $[0550-0830]$ | $[1600-1750]$ | $1+2+3$ |
| (Number of instances of cars) | (116) | (55) | (121) | (292) |
| Without Velocity factor | **4.3** | **0.0** | 10.7 | **5.0** |
| With Velocity factor | **4.3** | 3.6 | **5.8** | **4.6** |

**Table 5.6.** Swap errors as percentage of total car instances in video sequence 1

# Chapter 6

# Depth variation in the scene: development of the Vicinity Factor

Objects being tracked are at different distances from the camera in the real world. Due to the projection of the 3-D world onto a 2-D image plane, real life distances are not consistent across the imaged scene. Depth information is lost in the image plane. Because of perspective projection in imagery, objects far away appear smaller, and the distance (in pixel space) between far-from-camera objects is smaller than the distance between close-to-camera objects.

## 6.1 Problems due to depth variation in the scene

Since most tracking algorithms use the position information (Y and X coordinates) in some way or the other, this variation in depths in different parts of the scene can affect tracking algorithms adversely. We alluded to this problem in

chapter 5 in the context of the position thresholds that are used in the blob merging algorithm. There is a need for adjusting the thresholds used for the merging algorithm in different regions of the scene to reflect the real world 3-D distances in these parts. For example (see Fig. 6.1), the threshold for 'nearness' around car 02, which is nearer to the camera, must be greater than the threshold around car 01, which is far away.

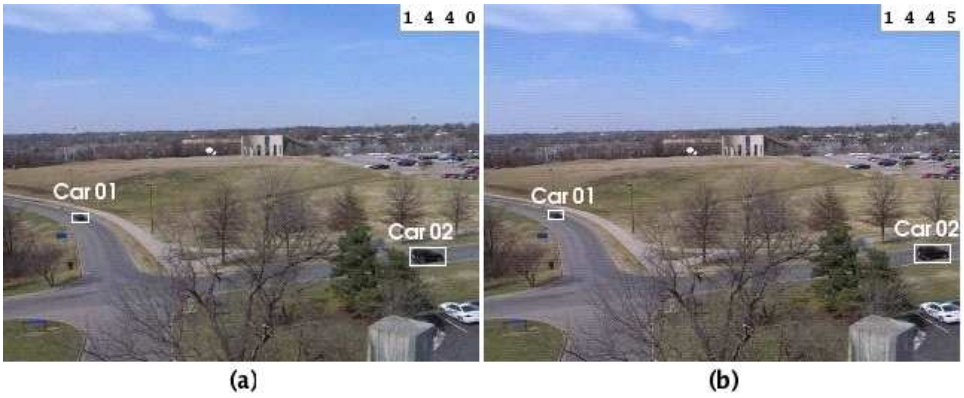



**Figure 6.1.** Different distances between camera and the objects- The depth variation problem

Some modeling of the real world can greatly increase the accuracy of the detection and tracking results. 3-D modeling of scenes is a challenging research area and various approaches have been presented in literature. Camera calibration, particularly the estimation of camera's focal length, is an important aspect in 3-D modeling methods. To arrive at 3-D models for scenes, some methods use a fully calibrated camera [29] while others use an automatic camera calibration method [27] [30]. While some methods use multiple views of the same scene and implement stereo matching algorithms to arrive at 3-D world model [27] [28], others use monocular image sequences [29] [30].

[27] and [28] discuss modeling of 3-D scenes and depth estimation by use of multiple views of the scene. [29] develops a method for 3-D modeling of traffic

scenes by using multiple views of the scene, a calibrated camera system, and some prior knowledge about background regions and foreground objects. We do not use a calibrated camera approach for estimating object depth. Since we use a single view of the scene, methods involving multiple views of the scene cannot be applied. [30] presents a method for estimating 3-D structure from observing motion of some feature points in a video sequence. Translation and rotation models are used to estimate 3-D structure and motion of tracked features. The translational model may be extended for use in vehicle tracking applications. However, the model and equations are based on principles of geometric modeling and graphics. Since our application demands only the estimation of relative 3-D distances across different regions in the scene, we do not use these complicated models for estimating object depth. Our method, explained in section 6.2, offers a simpler alternative based on the observed track motion in different parts of the scene.

Explicit 3-D modeling methods discussed above are specific to the scene that is being observed and can be complicated. While they may have their advantages and offer a detailed 3-D model of the scene, for the task of tracking in static camera scenes, we suggest an alternative to 3-D modeling. Since our main requirement for 3-D modeling is to estimate how close objects really are in the 3-D world, we use a feedback approach to automatically learn this variation across the scene from motion seen in previous frames. Because of the efficacy of our method and its ease of implementation, we are able to avoid the use of more complicated 3-D modeling methods for this task.

## 6.2 The Vicinity Factor

The innovation in our set of algorithms is in gathering knowledge from the tracking stage and using it to estimate 3-D world distances corresponding to different regions in the image plane. This estimate can be used in the blob-merging stage by applying the learned distance information to tune the position thresholds being used in the blob-merge algorithm. Our solution is based on the observation that close-by blobs move "faster" (in image space) than far-away blobs in the scene. Keeping track of the change in position of blobs in different regions of the image can help us get an idea of relative distances in the real world plane corresponding to these regions. We use this positional change to estimate what we call the Vicinity Factor over the image. We break the image into grids of 30 by 30 pixels in Y and X coordinates respectively, and calculate the Vicinity Factor for each cell in the grid. Initially, the value for Vicinity Factor over the entire grid is set to a predetermined average value. Every time a successful frame-to-frame blob match is found for a track, the Vicinity Factor for that region in the image is increased (or decreased) depending on whether the track has moved more (or less) than the average motion for the whole image. After a few hundred frames, thus, the Vicinity Factor reflects the amount of motion that is observed in each cell of the grid. Thus, regions in the image grid that correspond to regions in the 3-D world close to the camera tend to show higher values for the Vicinity Factor than regions in the image that correspond to regions of the 3-D world far away from the camera.

The concept is explained in Fig. 6.2, where car 02, which is nearer to the camera, must be treated with a larger position threshold than the threshold used for car 01, which is far away. From frames 1440 and 1445, it can be seen that

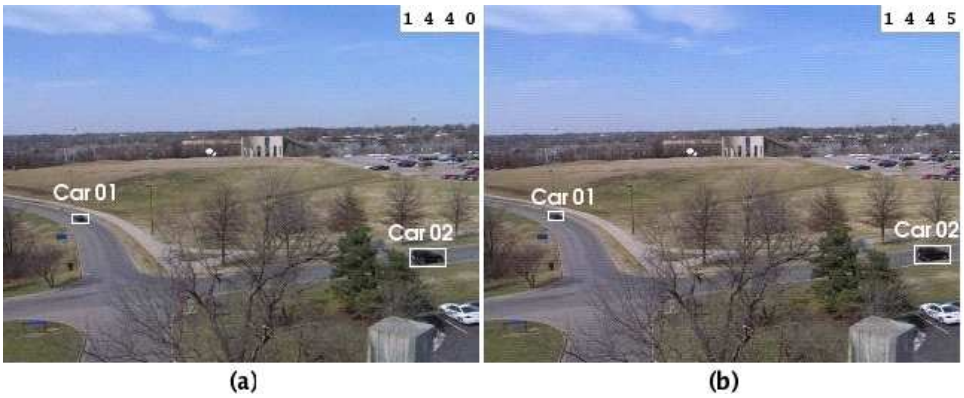



**Figure 6.2.** Illustration of the depth variation problem

the nearer object (car 02) moves much more (in pixel space) than the far away object (car 01). This difference in motion is utilized to arrive at the Vicinity Factor in these regions (detailed explanation in section Fig. 6.3). Given a base threshold for 'nearness' as 10 pixels, the Vicinity Factor for the region around car 02 (the close region) over time becomes 4. The threshold to classify two blobs as near to each other would then be 40 pixels. But in the region around car 01 (the far region), the Vicinity Factor over time becomes 2, and the threshold for 'nearness' in this region would be 20. Thus, the adjusted threshold reflects the relative real world distances in these two regions. Essentially, we are estimating relative 3-D distances over the scene from observations and using them to adjust our expectation of vehicle speed observed in the 2-D image space.

## 6.3 Vicinity Factor calculation

The Vicinity Factor, which is an estimate of relative 3-D distance and vehicle speed, is iteratively calculated for regions over the image. We divide the image into a grid of size $(Ydimension/binsizeY)$ rows by $(Xdimension/binsizeX)$ columns, where $Ydimension$ and $Xdimension$ are the image dimensions and $binsizeY$ and

$binsizeX$ are the sizes of the grid (in pixels) along Y and X directions, respectively. The Vicinity Factor in Y and X directions is calculated separately ($vicinityY$ and $vicinityX$).

For each successfully updated track in frame $k$, the Vicinity Factor grid is updated at the position corresponding to the position of the track in the grid. We set both a minimum and maximum possible value for the Vicinity Factor as 1 and $maxFactor$, respectively, based on a reasonable estimate of maximum expected object speed in the scene. The Vicinity Factor values vary continuously from 1 to $maxFactor$.

To initialize the Vicinity Factor:

$$vicinityY[g_y][g_x] = vicinityX[g_y][g_x] = \frac{1 + maxFactor}{2} \tag{6.1}$$

where

$g_y \in \{0, 1, 2, \ldots, (\frac{Ydimension}{binsizeY} - 1)\}$,

$g_y \in \{0, 1, 2, \ldots, (\frac{Ydimension}{binsizeY} - 1)\}$,

$vicinityY[g_y][g_x]$ represents the Vicinity Factor in Y direction at column $g_y$ and row $g_x$ in the grid, and

$vicinityX[g_y][g_x]$ represents the Vicinity Factor in X direction at column $g_y$ and row $g_x$ in the grid.

To update the Vicinity Factor:

For every successfully updated track $t_j^k \in T^k$,

$$\Delta y = y(t_j^k) - y(t_j^{k-1}) \tag{6.2}$$

where $y(t_j^k)$ is the position of track $t_j^k$ in the Y direction ,and

$$\Delta x = x(t_j^k) - x(t_j^{k-1}) \tag{6.3}$$

$x(t_j^k)$ is the position of track $t_j^k$ in the X direction. The update equation is:

$$\begin{aligned} vicinityY[g_y][g_x] &= vicinityY[g_y][g_x]_{old} + \\ &\quad ((\Delta y - 1) \times (maxFactor - minFactor) \times constant) \\ &= vicinityY[g_y][g_x]_{old} + ((\Delta y - 1) \times (maxFactor - 1) \times \alpha) \\ &\approx vicinityY[g_y][g_x]_{old} + ((\Delta y - 1) \times maxFactor \times \alpha) \end{aligned} \tag{6.4}$$

and likewise,

$$vicinityX[g_y][g_x] = vicinityX[g_y][g_x]_{old} + ((\Delta x - 1) \times maxFactor \times \alpha) \tag{6.5}$$

where $\alpha$ is a constant that determines the rate of learning. We choose $\alpha = 0.1$.

The Vicinity Factor in X and Y directions for video sequence 1 are shown in Fig. 6.3. Fig. 6.3(a) is the Vicinity Factor in X direction and Fig. 6.3 is the Vicinity Factor in Y direction. Fig. 6.3(c) shows the scene and grid layout in the sequence. The initial value for the Vicinity Factor for the entire grid was set to 3. As the video sequence progresses, the values at different locations in the grid get adjusted to result in the final values as shown in the figure. As can be expected, the factor is higher in regions closer to the camera and where there is larger motion of objects. Also, in areas of the scene where there is no tracked motion, the values of the Vicinity Factor are unchanged from the initial value of 3. These areas are shown as unshaded in the grids in figures 6.3(a) and (b). It may be noted that the

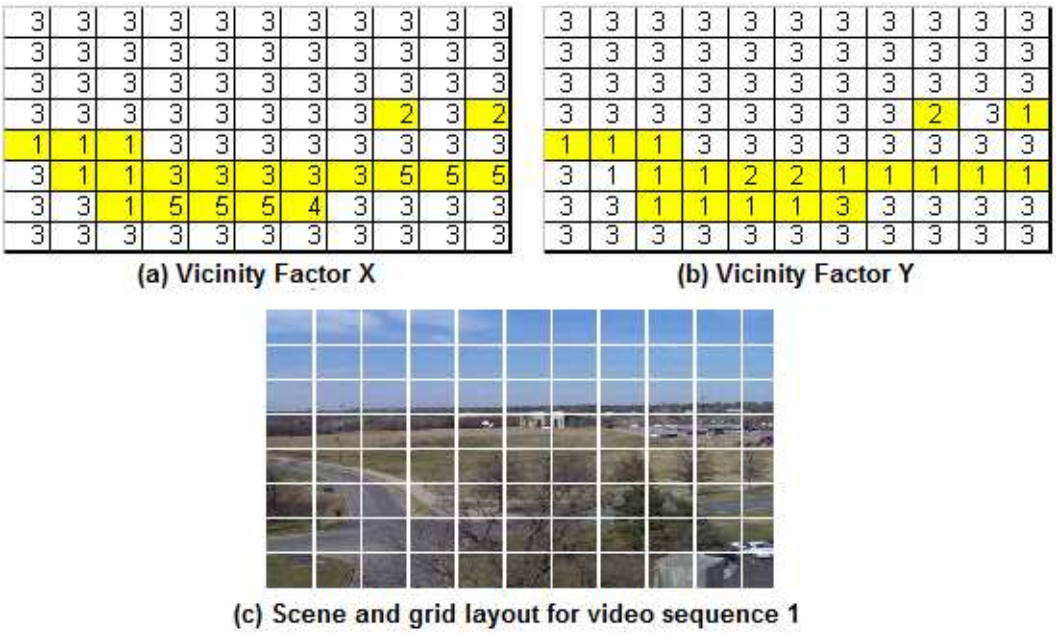



**Figure 6.3.** The automatically learned Vicinity Factor for video sequence 1

vicinity may differ in X and Y dimensions because in some areas the object may move only in the vertical direction with little horizontal motion and vice versa. Of significance is the fact that the Vicinity Factor was automatically learned by the algorithm.

We found that over a period of time, the Vicinity Factor reflects the 'real' nature of distances as they are in the 3-D world. We use this Vicinity Factor to assess the closeness of blobs for the blob merging stage. Use of the Vicinity Factor results in a much more accurate blob merging algorithm. Fig. 6.4 shows how the use of the Vicinity Factor is helpful in achieving a correct merging of blobs. In Fig. 6.4(a), three cars get incorrectly merged together as one and another car is split into two pieces. Fig. 6.4(b) shows the correct output obtained by the use of the Vicinity Factor. A hard threshold would be insufficient for handling all potential merge cases appropriately.


**Figure 6.4.** Improvement in blob merging due to use of the Vicinity Factor

# Chapter 7

# Vicinity Factor results

This chapter discusses improvements brought about by the Vicinity Factor. The improvements in segmentation and blob merging stages of the system are shown using the quantitative measures that were introduced in section 5.2.

## 7.1 Results after using the Vicinity Factor

The *number of blobs error*, explained in section 5.2, is the measure we use to determine accuracy of the segmentation and the blob merging processes. In section 5.3, the improvement brought about in the system by use of blob merging was discussed. To compare the performance of the blob merging after the Vicinity Factor has been implemented, the tables of section 5.3 are presented again with the addition of a row for results after the use of the Vicinity Factor. Table 7.1 shows errors that result in the segmentation when blob merging is not used along with errors that result from the use of blob merging with various thresholds and blob merging with Vicinity Factor. Table 7.2 shows the same errors as a percentage of number of car instances in the subsequences.

In two of the three subsequences (1 and 2), the use of blob merging with the Vicinity Factor produces the best results. The total numbers show that the use of the Vicinity Factor results in the least number of errors. Use of the Vicinity Factor brings about an improvement of 3 errors from the earlier best case of 48 errors that resulted from use of blob merging with a threshold of 15.

An important point to be made is that subsequence 1 occurs in the early part of the video (frames 0200 to 0400). At that stage, the Vicinity Factor has not yet learned the distance information adequately. A few of the errors that occur in subsequence 1 are due to the insufficient learning. The error numbers that the use of the Vicinity Factor shows for subsequence 1 may be seen as the worst-case performance of the Vicinity Factor. Interestingly, even the worst-case performance with use of the Vicinity Factor produces the best results (least errors) in segmentation.

| Errors in segmentation and merging | | | | |
|---|---|---|---|---|
| [Frames] (Number of instances of cars) | Subsequence 1 $[0200-0400]$ (116) | Subsequence 2 $[0550-0830]$ (55) | Subsequence 3 $[1600-1750]$ (121) | Total $1+2+3$ (292) |
| No Blob merging | 24 | 5 | 29 | 58 |
| Blob merging (threshold= 10) | **23** | 5 | 23 | 51 |
| Blob merging (threshold= 15) | 27 | **2** | **19** | 48 |
| Blob merging (threshold= 20) | 34 | **2** | 20 | 56 |
| Blob merging (Vicinity Factor) | **23** | **2** | 20 | **45** |

**Table 7.1.** Segmentation errors in video 1 along with use of the Vicinity Factor

| Percentage Errors (against number of car instances) in segmentation and merging | | | | |
|---|---|---|---|---|
| [Frames] (Number of instances of cars) | Subsequence 1 $[0200-0400]$ (116) | Subsequence 2 $[0550-0830]$ (55) | Subsequence 3 $[1600-1750]$ (121) | Total $1+2+3$ (292) |
| No Blob merging | 20.7 | 9.1 | 24.0 | 17.9 |
| Blob merging (threshold= 10) | **19.8** | 9.1 | 19.0 | 16.0 |
| Blob merging (threshold= 15) | 23.3 | **3.6** | **15.7** | 14.2 |
| Blob merging (threshold= 20) | 29.3 | **3.6** | 16.5 | 16.5 |
| Blob merging (Vicinity Factor) | **19.8** | **3.6** | 16.5 | **13.3** |

**Table 7.2.** Segmentation errors as percentage of total car instances in video 1

## 7.2 Applications

The Vicinity Factor, being an estimate of the real world distances in Y and X directions in the scene, can be used in different ways to achieve better results in the tracking system.

### Blob merging

We have already discussed the use of the Vicinity Factor in the blob merging process. It enables the merging module to make a better decision whether two

blobs need to be merged based on their relative positions in the image.

### Object size estimation

We can estimate relative object size in different parts of the scene by using the values of the Vicinity Factor in these parts. The square root of sum of squares of Y and X values of the Vicinity Factor can be a reasonable estimate for relative object size in different parts of the scene. Fig. 7.1 shows the square root of the sum of squares of the Vicinity Factor values in X and Y directions in the parts of the scene that exhibit object motion. The values in the matrix of Fig. 7.1(d) are good approximations of relative object size in the corresponding locations in the grid layout of the scene.

### Detecting active regions of the scene

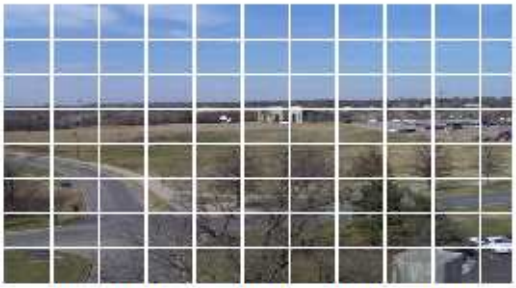
Results show that in the grid locations where no real and trackable object motion was tracked, the Vicinity Factor remains unchanged from the initialized value. Only tracked objects affect the Vicinity Factor values. In some parts of the scene, noise causes false detection of objects. But since these are false detections, these objects are not reliably tracked by the tracking algorithm. Since the tracking algorithm does not track the noise blobs, these blobs do not affect the Vicinity Factor values considerably. The Vicinity Factor remains largely unchanged in noisy regions of the scene. Thus, by observing the values in the Vicinity Factor matrix, we can identify 'interesting' and 'active' regions of the scene.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 2 |
| 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 5 | 5 | 5 |
| 3 | 3 | 1 | 5 | 5 | 5 | 4 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

(a) Vicinity Factor X

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 1 |
| 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 |
| 3 | 3 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

(b) Vicinity Factor Y



(c) Scene and grid layout for video sequence 1

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | 2.8 | | 2.2 |
| 1.4 | 1.4 | 1.4 | | | | | | | | |
| | 1.4 | 1.4 | 3.6 | 3.6 | 3.2 | 3.2 | 3.2 | 5.1 | 5.1 | 5.1 |
| | | 1.4 | 5.1 | 5.1 | 5.1 | 5.0 | | | | |
| | | | | | | | | | | |

(d) Square root of sum of squares of Vicinity Factor X and Y values in active parts of the scene

**Figure 7.1.** Estimation of object size using the Vicinity Factor values

## 7.3 Drawbacks

The basis for calculating the Vicinity Factor is that different objects move with the same velocity in the video. An object moving very fast (or very slowly) can briefly alter the Vicinity Factor calculation by exhibiting more (or less) motion than the average object in that part of the scene. The learning rate $\alpha$ controls the contribution of each track instance to the Vicinity Factor. Thus, the Vicinity Factor calculation is not adversely affected by the rare occurrence of a fast or slow

moving object. For example, in our domain of traffic surveillance, most vehicles move with the same velocity, with the occasional speeding or slow moving car. In fact, the Vicinity Factor may be useful in identifying such anomalous object behavior in the video.

Since it is based on average motion values in different parts of the image, it may not be useful in applications where there is high variance in motion values of objects. This is because high variance in the motion values of different objects would mean that the algorithm would not be able to learn a reasonable average motion value for the Vicinity Factor. However, in case of traffic scenarios, most vehicles move in the scene with similar velocities (due to speed limits on the road) and thus, the algorithm gives good results.

If there are frequent occurrences of slower moving objects (like people) in the same regions as faster objects (like cars) in the same part of the image, then the Vicinity Factor, as implemented in this thesis cannot be directly used. An improved Vicinity Factor model maintaining separate Vicinity Factor matrices for each class of objects (humans, cars, bikes, etc.) could be a solution in such cases.

# Chapter 8

# Bayesian tracking algorithm

Tracking is essentially a correspondence task, the goal of which is to find a match for each blob or object in the previous frame to one of the blobs or objects in the current frame. The tracking algorithm of section 3.5 was based on the match matrix values and the track assignments were made by looking for the blob which was nearest (in Euclidean distance) to the track. Given the various factors of uncertainty in the segmentation and tracking process, some of which are listed in section 3.5.3, a probabilistic assignment of tracks to blobs can be more useful for future research.

## 8.1 Need for probabilistic assignment

Since the video shows high amount of noise, occlusion, and sudden appearance and disappearance of objects, a probabilistic track assignment algorithm can lead to better inference about the object behavior in higher-level intelligence applications. For example, if two tracks have converged into a single blob in a particular frame, a matrix of probabilities of track-to-blob associations would be able to of-

fer a better numerical representation of the situation making it easier to infer the 'merge' event. Similarly, if an object is occluded or leaves the scene, a probabilistic algorithm can give the probability that the track is 'lost'. The distance-based approach that we described earlier cannot be extended to find the probability that a track has been 'lost'.

We felt a strong need for a probabilistic tracking algorithm, and thus, designed a Bayesian inference method to assign tracks to blobs. In the new algorithm, color and position information of the moving blobs is used as observation in a Bayesian framework to assign tracks to blobs. Based on the observations, track-to-blob association probabilities are calculated for all the active tracks and blobs. When the information from the blobs in a frame does not support the presence of an object which was seen in the previous frame, we allow for the corresponding track to be declared 'lost'. If the object resurfaces in subsequent frames, the system reassigns the track to the object. Thus, the Bayesian method is able to handle occlusion, disappearance of objects, sudden changes in object velocity, and changes in object color profile and size.

## 8.2 Background

Bayes formula, proposed by Reverend Thomas Bayes (1702-1761), basically solves the problem of "inverse probability". Often, the "forward probability" of an observation resulting from a hypothesis is known. By using the Bayes formula, given an observation, the "posterior" probability of a hypothesis can be calculated.

The Bayes formula is given by:

$$p(A|B) = \frac{p(B|A) \times p(A)}{p(B)} \tag{8.1}$$

where $(A|B)$ implies "Event $A$ given event $B$" and $p(A)$ is the probability of occurrence of the event $A$.

Bayesian reasoning has been applied for tracking applications using a number of different approaches. Bayesian networks, which are extensions of the Bayes method, are in use for building long trajectories from smaller track pieces [34–37]. These Bayesian networks are used to assemble small tracks that have been generated from the video sequence in a prior processing step. Kang and Cho [38] combine edge and color information in a Bayesian network to recognize and track objects. We do not use a Bayesian network to solve the problem of tracking. Instead, we use the basic Bayesian formula to calculate probability of track-to-blob assignments.

More popular Bayesian tracking methods involve generating a reference model of the object being tracked, then looking for the best match for the model in each frame by using predictive approaches like Kalman filters [20] [23], and sampling-based approaches like Condensation [24] and the Joint Probabilistic Data Association Filter(JPDAF) [23] [26]. Joint Probabilistic Data Association Filter(JPDAF) is the most widely used method for probabilistic tracking of multiple objects at a time. Our method does not use a model of tracked objects to search for and segment from the image. Instead, we perform motion segmentation to detect objects for tracking. This leads to a fundamental difference in the way posterior probability is calculated for the JPDAF versus our method. In JPDAF, a model is maintained for each object being tracked in the video. For each frame, sampling and search techniques are used to arrive at candidate blobs (observations) for the active tacks. In JPDAF, the posterior probability of associating each observation to each track is calculated using the Bayesian formula on the track model

equations. In other words, JPDAF calculates the probability of blob-to-track association, given the track models and equations. We approach the assignment differently by calculating the probability of blob-to-track association, given the previous frame's blob information. Thus, we do not need to maintain track models for each track. We probabilistically assign tracks to blobs by using the color values and position information of the segmented blobs as observations in the Bayes formula. Isard and MacCormick [33] describe a Bayesian tracking system, Bramble, for tracking objects in real time. It works for indoor scenes with a non-dynamic background model and uses 3D modeling for objects. These tracking methods are Bayesian in the sense that they use particle filtering or sampling techniques based on the Bayes formula. We developed the use of Bayesian inference in a new way to solve the problem of simultaneous multiple target tracking in noisy outdoor scenes. Rather than using sampling methods, we use direct color and position observations from each moving blob to infer the best track labeling for the observed blobs.

## 8.3 Bayesian correspondence algorithm

In section 3.5.1, a Euclidean distance-based match matrix was used to solve the correspondence problem. In this section, we develop an alternative Bayesian inference method to find matching blobs in consecutive frames to facilitate track correspondence. We also calculate the probability that a track is 'lost' in the current frame. The basic idea behind our approach is that given a track in the previous frame, there is some probability that a blob of similar color and position will be observed in the current frame. Fig. 8.1 illustrates that a blob can be expected to be found in the region around the track's previous frame position.

Conversely, using the Bayes formula, given the current blob's color and position observations, we can find the probability of the current blob being associated with a track from the previous frame. This is illustrated in Fig. 8.2, where a blob $O_1$ is observed in the current frame.

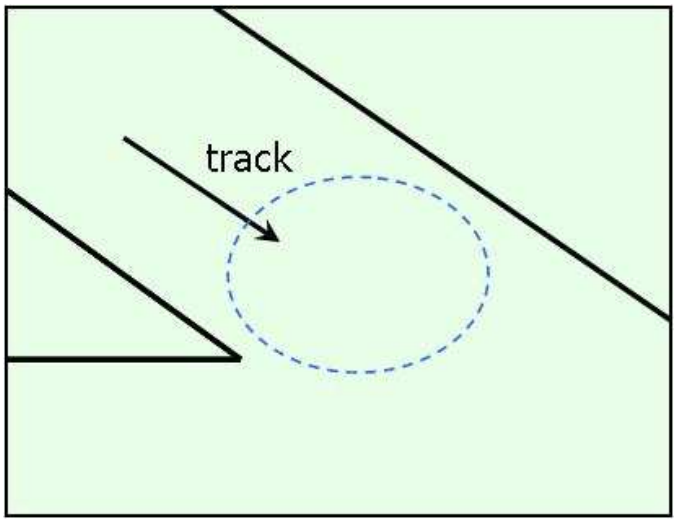



**Figure 8.1.** Illustration : A blob is probable in the region around a track's location in the previous frame

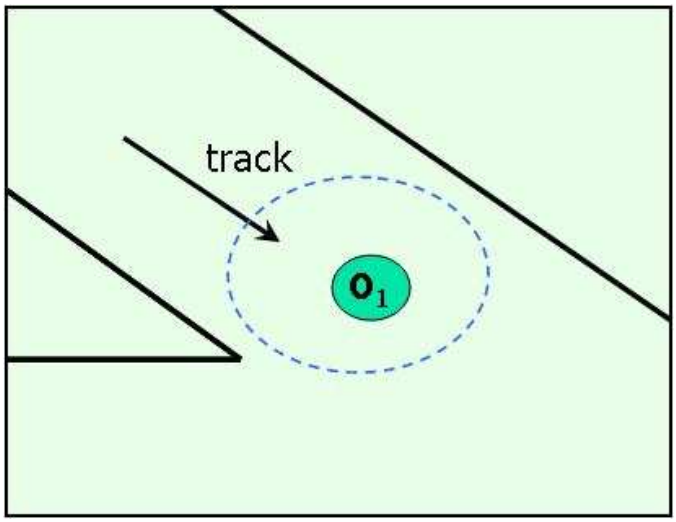



**Figure 8.2.** Illustration : If a blob is seen in current frame, Bayes' formula may be used to associate it with a track from previous frame

The sub-problem can be formulated thus:

If $T^{k-1} = \{t_1^{k-1}, t_2^{k-1}, t_3^{k-1}, \ldots, t_{u^{k-1}}^{k-1}\}$ is the set of tracks from the previous frame $(k-1)$, and $O^k = \{o_1^k, o_2^k, o_3^k, \ldots, o_{v^k}^k\}$ is the set of moving blobs in frame $k$, what is the probability of correspondence between each track and blob? As in the case of the match matrix, $t_j^{k-1}$ is the $j^{th}$ track from frame $(k-1)$, $u^{k-1}$ is the total number of tracks in frame $k-1$, $o_i^k$ is the $i^{th}$ blob found in frame $k$, and $v^k$ is the total number of moving blobs in frame $k$.

In other words, what is the probability of assignment of track $t_j^{k-1}$ to blob $o_i^k$ for all $j$ and $i$?

By looking at the observations for all elements $t_j^{k-1} \in T^{k-1}$ and $o_i^k \in O^k$ , we update a Belief Matrix which contains the probability of match of each track to each candidate blob. An increase in the probability of assignment of a track $t_j^{k-1}$ to a blob $o_i^k$ should automatically reduce the probability of assignment of the track $t_j^{k-1}$ to other blobs $o_m^k$ , where $m \neq i$. If no suitable match for a track $t_j^{k-1}$ is found, then the probability that the track $t_j^{k-1}$ is 'lost' should be made high. The probabilistic network that is used to calculate this Belief Matrix is given in Fig. 8.3.

### 8.3.1 Method

In the probabilistic network, $R$, $G$, and $B$ are the color observations. $R$ is the difference between mean red values, $G$ is the difference between mean green values, and $B$ is the difference between mean blue values of the blob $o_i^k$ and track $t_j^{k-1}$. Together $R$, $G$, and $B$ form the color observation $c$:

$$c = \{R = r, G = g, B = b\} \tag{8.2}$$

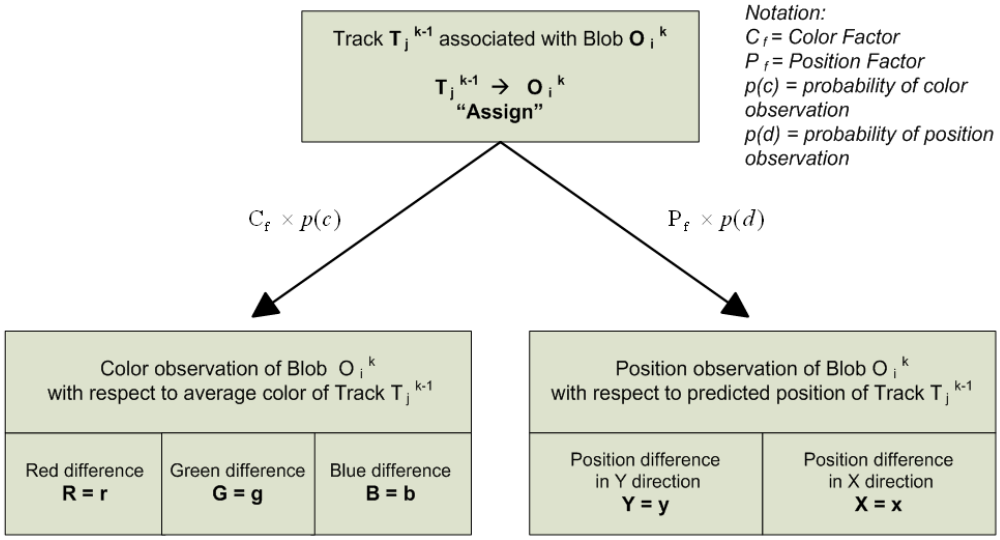



**Figure 8.3.** The Probabilistic Network used to calculate probability of track-to-blob association

$Y$ and $X$ are the differences between position of blob $o_i^k$ and predicted position of track $t_j^{k-1}$ in $y$ and $x$ coordinates, respectively. A simple estimate based on current velocity and current position of track is used as the predicted position for the track. Together, $Y$ and $X$ form the position observation $d$:

$$d = \{Y = y, X = x\} \tag{8.3}$$

Depending on the state of the color and position observations of blob $o_i^k$, the posterior probability of the blob $o_i^k$ belonging to a track $t_j^{k-1}$ is calculated. The probability functions for $p(c)$ and $p(d)$ are arrived at by manually observing about 200 frames in the video. By analyzing the color and position value difference between a track in current frame and its corresponding blob in the next frame for a few hundred frames, these PDF's were designed.

$C_f$ and $P_f$ are weight factors that may be used to weight the color and the

position probabilities. Currently, we weight the color and the position observations equally. However, they can be varied if either the color or the position observations need to be weighted differently.

Though R, G and B values are not independent in nature, for simplicity, we assume independence among $R$, $G$, and $B$ observations. The assumption of independence does not affect the results adversely. Thus,

$$p(c) = \{p(R = r) \times p(G = g) \times p(B = b)\} \tag{8.4}$$

Similarly, the Y and X independence assumption leads to

$$p(d) = \{p(Y = y) \times p(X = x)\} \tag{8.5}$$

For ease of notation, we now refer to the event "track $t_j^{k-1}$ associated with blob $o_i^k$" as '$Assign$'. The event that track $t_j^{k-1}$ is not associated with current candidate blob $o_i^k$ is called '$NotAssign$'. '$NotAssign$' implies that track $t_j^{k-1}$ has been associated with another blob $o_i^m$ (where $m \neq i$ ) or has been declared 'lost'.

From equations 8.4 and 8.5, we can say

$$p(c \mid Assign) = \{p(R = r \mid Assign) \times p(G = g \mid Assign) \times p(B = b \mid Assign)\} \tag{8.6}$$

$$p(d \mid Assign) = \{p(Y = y \mid Assign) \times p(X = x \mid Assign)\} \tag{8.7}$$

Also, we set

$$p(c \mid NotAssign) = 0.1 \tag{8.8}$$

$$p(d \mid NotAssign) = 0.1 \tag{8.9}$$

Equations 8.8 and 8.9 give the probabilities that a given blob $o_i^k$ observation is seen in the current frame without a corresponding track $t_j^{k-1}$ existing in the previous frame. It may be noted that these numbers are small, which reflects that the probability of finding a blob of given color and position without a corresponding track in the previous frame is low.

The prior probability for the assignment is:

$$p(t_j^{k-1} \rightarrow o_i^k) = p(Assign) = \frac{1}{v^k + 1} \tag{8.10}$$

where $v^k$ is the number of moving blobs in frame $k$.

Note: The factor $\frac{1}{v^k+1}$ implies that the track $t_j^{k-1}$ has an equal prior probability of being associated with any of the $v^k$ blobs of frame $k$ or of being declared as 'lost'.

$p(c)$ is given by:

$$p(c) = \{p(c \mid Assign) \times p(Assign) + p(c \mid NotAssign) \times p(NotAssign)\} \tag{8.11}$$

Similarly, $p(d)$ is given by:

$$p(d) = \{p(d \mid Assign) \times p(Assign) + p(d \mid NotAssign) \times p(NotAssign)\} \tag{8.12}$$

where $p(NotAssign) = 1 - p(Assign)$

By Bayes formula, the posterior probability of a track being assigned to a blob,

given the blob's color observation is,

$$p(Assign \mid c) = \frac{p(c \mid Assign) \times p(Assign)}{p(c)} \tag{8.13}$$

Similarly, the posterior probability of a track being assigned to a blob given the blob's position observation is

$$p(Assign \mid d) = \frac{p(d \mid Assign) \times p(Assign)}{p(d)} \tag{8.14}$$

### 8.3.2 Belief Matrix and its calculation

The Belief Matrix is a $u^{k-1}$ rows by $(v^k+1)$ columns matrix. The Belief Matrix contains the probabilities of each track being assigned to each blob and also of being declared as 'lost', hence the $(v^k + 1)$ columns. Denoted by $BP_{ji}$, Belief is the probability of assignment of track $j$ to blob $i$; i.e., $p(Assign\ j \rightarrow i)$. For each frame $k$, the following procedure is followed to update the Belief Matrix:

**Step** 1. Initialize the Belief Matrix to $\frac{1}{v^k+1}$

$$BP_{ji}(0) = \frac{1}{v^k + 1}, j \in \{1, 2, 3, \cdots u^{k-1}\}\ and\ i \in \{1, 2, 3, \cdots v^k - 1\} \tag{8.15}$$

where $u^{k-1}$ is the number of tracks in frame $(k-1)$, $v^k$ is the number of moving blobs in frame $k$ and $BP_{ji}(0)$ is the initial value of the Belief Matrix at row $j$ and column $i$.

**Step** 2. Iterate through all the tracks and blobs, and:

- For each track $t_j^{k-1} \in T^{k-1}$
  - For each blob $o_i^k \in O^k$

(a) Calculate $BP_{ji}(n)$ based on the color observation of $o_i^k$ where $BP_{ji}(n)$ stands for the value of the Belief Matrix at row $j$ and column $i$ due to observation from the $n^{th}$ blob (which is $o_i^k$). Note that the value of $n$ is always equal to the value of $i$.

(b) Normalize and update other Beliefs $BP_{jm}(n), m \neq i$

(c) Recalculate $BP_{ji}(n)$ based on the position observation of $o_i^k$

(d) Normalize and update other Beliefs $BP_{jm}(n), m \neq i$

**Step** 3. Match tracks and blobs based on the updated Belief Matrix

The Belief calculation based on the color observation is:

$$BP_{ji}(n) = \frac{p(c \mid Assign) \times BP_{ji}(n-1)}{p(c)} \tag{8.16}$$

where $BP_{ji}(n)$ is the Belief $BP_{ji}$ at the end of the observation from the $n^{th}$ blob (which is $o_i^k$). As noted before, the value of $n$ is always equal to value of $i$. Equation 8.16 is obtained by replacing $p(Assign \mid c)$ and $p(Assign)$ in the posterior equation 8.13 with $BP_{ji}(n)$ and $BP_{ji}(n-1)$ , respectively. This follows from the fact that $BP_{ji}(n)$ is the posterior probability *after* the color observation from $o_i^k$, and $BP_{ji}(n-1)$ is the prior probability *before* the color observation from $o_i^k$.

Similarly, the Belief calculation based on the position observation is:

$$BP_{ji}(n) = \frac{p(d \mid Assign) \times BP_{ji}(n-1)}{p(d)} \tag{8.17}$$

After each blob $o_i^k$ observation, it is necessary to normalize the Belief Matrix row so that $\sum_{i=1}^{v^k} BP_{ji}(n) = 1$. This is because each row reflects the probability of associating a track with the existing blobs or being 'lost'. The probabilities in each row should, thus, sum to 1.

We propose the following formula for normalization and update:
After the Belief of $j^{th}$ track is calculated due to observation from blob $o_i^k$,
For all $m \neq i$,

$$Belief_{jm\ new} = Belief_{jm\ old} \times \left[1 + \frac{\Delta p(NotAssign\ j \rightarrow i)}{p(NotAssign\ j \rightarrow i)_{old}}\right] \tag{8.18}$$

where $\Delta p(NotAssign\ j \rightarrow i)$ is the change in the Belief value for $j^{th}$ track and $i^{th}$ blob.
This formula ensures that all Beliefs are altered proportionally while maintaining the probability requirement $\sum_{i=1}^{v^k} BP_{ji}(n) = 1$.

Using the following formal definitions,
$Belief_{jm\ old} = BP_{jm\ old} = BP_{jm}(n-1)$
$Belief_{jm\ new} = BP_{jm\ new} = BP_{jm}(n)$
$p(Assignj \rightarrow i) = BP_{ji}(n)$
$p(NotAssignj \rightarrow i) = 1 - BP_{ji}(n)$
$p(NotAssignj \rightarrow i)_{old} = 1 - BP_{ji}(n-1)$
$\Delta p(NotAssignj \rightarrow i) = BP_{ji}(n-1) - BP_{ji}(n) = -\Delta BP_{ji}$
equation 8.18 becomes

$$BP_{jm\ new} = BP_{jm\ old} \times \left[1 - \frac{\Delta BP_{ji}}{1 - BP_{ji\ old}}\right] \tag{8.19}$$

### 8.3.3 PDF's used for R,G,and B; Y and X

By manually analyzing the difference in color and position values in object tracks over two hundred consecutive frames in the data, appropriate Probability Distribution Functions for the color and the position probability were determined. The PDF's are shown in figures 8.4 and 8.5. Fig. 8.4 is the PDF used for the probability of red color value ($p(R)$). The same PDF is used for green ($p(G)$) and blue ($p(B)$) color values. The PDF used for the $Y$ position observation ($p(Y)$) is given in Fig. 8.5. The same PDF is used for the $X$ position observation ($p(X)$) as well.

In our implementation, we chose $C_f$ and $P_f$ both to be 30 so that the probability function peaks of figures 8.4 and 8.5 become close to 1.0. Also, the color and the position observations are weighted equally because $C_f$ and $P_f$ are equal.

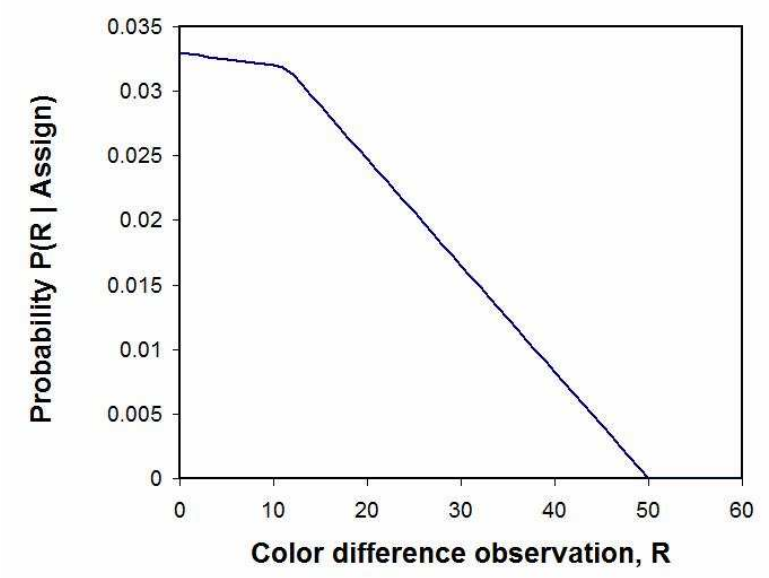



**Figure 8.4.** PDF for color observation,R

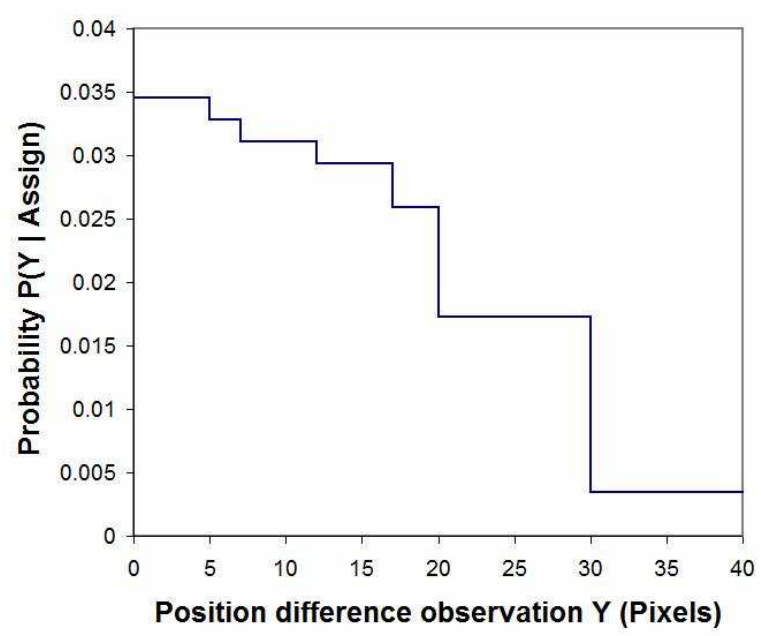



**Figure 8.5.** PDF for position observation,Y

### 8.3.4 Velocity factor in the Bayes algorithm

The Velocity factor is incorporated in the Bayesian tracking algorithm as well. The closer the velocity of a candidate blob is to a given track's velocity, the higher should be the probability of assigning the blob to that track. Thus, the Velocity factor function used in the Bayesian tracking algorithm is different from the function used in the non-Bayesian tracking algorithm earlier. The linearly decreasing function shown as an inset in Fig. 4.12 is used as the Velocity factor in the Bayesian algorithm. The linearly decreasing function weights smaller velocity differences more than larger velocity differences. Thus, the probabilities of the assignments that are more congruous in velocity are increased while the probabilities of incongruent assignments are decreased by use of the Velocity factor. The Velocity factor function used for the Bayesian tracking algorithm is plotted against the difference between blob and track velocities in Fig. 8.6.

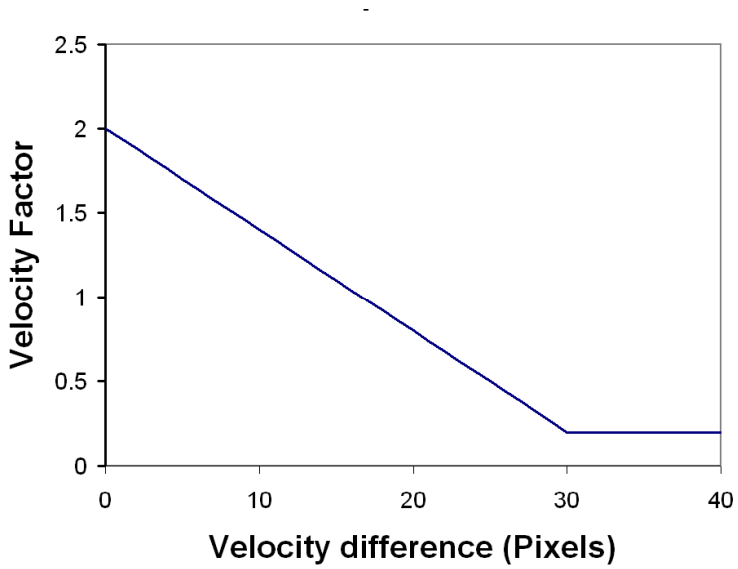



**Figure 8.6.** Velocity factor vs. observed difference in velocity – for the Bayesian algorithm

For each element in the Belief Matrix, the Belief value $BP_{ji}(n)$ is multiplied with the Velocity factor after the normalization that follows the Belief calculation due to the position observation from blob $o_i^k$. The normalization procedure is repeated after the multiplication with the Velocity factor. This ensures the probability requirement that the row sum is 1.

The tracking procedure after the inclusion of the Velocity factor is as follows:

**Step** 1. Initialize the Belief Matrix to $\frac{1}{v^k+1}$

$$BP_{ji}(0) = \frac{1}{v^k + 1}, j \in \{1, 2, 3, \cdots u^{k-1}\} \; and \; i \in \{1, 2, 3, \cdots v^k - 1\} \tag{8.20}$$

where $u^{k-1}$ is the number of tracks in frame $(k-1)$, $v^k$ is the number of moving blobs in frame $k$ and $BP_{ji}(0)$ is the initial value of the Belief Matrix at row $j$ and column $i$.

**Step** 2. Iterate through all the tracks and blobs and:

- For each track $t_j^{k-1} \in T^{k-1}$
  - For each blob $o_i^k \in O^k$
    (a) Calculate $BP_{ji}(n)$ based on the color observation of $o_i^k$ where $BP_{ji}(n)$ stands for the value of the Belief Matrix at row $j$ and column $i$ due to observation from the $n^{th}$ blob (which is $o_i^k$). Note that the value of $n$ is always equal to the value of $i$.
    (b) Normalize and update other Beliefs $BP_{jm}(n), m \neq i$
    (c) Recalculate $BP_{ji}(n)$ based on the position observation of $o_i^k$
    (d) Normalize and update other Beliefs $BP_{jm}(n), m \neq i$
    (e) Multiply $BP_{ji}(n)$ by the Velocity factor
    (f) Normalize and update other Beliefs $BP_{jm}(n), m \neq i$

**Step** 3. Match tracks and blobs based on the updated Belief Matrix

Fig. 8.7 shows the steps involved in calculating the first element in the Belief Matrix. The example illustrates a case of three tracks ($t_1$, $t_2$, and $t_3$) and two blobs ($O_1$ and $O_2$). The elements in the matrix that have changed in each step are highlighted. The Belief Matrix is initialized (Fig. 8.7(a)) so that each track has the same probability of being associated with any of the current blobs or of being declared as 'lost'. First, the color observation from blob $O_1$ is used to update the first element in the matrix (Fig. 8.7(b)). Normalization of other elements in the row is shown in Fig. 8.7(c). The first element in the matrix is then updated based on the position observation from blob $O_1$ (Fig. 8.7(d)). Next, the normalization process updates the other probabilities in the row (Fig. 8.7(e)). Multiplication with the Velocity factor results in an increase in the probability value (Fig. 8.7(f)). Finally, normalization results in the matrix shown in Fig. 8.7(g). The illustration shows the calculation of the first element in the Belief Matrix. Similarly, the other elements in the matrix are calculated.

After the Belief Matrix is calculated by using all blob observations available in the current frame, the best possible track assignments are made. An example of the Belief Matrix for frame 0240 is shown in Fig. 8.8. Fig. 8.8(b) shows the resulting track assignments for the frame. Blob 1 is assigned to track 12 and blob 2 to track 07. Tracks 3 and 12 are 'lost' in frame 0240.

| | Blob $O_1$ | Blob $O_2$ | "lost" |
|---|---|---|---|
| $t_1$ | 0.33 | 0.33 | 0.33 |
| $t_2$ | 0.33 | 0.33 | 0.33 |
| $t_3$ | 0.33 | 0.33 | 0.33 |

(a) Initial Belief Matrix

| | Blob $O_1$ | Blob $O_2$ | "lost" |
|---|---|---|---|
| $t_1$ | **0.50** | 0.33 | 0.33 |
| $t_2$ | 0.33 | 0.33 | 0.33 |
| $t_3$ | 0.33 | 0.33 | 0.33 |

(b) Color observation update

| | Blob $O_1$ | Blob $O_2$ | "lost" |
|---|---|---|---|
| $t_1$ | 0.50 | **0.25** | **0.25** |
| $t_2$ | 0.33 | 0.33 | 0.33 |
| $t_3$ | 0.33 | 0.33 | 0.33 |

(c) Normalization of row

| | Blob $O_1$ | Blob $O_2$ | "lost" |
|---|---|---|---|
| $t_1$ | **0.70** | 0.25 | 0.25 |
| $t_2$ | 0.33 | 0.33 | 0.33 |
| $t_3$ | 0.33 | 0.33 | 0.33 |

(d) Position observation update

| | Blob $O_1$ | Blob $O_2$ | "lost" |
|---|---|---|---|
| $t_1$ | 0.70 | **0.15** | **0.15** |
| $t_2$ | 0.33 | 0.33 | 0.33 |
| $t_3$ | 0.33 | 0.33 | 0.33 |

(e) Normalization of row

| | Blob $O_1$ | Blob $O_2$ | "lost" |
|---|---|---|---|
| $t_1$ | **0.86** | 0.15 | 0.15 |
| $t_2$ | 0.33 | 0.33 | 0.33 |
| $t_3$ | 0.33 | 0.33 | 0.33 |

(f) Velocity factor multiplication

| | Blob $O_1$ | Blob $O_2$ | "lost" |
|---|---|---|---|
| $t_1$ | 0.86 | **0.07** | **0.07** |
| $t_2$ | 0.33 | 0.33 | 0.33 |
| $t_3$ | 0.33 | 0.33 | 0.33 |

(g) Normalization of row

**Figure 8.7.** Illustration: Calculation of the Belief Matrix

| | Blob 1 | Blob 2 | "lost" |
|---|---|---|---|
| track 03 | 0.00 | 0.10 | **0.90** |
| track 07 | 0.00 | **1.00** | 0.00 |
| track 11 | 0.00 | 0.39 | **0.61** |
| track 12 | **1.00** | 0.00 | 0.00 |

**(a) Belief Matrix for frame 0240**


**(b) Resulting tracks for frame 0240**

**Figure 8.8.** Belief Matrix example - frame 0240

# Chapter 9

# Results from the Bayesian tracking algorithm

The concept of the Bayesian tracking algorithm and results from its use were presented in the IEEE Conference on Computer Vision and Pattern Recognition 2007 at the Workshop for Object Tracking and Classification in and Beyond the Visual Spectrum (OTCBVS). This chapter discusses the results that were submitted to the workshop as a paper [7].

## 9.1 Tracking results

Figures 9.1 and 9.2 show results of the Bayesian tracking algorithm. In Fig. 9.1 two cars are tracked with numbers 03 and 06. Trees in the foreground occlude the cars on numerous occasions. Sometimes the cars are occluded completely by the trees, as can be observed in Fig. 9.1(b) where car 03 has disappeared behind the tree. As subsequent frames show, the algorithm is capable of reassigning the correct track numbers when the cars reemerge. Results for a section of video

where cars are very close to each other are shown in Fig. 9.2. In this section of the video, four cars are successfully tracked with numbers 36, 37, 38, and 39. Tracks 39 and 36 temporarily disappear due to failure in the detection of the objects in frames 1625 and 1640 (Figs. 9.2(a) and (b)), respectively. When the segmentation algorithm detects the objects in subsequent frames, the tracks are successfully recovered.

In Fig. 9.3, the Belief Matrix from the Bayesian algorithm is compared to the Euclidean distance-based match matrix from the basic algorithm of section 3.5. The Bayesian Belief Matrix and the Euclidean distance matrix for frame 0275 are presented for analysis in the figure. Each value in the Euclidean distance matrix is the sum of the Euclidean distances in color (R, G, and B) and position (y and x coordinates) values between a blob of the current frame and a track from the previous frame. The smaller the distance value, the better the match between the blob and the track. Frame 0275 (Fig. 9.3) has three detected objects and three active tracks from previous frames. Frame 0265 (Fig. 9.3(a)) shows the earlier track placements. Car 12, which is detected and tracked in frame 0265 is not detected in frame 0275 due to shortcomings of the segmentation algorithm. Also, a new car enters the scene in frame 0275. In the Belief Matrix, blobs 1 and 2 are the blobs on the left side of frame 0275 and blob 3 is the blob on the right side of frame 0275. From the Bayesian Belief Matrix, we can infer that blobs 1 and 2 are best assigned to tracks 07 and 03, respectively. Track 12 is declared 'lost' and a new track 14 is generated for blob 3. The Euclidean distance matrix would assign blobs 1 and 2 correctly, but would incorrectly assign blob 3 to track 12. From the frames shown, we can see that this would be an error.

For every track, the Bayesian method gives the probability of the track being

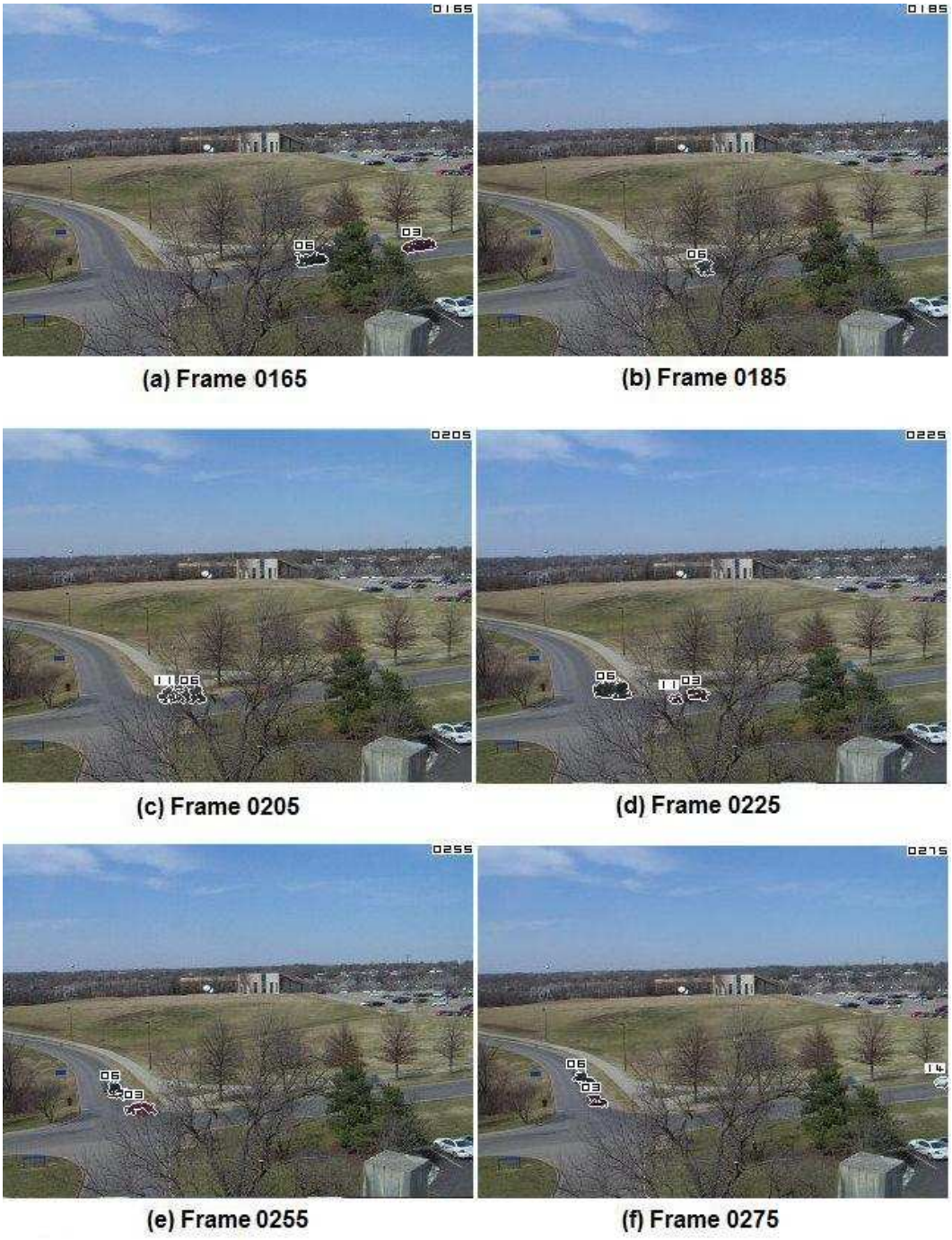



**Figure 9.1.** Bayes algorithm tracking results: Example 1 - Tracks 03 and 06 are accurately maintained through the set of frames

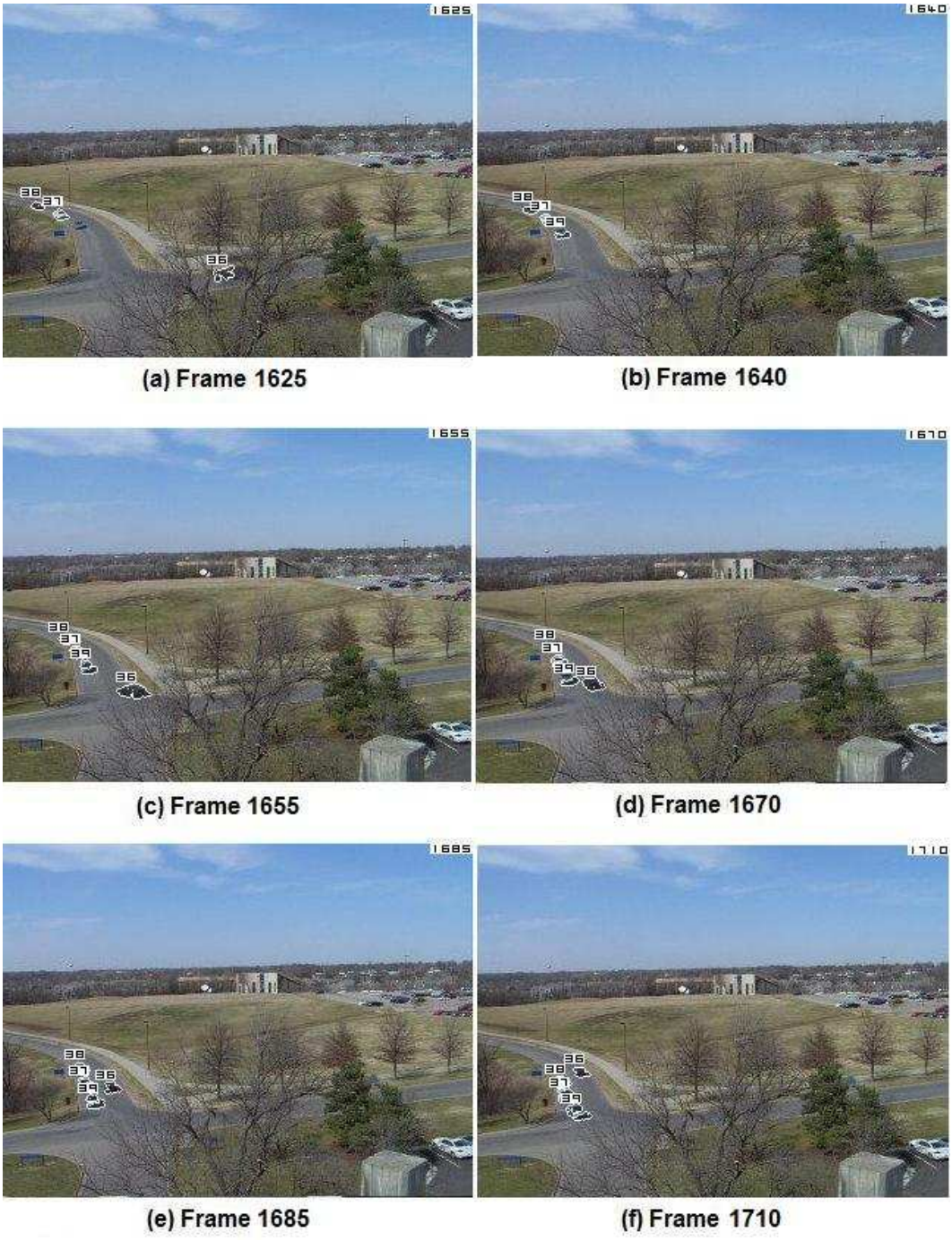



**Figure 9.2.** Bayes algorithm tracking results: Example 2 - Four tracks are accurately maintained through a difficult section of the sequence

(a) (b)

| | Blob 1 | Blob 2 | Blob 3 | "lost" |
|---|---|---|---|---|
| Track 3 | 0.35 | **0.55** | 0.00 | 0.10 |
| Track 7 | **0.38** | 0.52 | 0.00 | 0.10 |
| Track 12 | 0.00 | 0.00 | 0.49 | **0.51** |

(c) Bayes belief matrix - frame 0275

| | Blob 1 | Blob 2 | Blob 3 |
|---|---|---|---|
| Track 3 | 0.17 | **0.03** | 1.45 |
| Track 7 | **0.03** | 0.15 | 1.44 |
| Track 12 | 1.29 | 1.36 | **0.29** |

(d) Euclidean distance matrix - frame 0275

**Figure 9.3.** Comparison of Bayes Belief Matrix with Euclidean distance matrix for frame 0275

'lost'. It is not possible to calculate the chances of a track being 'lost' from the Euclidean distance matrix. Thus, the Belief Matrix can be more useful than the Euclidean distance matrix in higher-level inference tasks.

The Belief Matrix calculation, explained in section 8.3.2, captures the dependency between different track-blob assignments. If a particular blob is a very strong match for a track, then the matrix should reflect weak match values between other blobs and the track. The normalization process tends to reflect this dependency by proportionally scaling all the beliefs in a row. If a particular track-blob pair is a very strong match, then the Bayesian algorithm increases the corresponding probability in the Belief Matrix and the normalization step causes all

other probabilities in the row get scaled down. Similarly, if a particular track-blob pair is a weak match, then the assignment probability for this pair gets reduced and all the other probabilities in the row get scaled up. Since the distance-based match matrix treats each track-blob case independent of other track-blob cases, the dependence between assignments is not captured. Thus, we believe that the Belief Matrix can be a valuable alternative to Euclidean distance-based match matrices. The Belief Matrix, by reflecting dependency of track-blob assignments more accurately and by giving the probability of a track being 'lost', can give better results if used in a higher-level intelligence application.

## 9.2 Tracking results from other video sequences

The tracking algorithm has been tested on other video sequences in the database and good tracking results are achieved in the other sequences as well. Sample frames from the results of tracking in a parking lot scene (video sequence 2) are shown in figures 9.4 and 9.5. In Fig. 9.4, a single car is seen entering the parking lot and is tracked with the number 01. The algorithm is also sensitive enough to detect and track a human target (71) in the sequence, as Fig. 9.5 shows.

Figures 9.6 and 9.7 are results from video sequence 3, where the camera overlooks a straight section of a road. Two cars and a human being are tracked in Fig. 9.6 with numbers 06, 22, and 21, respectively. In frame 0720 (Fig. 9.6(d)), the two cars are merged because their pixels are connected. Track 22 is 'lost' in this frame. In subsequent frames, the track numbers are recovered correctly. In Fig. 9.7, two cars are tracked with numbers 59 and 63. Video sequence 3 is another example of a video where there is considerable variation in the distances between the camera and the objects in different parts of the scene. The use of the Vicinity

Factor brings about significant improvements in the segmentation and tracking results in this video as well.

## 9.3 System performance

The Bayesian tracking algorithm takes about 100 seconds of CPU time to process a video sequence of 1843 frames of size 320 by 240 pixels. This includes the time required for reading input jpeg frames and writing output jpeg frames on a Pentium dual processor 3 GHz machine. This is a throughput of approximately 18 frames per second (fps); thus, the system is capable of working with 15 fps surveillance video in real time.

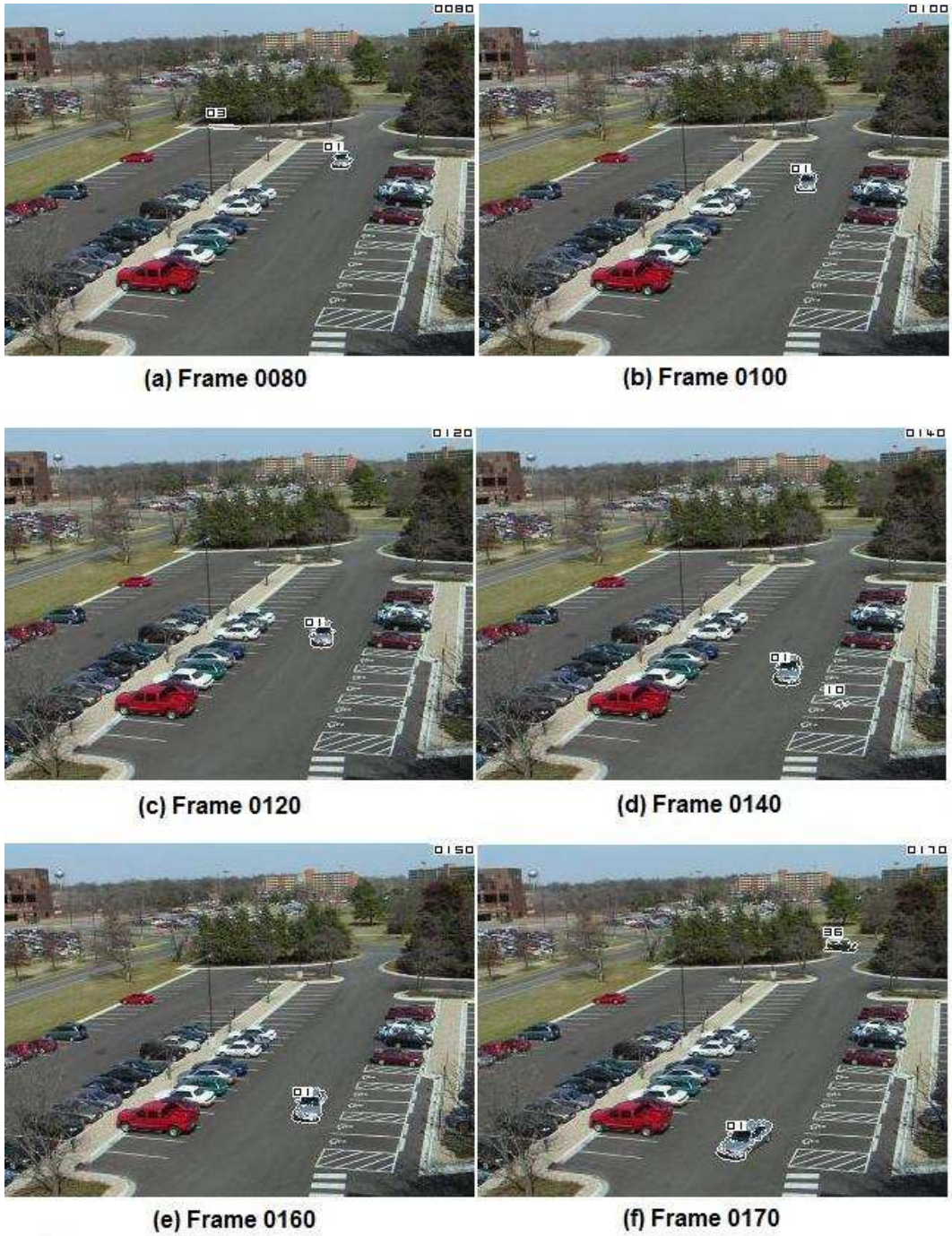



**Figure 9.4.** More tracking results: Video sequence 2 - Tracks 06, 21, and 22 through a set of frames

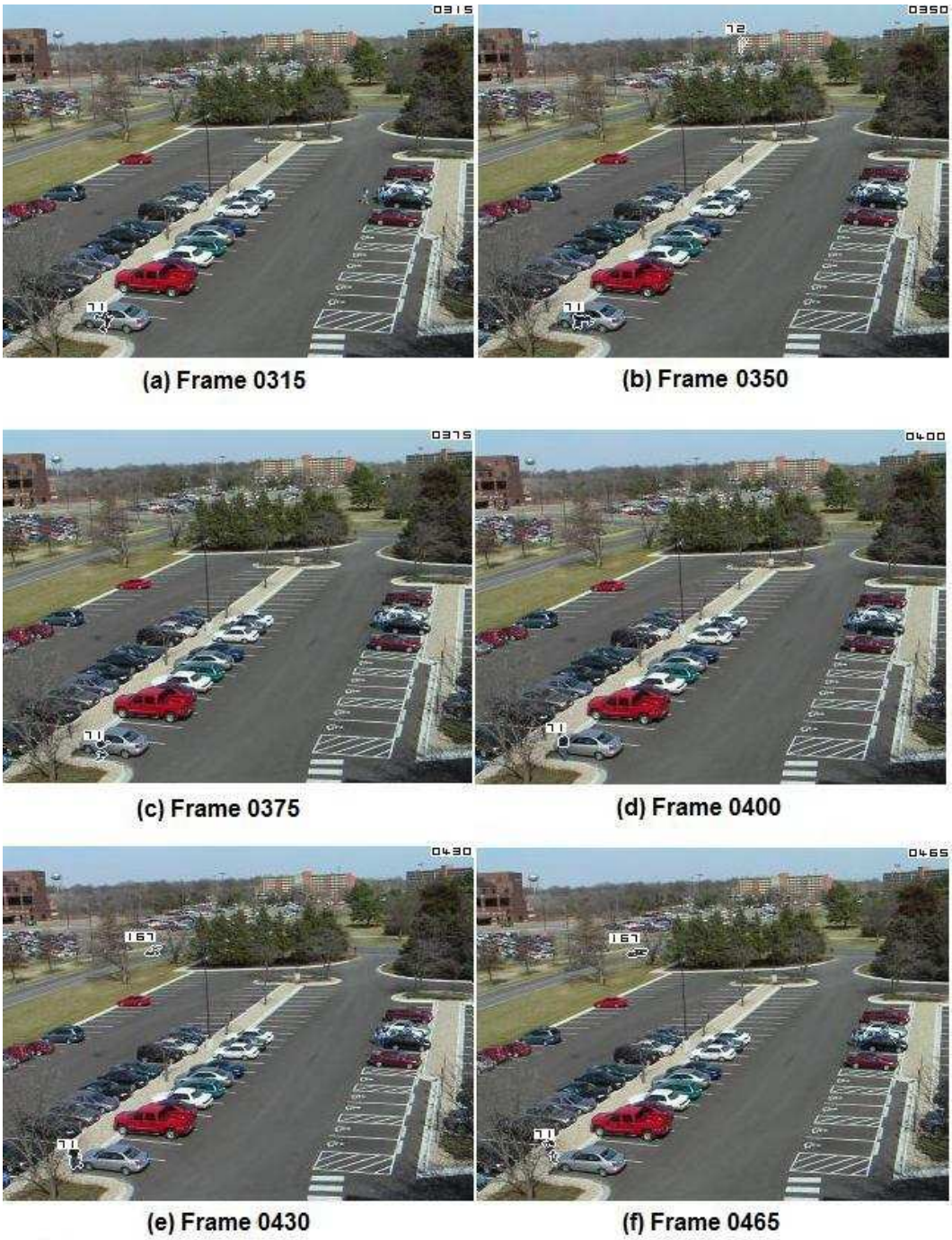



**Figure 9.5.** More tracking results: Video sequence 2 - Human target 71

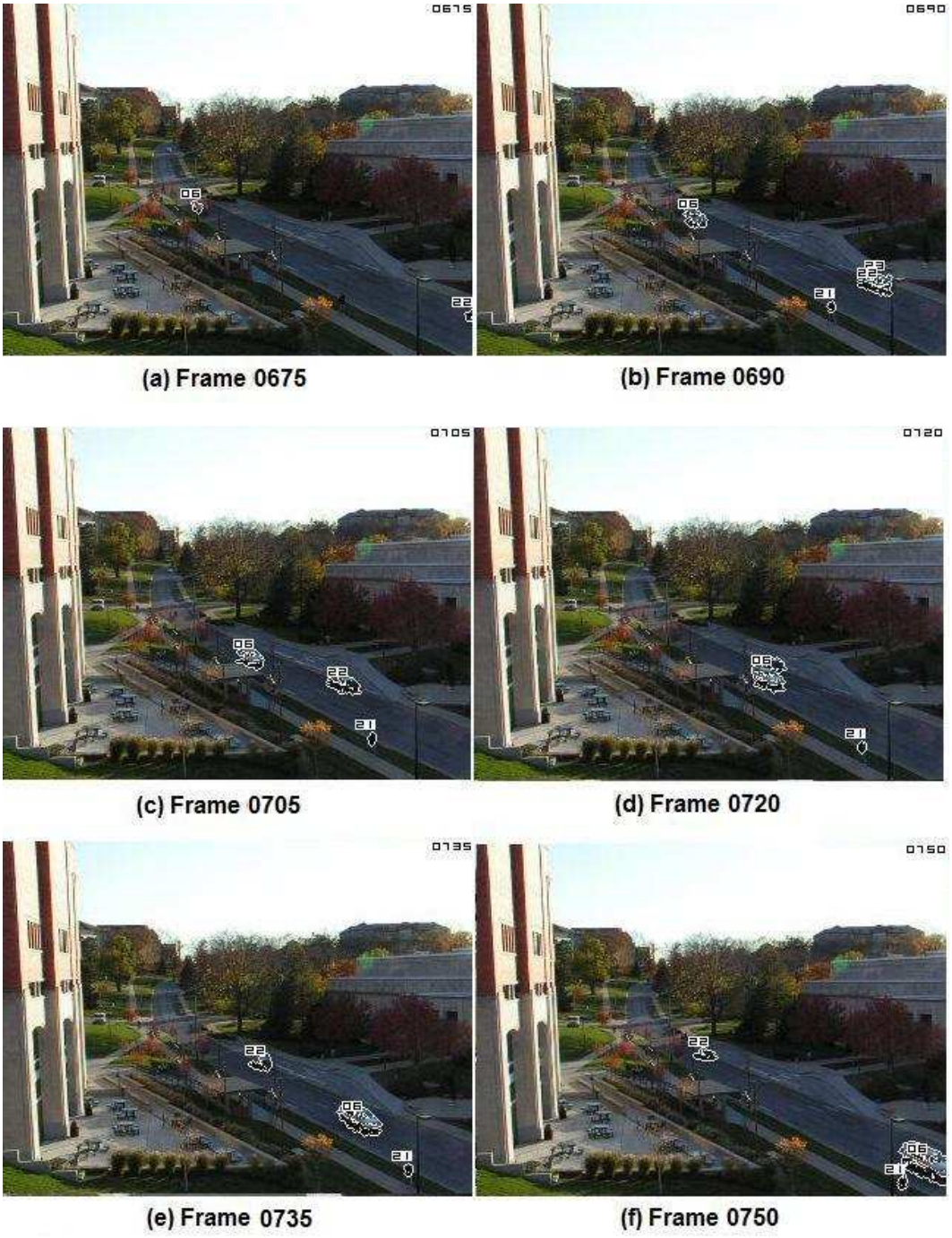



**Figure 9.6.** More tracking results: Video sequence 3 - Tracks 06, 22, and 21 through a set of frames

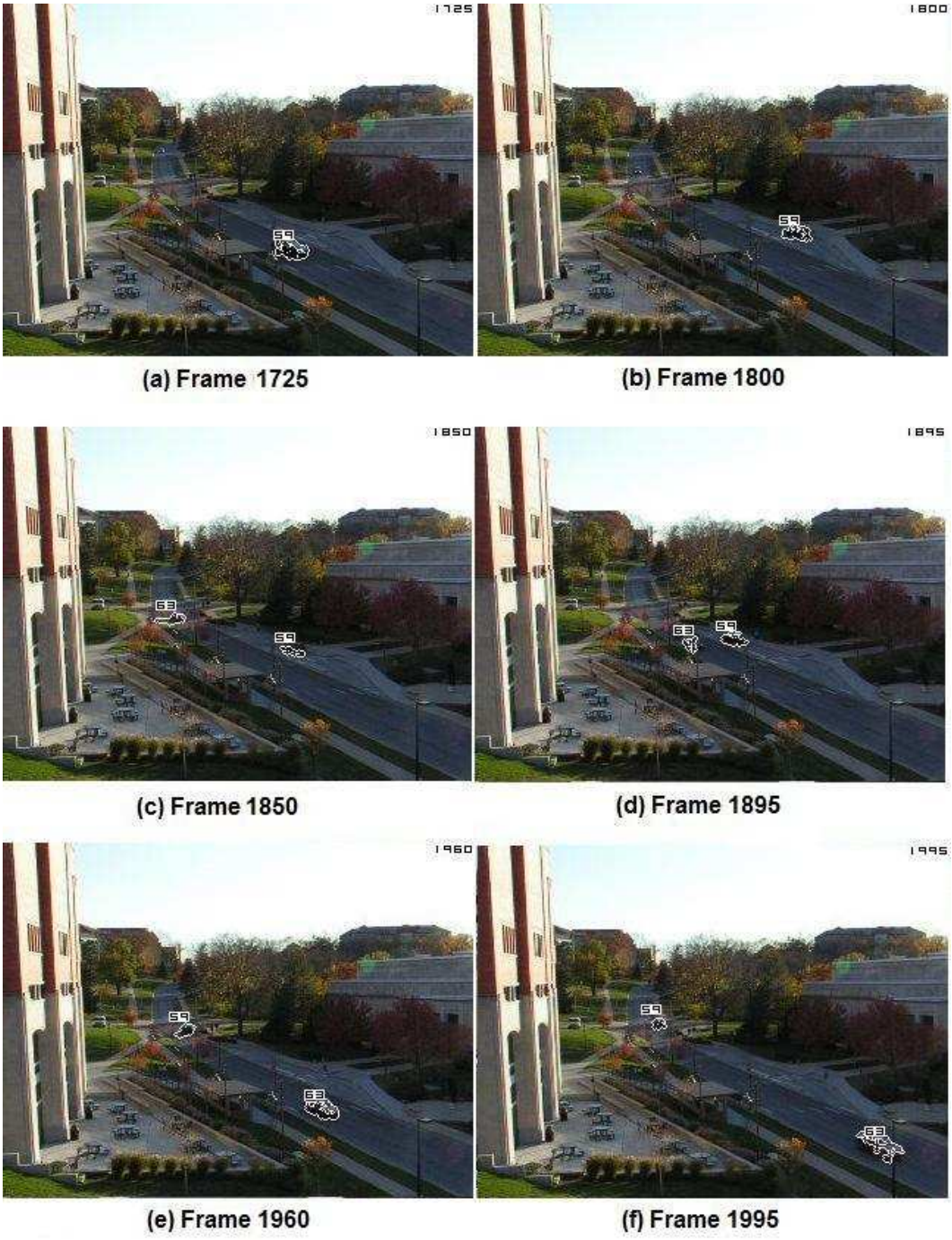



**Figure 9.7.** More tracking results: Video sequence 3 - Tracks 59 and 63 through a set of frames

# Chapter 10

# Conclusions

Automatic tracking is a very interesting research area that can lead to numerous intelligent applications. Our work focused on setting up a system that may be used for intelligent applications in video surveillance of outdoor traffic scenarios. Outdoor videos are more challenging than indoor videos because of illumination changes, non-static backgrounds, and occlusion. The high amount of noise and uncertainty observed in outdoor sequences makes tracking in these sequences a difficult problem to solve.

We have designed a tracking system that is able to detect and track moving objects in outdoor video. After setting up a basic system, we were able to bring significant improvements in the tracking by use of new algorithms. Thus, we were successful in both our goals of establishing a base system that can serve as a platform for future Automatic Video Surveillance research at KU and of developing new algorithms to further the state of research in the field.

We have implemented the simple and efficient method of moving average background model for segmenting the moving objects in a video sequence. The moving average background method comes with the disadvantage that significant tails are

segmented along with the moving object. We overcame this problem by designing a Secondary Background model that uses a learning approach to remove the object tails. Our segmentation algorithm is both efficient and effective.

The object tracking algorithm based on the Euclidean distance between the tracks and blobs resulted in good tracking results. Suitable class structures were designed to effectively represent the blobs and the tracks in the video for the purpose of tracking. We found that due to the nature of the video sequences being used, the basic algorithm had a few shortcomings. These were addressed by implementing a merge module and using the velocity vector of objects as an important factor in the tracking algorithm. Depending on the threshold values used, the merge module resulted in a 12% to 17% improvement in the segmentation results. Use of the Velocity factor improved the tracking results by over 44%.

The Vicinity Factor is a useful innovation that can be used in tracking applications where the camera is static. It is learned automatically by the system and is able to reflect real world 3-D distances around tracked objects without the need for complicated 3-D modeling procedures. The efficacy of the Vicinity Factor is exhibited by the improvement brought about by its use in the merge module. The use of the Vicinity Factor solved the problem of dependence of the segmentation results on the threshold values that are used in the merge module. The merge module performance was further improved by 6% by the use of the Vicinity Factor. The Vicinity Factor also offers other advantages and can be exploited further in tracking applications.

The Bayesian tracking algorithm adds the capability of probabilistic track assignment to our tracking system. The Belief Matrix that shows the probability of assignment of each track to each blob is a better numerical representation of the

relationships between the tracks and the blobs in the scene. The Belief Matrix could be used as an alternative to distance-based match matrices. Thus, the probabilistic tracking algorithm can be very useful when higher-level intelligence and inference applications are built on the system.

Thus, we have established a system for automatic tracking of objects by extending the existing KUIM image processing code at the University of Kansas. The new tracking and segmentation functionalities are substantial additions to the KUIM repository. The tracking system can serve as a platform for future research in intelligent video applications.

# Chapter 11

# Future work

We see two directions for future expansion of work in this thesis - (a) Improvement in the current system's segmentation and tracking algorithms, and (b) Extension of the system towards building higher-level intelligence applications based on motion tracks. In terms of improving the current system, there are many opportunities for future work in the object segmentation, the object tracking, the Vicinity Factor, and the Bayesian tracking algorithm sections.

## 11.1 Object segmentation

There is scope for improvement in the basic segmentation algorithm of this thesis. We use a moving average background model which is simple and very efficient. In the future, a Gaussian background model may be implemented for arriving at the background model for each pixel.

Motion segmentation used in this thesis is an easy method to detect objects of interest. However, motion segmentation methods, due to their very nature, have drawbacks. They fail when the tracked object stops moving. Even slight motion

in the camera can cause large number of false object detections. The moving average background model used in this thesis is able to recover from the effects of camera motion after a few hundred frames, but an ideal segmentation algorithm should not be affected by camera motion. Template matching is the best way to overcome the drawbacks in motion segmentation methods. In template matching, a template is created for the object being tracked and in subsequent frames, the template is searched for by using various search algorithms or sampling methods. Template-based segmentation would enable our system to be used in applications where the camera is not static.

## 11.2 Object tracking

Currently, the size of blobs is not used in calculating distances between the active tracks and the detected blobs. In the future, blob size can be used along with the blob's color and position values for determining distances in the match matrix.

After first generating tracks using the tracking algorithm, a secondary analysis of the tracks can be done to remove extremely small tracks which are likely to be noise. Such analysis can also help in joining together tracks that are broken due to lack of continuity.

Our approach to tracking was to find correspondences between objects that have already been segmented using motion segmentation. The method is very efficient for a static camera surveillance application. If a future application demands use of a non-static camera, a template-based segmentation algorithm may be used in conjunction with tracking methods like Kalman tracking, Condensation, and JPDAF.

## 11.3 The Vicinity Factor

The Vicinity Factor introduced in this thesis resulted in improved tracking results. As explained in section 7.3, it has a few drawbacks and it may be improved for future applications. The Vicinity Factor may be useful in identifying anomalous object motion behavior in the video by analyzing the object's motion with respect to the Vicinity Factor values in that region of the scene.

For tracking algorithms where multiple classes of objects are to be tracked (humans, cars, bikes, etc.), an improved Vicinity Factor model maintaining separate Vicinity Factor matrices for each class of objects could be designed.

As explained in chapter 7, the Vicinity Factor can be useful in calculating object size in different parts of the scene. It can also be used as a robust method to detect 'active' regions of the scene.

## 11.4 Bayesian tracking algorithm

Currently, the only human input that the Bayes algorithm needs is the PDF's for the functions that are used for $p(c)$ and $p(d)$. These PDF's are based on the statistics from the video and are obtained by manual analysis of the video. These statistics can easily be learned by the algorithm itself. An improved Bayesian algorithm in which the PDF's for the color and position probabilities are learned automatically would be able to run without any human inputs.

Along with the color and position, other observations like size can be incorporated in the Bayes method. Also, our system assumes independence between R, G, and B color values of the objects. Despite this inaccurate assumption, the Bayesian algorithm works very well. In the future, other color spaces like Hue Sat-

uration Value (HSV) may be experimented with to see if further improvements could be achieved.

## 11.5 Extensions of the current system

The current system can be extended to build systems intelligent analysis from video sequences for various applications. Based on motion tracks of objects, we can learn patterns of activity in the video. The learned patterns can then be used to detect, classify, and analyze future activities in the area under surveillance. Detection of events, identification of humans based on gait analysis, and inference of behavior between objects are some of the applications that can be researched in the future by extending the work in this thesis.

There is considerable academic and commercial interest in the field of intelligent analysis from video. With different applications of personal, corporate, military, and commercial nature, the possibilities for technological advances based on intelligent video analysis are immense. Our work has opened the door for such research at the University of Kansas.

# References


[1] Sangho Park and J. K. Aggarwal. Recognition of two-person interactions using a Hierarchical Bayesian Network. In *IWVS '03: First ACM SIGMM International Workshop on Video Surveillance*, pages 65–76, New York, NY, USA, 2003. ACM Press.

[2] Ju Han and Bir Bhanu. Individual recognition using gait energy image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(2):316–322, 2006.

[3] Sudeep Sarkar, Jonathon Phillips, Zongyi Liu, Isidro Robledo Vega Patrick Grother, and Fellow-Kevin W. Bowyer. The HumanID gait challenge problem: Data sets, performance, and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2):162–177, 2005.

[4] M. Petkovic, Z. Zivkovic, and W. Jonker. Recognizing strokes in tennis videos using Hidden Markov Models. In *Proceedings of the IASTED International Conference Visualization, Imaging and Image Processing, Marbella, Spain*, 2001.

[5] Nils T Siebel and Steve Maybank. Real-time tracking of pedestrians and vehicles. In *Proceedings of the 2nd IEEE International Workshop on Perfor-*

*mance Evaluation of Tracking and Surveillance (PETS´2001), Kauai, USA*, December 2001. CD-ROM proceedings.

[6] Deva Ramanan, David A. Forsyth, and Kobus Barnard. Building models of animals from video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1319–1334, 2006.

[7] Manjunath Narayana and Donna Haverkamp. A Bayesian algorithm for tracking multiple moving objects in outdoor surveillance video. In *Fourth Joint IEEE International Workshop on Object Tracking and Classification in and Beyond the Visual Spectrum, In conjunction with IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2007*, June 2007.

[8] Weiming Hu, Tieniu Tan, Liang Wang, and S. Maybank. A survey on Visual Surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 34(3):334–352, 2004.

[9] Guohui Li, Jun Zhang, Hongwen Lin, D. Tu, and Maojun Zhang. A moving object detection approach using Integrated Background template for smart video sensor. In *Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference*, volume 1, pages 462–466, May 2004.

[10] Tao Zhao and Ram Nevatia. Tracking multiple humans in crowded environment. *CVPR*, 02:406–413, 2004.

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

[12] Yaser Sheikh and Mubarak Shah. Bayesian modeling of dynamic scenes for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(11):1778–1792, 2005.

[13] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt. Performance of optical flow techniques. *CVPR*, 92:236–242.

[14] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, 1981.

[15] J. Wang and E. Adelson. Spatio-temporal segmentation of video data. In SPIE Proc. Image and Video Processing II, 1994.

[16] L. Biancardini, E. Dokadalova, S. Beucher, and L. Letellier. From moving edges to moving regions. In *Proceedings Second Iberian Conference IbPRIA 2005*, volume 1, pages 119–127, 2005.

[17] Gerald Kuhne, Stephan Richter, and Markus Beier. Motion-based segmentation and Contour-based classification of video objects. In *MULTIMEDIA '01: Proceedings of the Ninth ACM International Conference on Multimedia*, pages 41–50, New York, NY, USA, 2001. ACM Press.

[18] S. Intille, J. Davis, and A. Bobick. Real-time closed-world tracking. *IEEE CVPR*, pages 697–703, 1997.

[19] P. Kumar, S. Ranganath, W.M. Huang, and K. Sengupta. Framework for real-time behavior interpretation from traffic video. *ITS*, 6(1):43–53, March 2005.

[20] Rudolph Emil Kalman. A new approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[21] Dieter Koller, Joseph Weber, and Jitendra Malik. Robust multiple car tracking with Occlusion Reasoning. In *ECCV (1)*, pages 189–196, 1994.

[22] Jianguang Lou, Tieniu Tan, and Weining Hu. Visual vehicle tracking algorithm. *IEE Electronics Letters*, 38(18):1024–1025, August 2002.

[23] Y. Bar-Shalom. *Tracking and Data Association*. Academic Press Professional, Inc., San Diego, CA, USA, 1987.

[24] Michael Isard and Andrew Blake. Condensation – Conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[25] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing Survey*, 38(4):13, 2006.

[26] Christopher Rasmussen and Gregory D. Hager. Probabilistic Data Association methods for tracking complex visual objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):560–576, 2001.

[27] Reinhard Koch. 3D-Scene modeling from image sequences. *ISPRS Archives*, 34, 2003.

[28] Reihard Koch, Marc Pollefrys, and Luc Van Gool. Realistic 3-D scene modeling from uncalibrated image sequences. In *Proceedings of the International Conference on Image Processing, ICIP 99*, pages 500–504, 1999.

[29] Karsten Muller, Aljoscha Smolic, Michale Drose, Patrick Voigt, and Thomas Weigand. 3-D reconstruction of a dynamic environment with a fully calibrated background for traffic scenes. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(4):538–549, April 2005.

[30] Ali Azarbayejani and Alex P. Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):562–575, 1995.

[31] Robert Collins, Alan Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, and Osamu Hasegawa. A system for video surveillance and monitoring. Technical Report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2000.

[32] I. Haritaoglu, D. Harwood, and L. Davis. Who, when, where, what: A real time system for detecting and tracking people. In *Proceedings of the Third Face and Gesture Recognition Conference*, pages 222–227, 1998.

[33] Michael Isard and John MacCormick. BraMBLe: A bayesian Multiple-Blob tracker. In *Proceedings of the Eighth International Conference on Computer Vision*, volume 2, pages 34–41, 2001.

[34] Pedro Jorge, Arnaldo J. Abrantes, and Jorge Salvador Marques. On-line object tracking with Bayesian Networks.

[35] A. J. Abrantes, J. S. Marques, and J. M. Lemos. Long term tracking using Bayesian Networks. In *Proceedings of the International Conference on Image Processing, ICIP 02*, volume 3, June 2002.

[36] Jorge S. Marques, Pedro M. Jorge, Arnaldo J. Abrantes, and J. M. Lemos. Tracking groups of pedestrians in video sequences. *2003 Conference on Computer Vision and Pattern Recognition Workshop, CVPRW*, 09:101, 2003.

[37] Peter Nillius, Josephine Sullivan, and Stefan Carlsson. Multi-target tracking - linking identities using Bayesian Network inference. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2187–2194, Washington, DC, USA, 2006. IEEE Computer Society.

[38] Hang-Bong Kang and Sang-Hyun Cho. Adaptive object tracking using Bayesian Network and memory. In *VSSN '04: Proceedings of the ACM 2nd International Workshop on Video Surveillance & Sensor Networks*, pages 104–113, New York, NY, USA, 2004. ACM Press.