NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPTICAL FLOW ANALYSIS AND KALMAN FILTER TRACKING IN VIDEO SURVEILLANCE ALGORITHMS

by

David A. Semko

June 2007

Thesis Advisor:	Monique P. Fargues
Second Reader:	Roberto Cristi

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2007	Master's Thesis

4. TITLE AND SUBTITLE Optical Flow Analysis and Kalman Filter Tracking in Video Surveillance Algorithms	5. FUNDING NUMBERS
6. AUTHOR(S) Semko, David A.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

13. ABSTRACT (maximum 200 words)

A SIMULINK-based algorithm for monitoring contacts in a surveillance video sequence using Optical Flow Analysis and Kalman Filters was developed. The Horn-Schunk Optical Flow Algorithm was used to identify contacts in a surveillance video sequence. The position and behavior of these contacts was monitored by a modification of the traditional Kalman Filter. The Kalman Filter algorithm implemented has the ability to track up to ten contacts at a time, correctly assigning each of a maximum ten filters to their respective contacts on a frame-by-frame basis. Initial tests using artificial data show good performance of both the Optical Flow Analysis algorithm and the Kalman Filter Tracking algorithm. Surveillance video data was also used to test the algorithm with promising results.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Optical Flow, Kalman Filter, Horn-Schunk, Surveillance, Video Processing, Contact Tracking, Morphological Operations	89
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

OPTICAL FLOW ANALYSIS AND KALMAN FILTER TRACKING IN VIDEO SURVEILLANCE ALGORITHMS

David A. Semko
Ensign, United States Navy
B.S.E., Duke University, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2007

Author: David A. Semko

Approved by: Monique P. Fargues
Thesis Advisor

Roberto Cristi
Second Reader

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A SIMULINK-based algorithm for monitoring contacts in a surveillance video sequence using Optical Flow Analysis and Kalman Filters was developed. The Horn-Schunk Optical Flow Algorithm was used to identify contacts in a surveillance video sequence. The position and behavior of these contacts was monitored by a modification of the traditional Kalman Filter. The Kalman Filter algorithm implemented has the ability to track up to ten contacts at a time, correctly assigning each of a maximum ten filters to their respective contacts on a frame-by-frame basis. Initial tests using artificial data show good performance of both the Optical Flow Analysis algorithm and the Kalman Filter Tracking algorithm. Surveillance video data was also used to test the algorithm with promising results.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Professor Monique Fargues for sparking my interest in the interesting topic of surveillance video processing. I am indebted to her for her countless hours of support in developing the algorithm presented and editing this thesis. I would also like to thank Professor Roberto Cristi for his help throughout the completion of this project. His advice and explanations were invaluable in my coming to understand the topics of Optical Flow Algorithms and Kalman Filtering.

On a more personal note I would like to thank all of the other Ensigns, especially *The Crew*, who have made my time in Monterey so enjoyable. Special thanks go to Kyle and Dan, my EE support group. I'd also like to thank my girlfriend Bridgette for continually encouraging me when I became frustrated with my progress in this thesis. Finally, I'd like to thank my parents who have loved me and supported me in all of my endeavors and have helped me get to where I am today.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis is an introductory work examining the feasibility of using Optical Flow algorithms in conjunction with Kalman Filters for tracking of contacts in a video surveillance scene. The overall goal of the research conducted in automated video surveillance is to develop algorithms with the ability to automatically process surveillance video feeds and simplify their analysis for the user.

In order to create an algorithm that can not only track contacts in a scene, but also analyze their behavior, three different, large-scale tasks have to be accomplished. First the algorithm has to be able to take an incoming surveillance video feed and segment it into a stream of frames where contacts are distinguished from the background. Next these contacts must be tracked throughout the video sequence. Finally, these tracks must be processed to analyze their behavior.

In order to segment an incoming feed of surveillance video, the algorithm developed in this thesis uses the Horn-Schunk Optical Flow algorithm. This algorithm approximates the movement of objects in a current frame as referenced to some previous frame. By determining the motion of objects, one can distinguish between contacts and the background. After careful tuning and processing, the output of the Optical Flow algorithm consists of two main parts: the number of contacts present in the frame, and the location of the centroid of each of these contacts. This data is passed to the Kalman Filter Tracking algorithm for further processing.

The Kalman Filter is a type of recursive, adaptive filter that operates in the state space. It is well-known for its ability to track objects in a timely and accurate manner. The tracking algorithm developed in this thesis can process up to ten contacts at a time where the number of filters activated for each frame of data is dependent on the number of contacts deemed present in the scene. The

Kalman Filter Tracking algorithm developed in this thesis is robust in its ability to handle non-ideal data from the Optical Flow Analysis algorithm. It can maintain active tracks on a contact even when the contact is temporary lost by the Optical Flow Analysis algorithm. Its design allows it to correctly assign each filter to its corresponding contact to ensure the most accurate operation.

Finally, a simple, “Lost Contact Notification” algorithm is developed to automatically notify the user when a contact in the scene has been lost. This loss could be related to a variety of factors, including system malfunction, contact leaving the scene, contact behind an occlusion, contact loitering in the scene. The visual notification created by the algorithm alerts the user to the last known location of the contact and it is left to the user to decide the cause of the contact loss.

Results are obtained using both artificial and realistic data. With artificial data, the Kalman Filter Tracking algorithm and the Lost Contact Notification algorithm operate very well. The system is able to track simultaneous contacts under varying conditions and identify when a contact has been lost in the scene. The realistic surveillance video used to test the system also yields promising results. While tracking of contacts is not as consistent with realistic data, the algorithm performs satisfactorily. While the algorithm does encounter some difficulty in maintaining consistent performance, this is attributed to the inconsistency at the output of the Optical Flow algorithm. It is left for further research to assess the effectiveness of the Optical Flow algorithm in an objective manner.

Overall, the algorithm developed shows much promise as a tool to aid in automated surveillance video processing. Further research should be done in the area of segmentation algorithms to determine an optimum algorithm to segment the data into contacts and background. In addition, further development of algorithms that analyze the behavior of contacts in the scene could greatly increase the value of the system developed.

I. INTRODUCTION

A. VIDEO SURVEILLANCE

Interest in video surveillance techniques has grown significantly in the years following the September 11th attacks [1]. This interest has led to a surge of Closed Circuit Television (CCTV) cameras being installed in public areas for surveillance—a departure from the traditional use by private companies for property protection. In order for such systems to be a reliable tool for use by law enforcement officials and other interested individuals or organizations, the system must be able to accurately detect object behaviors and identify those behaviors of *significance*. As the number of cameras per system grows, the task of identifying events of significance becomes increasingly difficult for human operators and necessitates the aid of computer algorithms [2].

Hu et al. [2] present a survey of the different applications which generally necessitate the use of video surveillance algorithms. In this thesis we will investigate the area of *anomaly detection and alarming*. This application generally seeks to track people (and/or vehicles) moving through a scene, to classify the behaviors of each track, and to identify whether these behaviors can be considered normal or abnormal (significant).

The data used in this study was obtained from the European Community Funded CAVIAR project/IST 2001 37540 [3]. This data was taken from a wide-angled camera in the lobby of the INRIA Laboratories in Grenoble, France. Much of the video contains various pedestrians either moving through the lobby or loitering in certain areas of the lobby. This study attempts to track both types of contacts throughout the scene. A significant event and an alarm is generated to instruct the operator to investigate the area in which the contact was last known to be located when the algorithm is no longer able to obtain a track of said objects.

B. OBJECTIVE

The overall goal of the study is to create a system which pre-screens video surveillance feed and assists the user in identifying unusual activity. Towards that goal, we designed an automated scheme capable of performing three large-scale tasks: identifying contacts, tracking contacts, and characterizing contact behavior.

C OVERVIEW

This thesis is organized as follows. Chapter II provides background information on both Optical Flow Analysis and the Kalman Filter. The next three chapters investigate each of the main processing blocks shown in Figure 1. Chapter III describes the implementation of an Optical Flow Algorithm. Chapter IV describes the Kalman Filter Implementation used for tracking. Chapter V describes the post-processing performed and presents the results obtained from post-processing. Conclusions and recommendations are presented in Chapter VI.

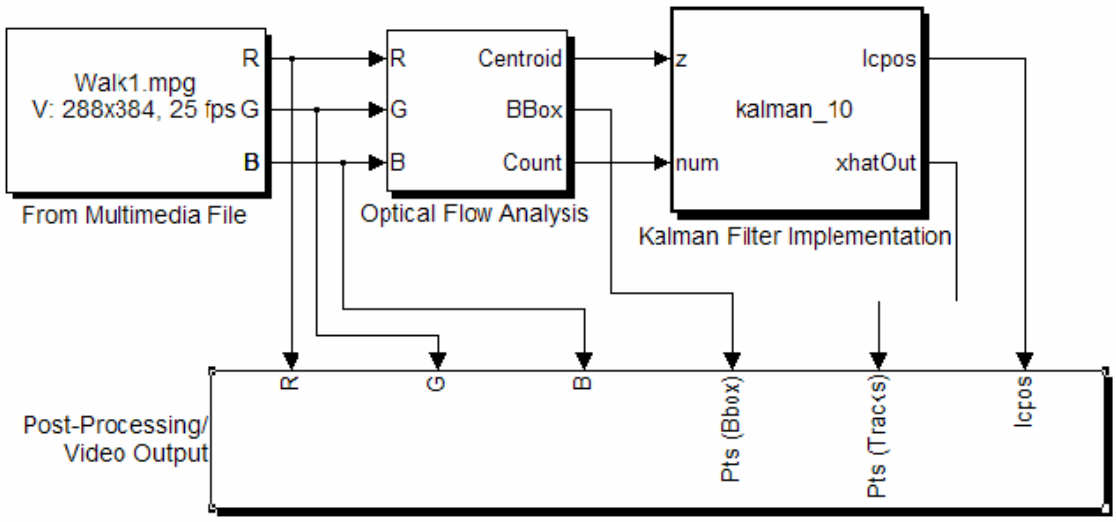

Figure 1. Simulink Block Diagram of Surveillance Video Processor

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This following chapter presents background information for both the Optical Flow algorithm and the Kalman Filter. While neither section provides exhaustive derivations, the information which is most relevant to application in this thesis is included. Wherever appropriate, references to more detailed explanations and derivations are provided for the reader's convenience.

A. OPTICAL FLOW

In their 1980 paper [4], Berthold K. P. Horn and Brian G. Schunk presented a novel algorithm (Horn-Schunk Algorithm) for determining the optical flow in a sequence of images (or frames of video.) Optical flow is a technique which determines the "distribution of apparent velocities of movement of brightness patterns in an image" [4]. These apparent velocities can arise from either motion of the objects or motion of the viewer. In this thesis, only the case of object motion will be considered due to the constraint of a stationary camera.

A typical frame of data at the output of the Horn-Schunk algorithm consists of a vector field with velocity vectors corresponding to each pixel in the image (Figure 2.) If each pixel's velocity were independent of the other pixel velocities in the image, it would be impossible to obtain an accurate vector field [4]. Horn and Schunk realized that each scene is generally composed of a finite number of objects. These objects occupy many adjacent pixels within a scene, and each of the pixels in a given object should ideally have similar velocities. To ensure this behavior, Horn and Schunk introduced a constraint called the *smoothness factor*.

The smoothness factor is defined as a constraint which controls how smoothly the velocity field of the brightness patterns in images varies throughout the image. The Horn-Schunk algorithm quantifies the smoothness of the velocity field using the magnitude of the gradient of the optical flow velocity defined as

$$\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial u}{\partial y}\right)^2 \text{ and } \left(\frac{\partial v}{\partial x}\right)^2+\left(\frac{\partial v}{\partial y}\right)^2, \quad (1)$$

where *u* and *v* are the velocity vectors corresponding to optical flow.

A low value for this gradient indicates that the vector field is very smooth. A high value for this gradient indicates that the vector field is not smooth. A smooth vector field tends to zero-out regions where no motion is detected leaving only limited areas of non-zero vector fields. By separating the image in this fashion, the Horn-Schunk algorithm achieves binary segmentation into regions of motion and regions of no motion.

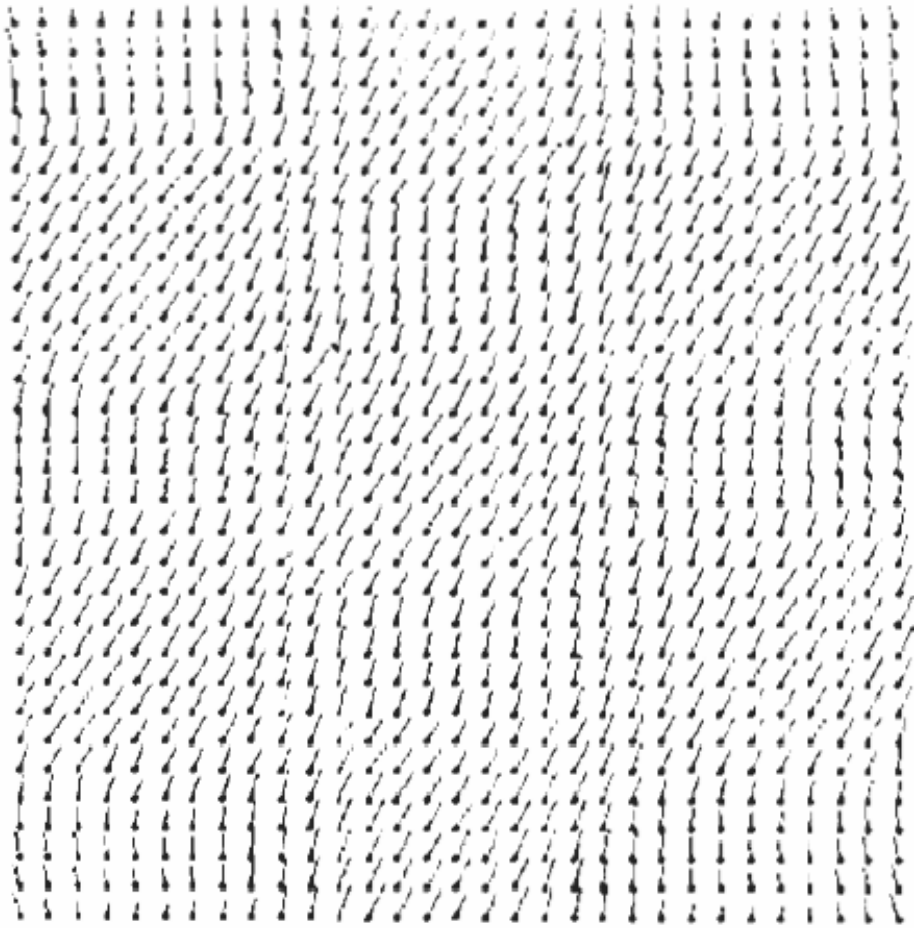

Figure 2. Sample Velocity Vector Field [After Ref. 4]

In this thesis the Optical Flow algorithm allows us to perform the first step in anomaly detection and alarming, namely identifying contacts. Through segmentation of a sequence of video frames into regions of non-motion and regions of motion, it is possible to separate the background of an image from contacts moving through the image. Only after contacts have been identified can contact behavior be monitored. The following section discusses the Kalman filter which is used for contact monitoring, anomaly detection and alarming.

B. KALMAN FILTER

In 1960, R. E. Kalman presented an alternate way to solve the problem of *linear minimum mean-square error filtering* using state space methods [5]. In essence, the Kalman filter is a type of recursive, adaptive filter that operates in the state-space. The discrete Kalman filter is characterized by both a process model and a measurement equation.

The process model is characterized by the assumption that the present state, x_k, can be related to the past state, x_{k-1}, by the relationship [5]

$$x_k = \Phi_k x_{k-1} + w_k \tag{2}$$

Here, w_k is assumed to be a discrete, white, zero-mean process noise with known covariance matrix, Q_k. The matrix Φ_k is referred to as the state transition matrix which determines the relationship between the present state and the past state.

In the case of this thesis, we attempt to track the state of a contact based on its last known state. Here, the state will consist of a two-dimensional position expressed in Cartesian coordinates, a two-dimensional velocity and a two-dimensional acceleration. As such, the state vectors will be vectors of size six by one with the first two elements corresponding to position, the next two to velocity, and the last two to acceleration. Assuming constant acceleration, the normalized state transition matrix can be obtained from basic kinematic equations as follows.

$$s_k = s_{k-1} + v_{k-1}t + \frac{1}{2}a_{k-1}t^2, \tag{3}$$

$$v_k = v_{k-1} + a_{k-1}t, \tag{4}$$

$$a_k = a_{k-1}. \tag{5}$$

Here *s* is defined as contact position, *v* is defined as contact velocity and *a* is defined as contact acceleration. The variable *t* is the sampling interval. In matrix form for the two-dimensional problem considered in this study, these kinematic equations may be rewritten as

$$\begin{bmatrix} s_{x,k} \\ s_{y,k} \\ v_{x,k} \\ v_{y,k} \\ a_{x,k} \\ a_{y,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x,k-1} \\ s_{y,k-1} \\ v_{x,k-1} \\ v_{y,k-1} \\ a_{x,k-1} \\ a_{y,k-1} \end{bmatrix} \tag{6}$$

Here, the subscripts *x* and *y* refer to the direction of the contacts position, velocity and acceleration in the two-dimensional plane. The sampling interval *t* is assumed to be 1. From this equation it follows that the state transition matrix, Φ_k, is defined as

$$\Phi_k = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

The measurement equation is defined as

$$z_k = H_k x_k + v_k. \tag{8}$$

Here, v_k is assumed to be the discrete, white, zero-mean measurement noise with known covariance matrix R_k. The variable z_k is defined as the vector

measurement. In this thesis, only position will be measured (not acceleration or velocity). As such, z_k is a vector of length two. The variable H_k is a matrix which describes the ideal relationship between the measurement vector, z_k, and the state vector, x_k. Given the state vector is of length six and the measurement vector is of length two, the matrix H_k must be of size two by six. It follows from the above description that H_k is defined as

$$H_k = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \end{bmatrix}. \tag{9}$$

Drawing from both the process model and the measurement equation, the Kalman filter attempts to improve the prior state estimate using the incoming measurement which has been corrupted by noise. Brown and Hwang [6] achieve this improvement by linearly blending the prior estimate, $\hat{x}_{k-1}$, with the noisy measurement, z_k, in the equation

$$\hat{x}_k = \hat{x}_k^- + K_k\left(z_k - H_k\hat{x}_k^-\right). \tag{10}$$

Here the superscript minus sign refers to the a-priori estimate. The factor, K_k, is referred to as a blending factor. It is shown [6] that the minimum mean squared error of the estimate is obtained when the blending factor assumes the value of the Kalman gain, i.e.

$$K_k = P_k^- H_k^T\left(H_k P_k^- H_k^T + R_k\right)^{-1}. \tag{11}$$

The variable P_k in Equation (11) is known as the state error covariance matrix, or simply either the state covariance matrix or the error covariance matrix. Generally the state covariance matrix is a diagonal matrix as implied by the assumption of independence of the error vectors, $e_k = x_k - \hat{x}_k^-$, at all lags not equal to zero. The state covariance matrix is calculated from the a-priori state covariance matrix by the equation

$$P_k = (I - K_k H_k)P_k^-. \tag{12}$$

In this thesis, P_k is a matrix of size six by six and thus I is the identity matrix of size six by six.

After the Kalman gain has been calculated (Equation (11)), and the state and state error covariance matrices have been updated (Equations (10) and (12)), the Kalman filter makes projections for the next value of k. These projections will be used as the *a-priori* estimates during processing of the next frame of data.

$$\hat{x}^{-}_{k+1} = \Phi_k \hat{x}_k \tag{13}$$

$$P^{-}_{k+1} = \Phi_k P_k \Phi^T_k + Q_k. \tag{14}$$

Equations (13) and (14) are the two projection equations for the state estimate and the state covariance matrix, respectively.

Figure 3 shows the Kalman filter loop in a flow diagram form. For the implementation used in this thesis, persistent variables for both the state estimate and the state covariance matrix were used, as described further in Chapter IV. In short, this choice allowed us to not distinguish between *a-priori* and *a-posteriori* estimates derived in the Kalman filter implementation. Instead of differentiating between *a-priori* and *a-posteriori* estimates, each estimate was overwritten as the next was calculated. That is, the previous frame's *a-posteriori* estimate was replaced by the current frame's *a-priori* estimate. Similarly, the current frame's *a-posteriori* estimate replaced the current frame's *a-priori* estimate after it was calculated. This required that no differentiation be made between *a-priori* and *a-posteriori* estimates and thus the same memory location could be allocated for both. This however is the only difference between the notation used in the equations in Figure 3 and the notation used in the Embedded MATLAB Function code found in the Appendix, which includes the Kalman filter tracking code developed in the study.

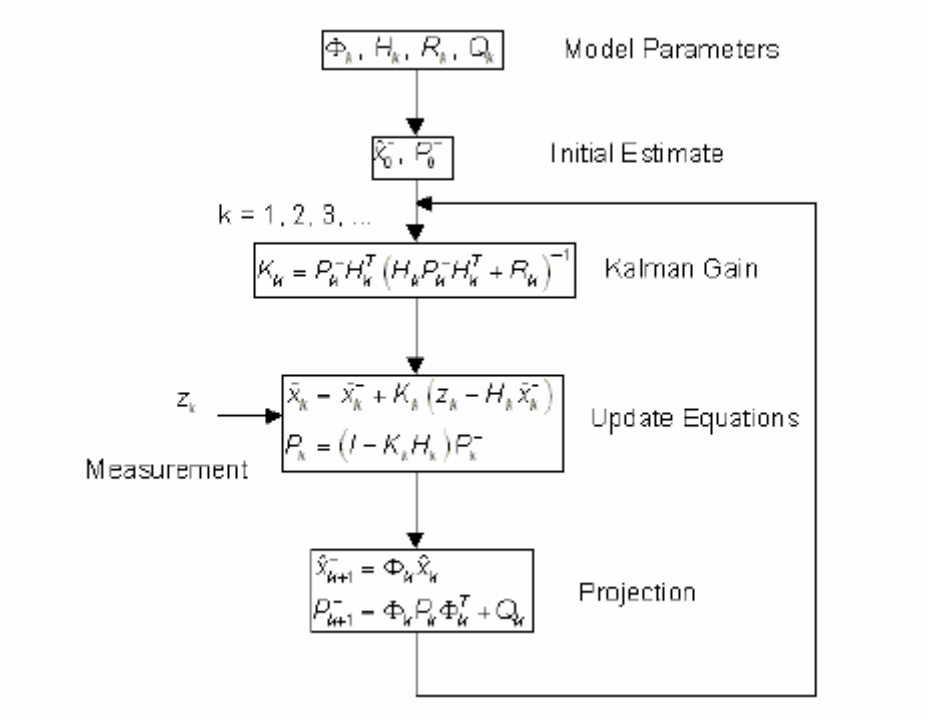

Figure 3. Kalman Filter Recursion [From Ref.7]

THIS PAGE INTENTIONALLY LEFT BLANK

III. OPTICAL FLOW ANALYSIS

Aside from post-processing, there are two main blocks in the Simulink Block Diagram in Figure 1. The first of these blocks, the *Optical Flow Analysis* block, will be analyzed in this section. The purpose of the Optical Flow Analysis block is to determine contacts in the incoming surveillance video and process them in such a way that the Kalman filter will be able to track contacts with minimal error. Error can be caused by a variety of factors, most notably changes in illumination that are perceived as contacts moving through the scene.

A. OVERVIEW

A view of the inner workings of the Optical Flow Analysis subsystem is found in Figure 4. The rest of this chapter describes the specific task performed by each block contained in this subsystem.

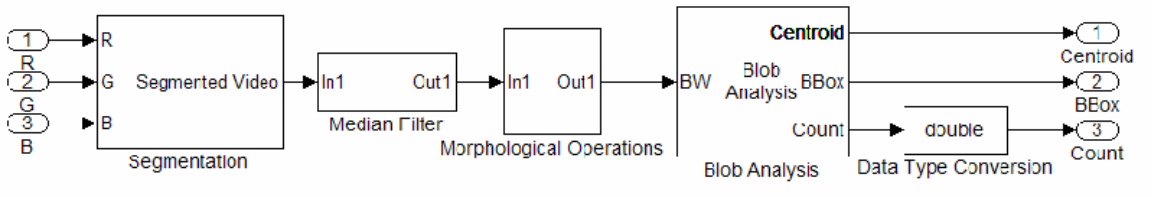

Figure 4. Simulink Block Diagram of Optical Flow Subsystem

It should be noted here that throughout the Optical Flow Analysis Block, certain subsystems are tuned for optimal performance given the CAVIAR data [3] used in this study. As such, results would undoubtedly differ if the exact subsystem were to be used on a different set of data. Given the application of the system created in this study, it is conceivable that cameras providing the data would be fixed. This implies that properties of the data from the same camera would not differ substantially over time. That said it is not only conceivable, but inevitable, that some initial tuning would be required to achieve optimum performance. Special care was taken to ensure, however, that the system was

not *over-tuned* to perform well on only certain data taken from the CAVIAR study [3]. Throughout this chapter, any tuning performed will be conspicuously documented.

B. SEGMENTATION

The term *segmentation* is used here to describe the processing of a video stream in such a way that it becomes a sequence of binary images. At the output stage of this subsystem each of these binary images has black areas (zeros) corresponding to portions of the frame in which no motion is detected. Similarly, each frame of the segmented stream will also have white areas (ones) corresponding to the frames in which motion is detected. The inner workings of the segmentation subsystem are shown in Figure 5.

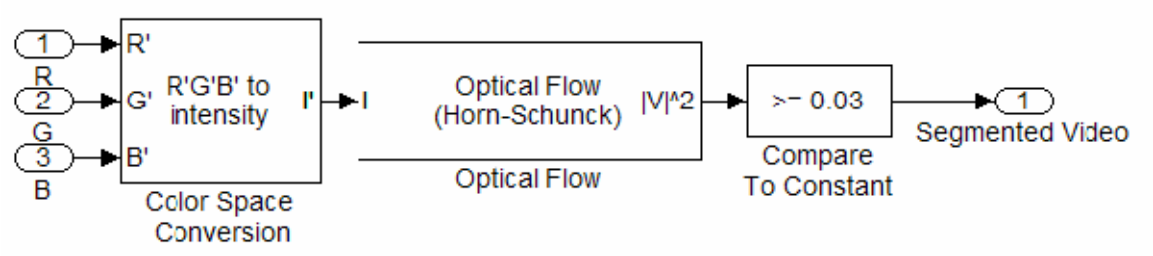

Figure 5. Simulink Block Diagram of Segmentation Subsystem

1. Color Space Conversion

The *Color Space Conversion* block (Figure 5) converts the color information from the incoming video (Figure 6) to the intensity color space (Figure 7) The video sequence which was previously coded in Red, Green and Blue (RGB) color space is now coded by intensity as required by the *Optical Flow* block.

Figure 6. Sample Frame of Original Video

2. Optical Flow

The *Optical Flow* block (Figure 5) uses the Horn-Schunk algorithm described earlier to estimate the optical flow between two video frames. The tuning of several parameters was required in this block. The block computes the optical flow between the current frame and the N-th frame back in time. As such the parameter N was tunable to the constraints of the data in this study.

Figure 7. Sample Frame of Original Video after Color Space Conversion

We noted that it is unrealistic that in the time difference between two successive frames a person will have moved significantly given the 25 frames per second (fps) frame rate. Thus it is unlikely that the algorithm would detect significant optical flow and consequently no motion would be have detected if N had been set to a low number (e.g., 1). By setting the value of $N = 10$, the algorithm uses a delay of 0.4 seconds to estimate optical flow, which is a reasonable choice for N given the walking speed of the average person. A lower N value would be sufficient in applications (e.g. traffic monitoring), where contact speeds are faster.

Figure 8. Sample Frame of Video after Optical Flow Block

Another tunable parameter in this block was the smoothness factor, as discussed in Chapter II. The smoothness factor input in Simulink is inversely proportional to the magnitude of the velocity gradients discussed in Chapter II. That is, a low value for the smoothness factor results in a high gradient, and thus a velocity vector field which is less smooth. Conversely, a high value for the smoothness factor results in a low gradient, and thus a velocity vector field which is smoother. In this thesis, the value of the smoothness factor was set equal to 0.5. By visual inspection, allowing the algorithm to detect motion throughout the scene without being overly sensitive to noise.

3. Compare to Constant

For each pixel in a frame of video, the *Optical Flow* block estimated the optical flow between the pixel in that frame and the same pixel in the 10th frame back. At the output of that block, each frame consists of a frame of optical flow vector magnitudes corresponding to each pixel (Figure 8.) For those pixels in which little motion is detected, the value of this optical flow vector is close to zero. For those pixels in which motion is detected, the optical flow vector is non-zero. By viewing a histogram of the optical flow output, it is trivial to find a constant value that is higher than zero and lower than the value for the majority of non-zero optical-flow vector magnitudes to serve as a threshold.

The *Compare to Constant* block (Figure 5) compares the magnitude of the flow vector at each pixel to the tuned value of 0.03, resulting in a sequence of binary frames. The flow vector values which are greater than this constant correspond to areas of motion and are represented by white areas (ones) at the output of the *Segmentation* subsystem. Similarly, the flow vector values which are less than this constant correspond to areas in which no motion is detected and are represented by black areas (zeros) at the output of the *Segmentation* subsystem (Figure 9.)

Figure 9. Sample Frame of Video at Output of Segmentation Subsystem

C. MEDIAN FILTER

One issue with the optical flow algorithm is the propensity, even with proper tuning, to be sensitive to changes in illumination or quality of video from frame to frame. This sensitivity often results in erroneous blobs appearing in individual frames at the output of the *Segmentation* subsystem (Figure 5.) If these erroneous blobs are small, they can generally be accounted for in the morphological operations described later in this chapter. However, large erroneous blobs (here large is defined as anything approaching the average size of a person tracked in the scene) create problems for successful morphological operations.

Two factors led to the use of a median filter to attempt to correct the issues caused by erroneous blobs. In general, most of these erroneous blobs are anomalies. That is, they appear in singular frames, not to appear again for at

least several more frames. Moreover, in a span of three frames at a frame rate of 25 fps, the movement of intended contacts is very small and most of the pixels in which movement is correctly detected will be the same. A median filter serves to decrease the effect of anomalies (or erroneous blobs) while still maintaining the information of the correctly detected contacts. The exact implementation of this median filter is shown in Figure 10.

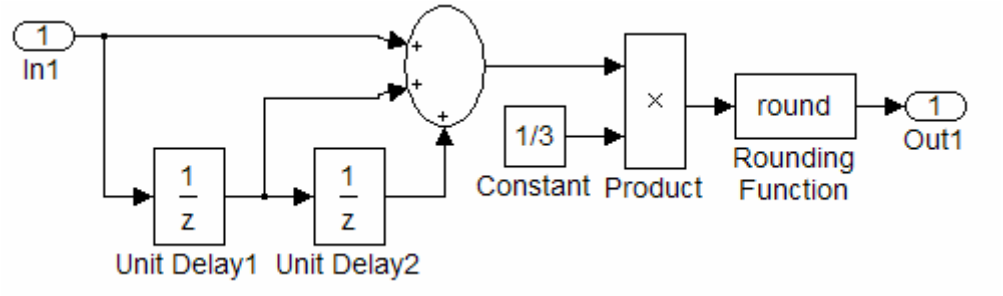

Figure 10. Simulink Block Diagram of Median Filter Implementation

Figure 11 shows three successive frames that serve as an input to the median filter as implemented in Figure 10. Each of the frames has correctly detected three general areas of motion. The second of these frames also has erroneous blobs that are most likely due to slight illumination changes in the scene.

Figure 11. Three Successive Frames of Segmented Video Used as an Input to the Median Filter Implementation

Given each of the frames is a binary image, the median filter of length equal to three is implemented as a combination of a simple average of each pixel over three frames followed by a rounding function operation. This approach was

chosen over a direct median filter operation due to the way in which MATLAB defines the Median function. In MATLAB if the input to a median function is an *M* by *N* matrix, the output will be a length-*M* vector consisting of the median value of each of the *N* column vectors. This obviously was undesired, and thus the filter was implemented as described here.

At the output of the *Product* block (Figure 10) each pixel has one of four values: 0, 1/3, 2/3 or 1. Rounding these values using the *Rounding Function* block results in a binary image identical to that which would have been obtained had a traditional median filter been applied on each pixel over a time-span of three frames. The result is a binary image for each frame which eliminates many erroneous blobs while maintaining correct detection of contact movement (Figure 12.)

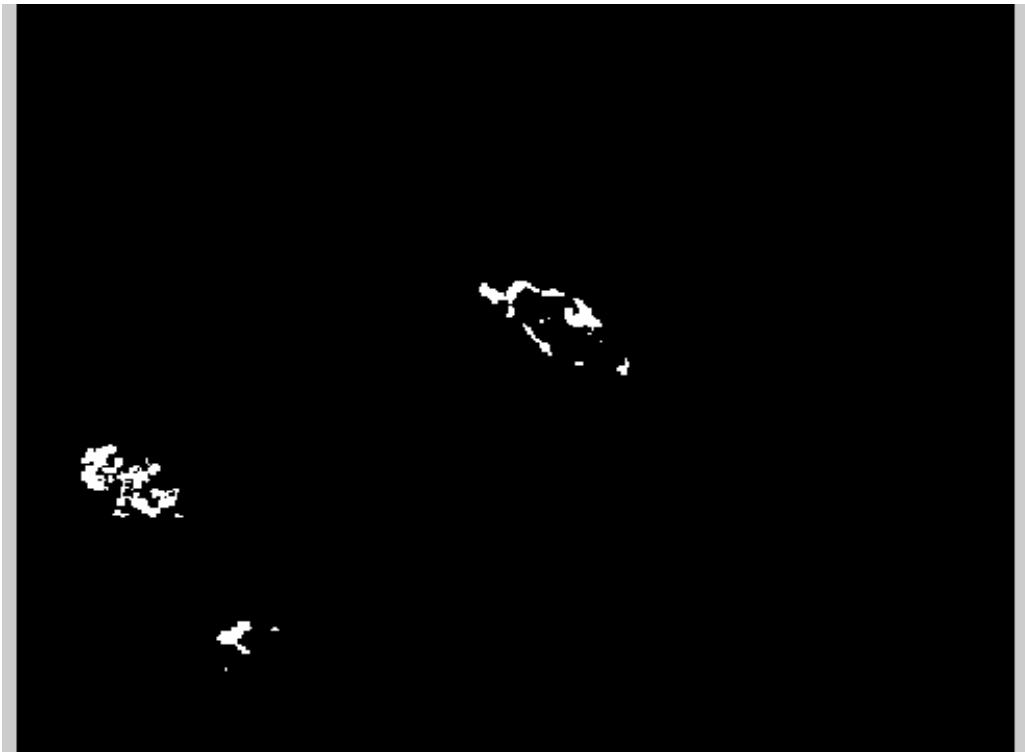

Figure 12. Frame without Erroneous Blobs at Output of Median Filter Implementation

D. MORPHOLOGICAL OPERATIONS

The *Morphological Operations* subsystem attempts to process the video at the output of the *Median Filter* subsystem in such a way that any remaining erroneous blobs are eliminated and all correctly detected contacts are maintained and classified as one contact. The inner workings of the *Morphological Operations* subsystem are presented in Figure 13.

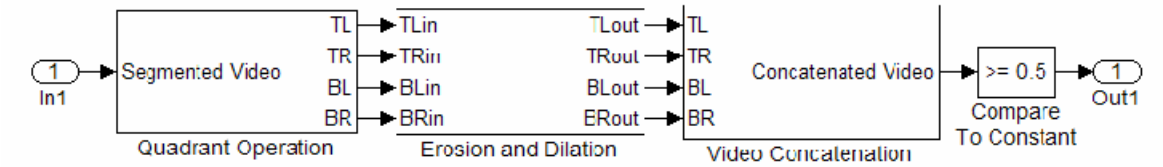

Figure 13. Simulink Block Diagram of Morphological Operations Subsystem

1. Quadrant Operation

Due to the wide-angle camera used in the CAVIAR data [3], the average size of a contact in the bottom right quadrant of the video is somewhat larger than the average size of a contact in the top left quadrant of the video. In order to provide flexibility for the erosion and dilation operations depending on a blob's location in the scene, each frame was split into overlapping quadrants. Further flexibility could have been obtained by again splitting each quadrant into smaller quadrants. However, such a step was deemed unnecessary given the size of the frames in this study and the relatively small difference in contact size within a primary quadrant. The block diagram of the *Quadrant Operation* subsystem is presented in Figure 14.

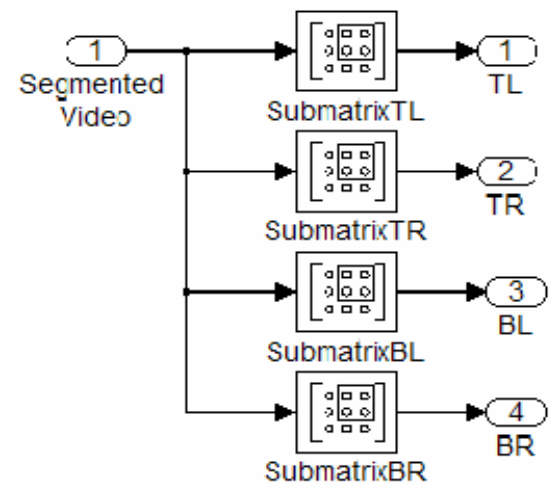

Figure 14. Simulink Block Diagram of Quadrant Operation Subsystem

One issue with splitting the data into quadrants is the possibility that at any time a blob may reside in more than one quadrant. In the extreme case where the blob is resting in all four quadrants, it is likely that erosion could eliminate the partial-blob from each quadrant and the contact would be lost. As a result, each quadrant overlapped the lines of horizontal and vertical symmetry by ten pixels to prevent this situation from occurring. This amount of overlap was found to be sufficient to ensure that contacts were not lost with the selected 4-pixel wide erosion block described later in this chapter. Figure 15 presents the four sub-frames at the output of the *Quadrant Operation* block to be used in the *Erosion and Dilation* block. The frame which generated these sub-frames is found in Figure 12.

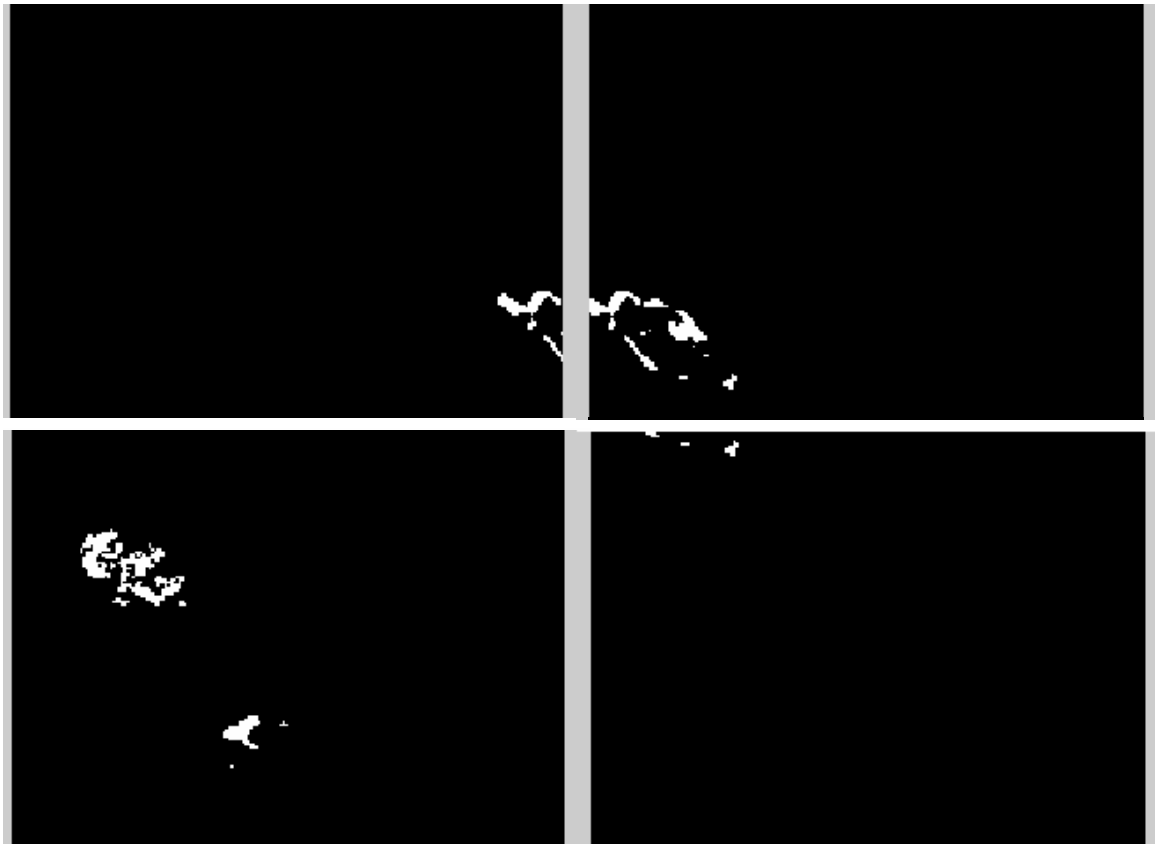

Figure 15. Segmented Frame Split into Four Quadrants at the Output of the Quadrant Operation Subsystem

2. Erosion and Dilation

The *Erosion and Dilation* subsystem performs tailored erosion and dilation of each sub-frame by using optimal structuring elements on each sub-frame (Figure 16.) Optimal erosion is achieved when the structuring element used keeps at least the remnants of a blob for all correct contacts. If a sub-optimal structuring element is used for the erosion operation, a valid contact could be lost completely, or an erroneous blob could be tracked. Both of these errors create significant barriers to optimal dilation, and, consequently, Kalman filter tracking. Optimal dilation is obtained when the structuring element used merges all remnants of a single blob into one contact. If a sub-optimal structuring element is

used for dilation, one contact could be viewed as multiple contacts or multiple contacts could be viewed as one contact. Again, both of these errors create significant issues for Kalman filter tracking.

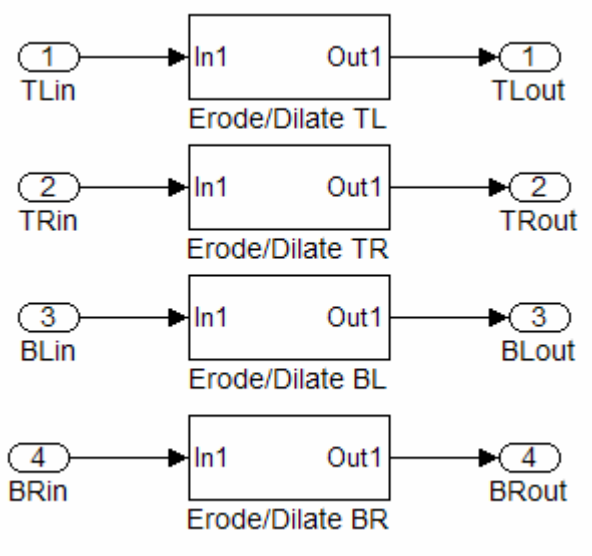

Figure 16. Simulink Block Diagram of Erosion and Dilation Subsystem

Next, each quadrant was processed with a unique set of structuring elements to allow for optimal morphological processing (Figure 17)..

Note that the optimal structuring elements to be used in each quadrant were derived empirically by processing a group of ten random frames from the CAVIAR data [3]. These frames were split into quadrants. Next, resulting sub-frames were grouped according to their quadrant, and each group of quadrant specific sub-frames eroded using a square structuring element of edge-length varying from two to five. At that point, the eroded sub-frames were compared to the corresponding original frames and a decision was made on the optimal structuring element size via visual inspection.

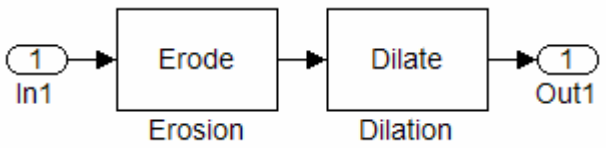

Figure 17. Simulink Block Diagram of a Erosion and Dilation Subsystem for a Particular Quadrant

After an optimal structuring element for erosion was determined, each group of sub-frames was eroded using said structuring element. Determination of the optimal structuring element for dilation was similar to that of erosion. Each group of sub-frames was dilated with a square structuring element of edge-length which varied from five to nineteen. The dilated sub-frames were compared to the corresponding frames and a decision was made on the optimal structuring element size via inspection.

Quadrant	Erosion Edge-Length (pixels)	Dilation Edge-Length (pixels)
Top Left	4	11
Top Right	4	14
Bottom Left	4	14
Bottom Right	4	15

Table 1. Optimal Edge-Length for Square Structural Elements

The optimal edge-lengths for the structural elements in each quadrant are presented in Table 1. For simplicity, only square structuring elements were used in this tuning process. An infinite number of possibilities exist for size and shape of structuring elements. Depending on the data used, the size and shape of the optimal structuring element could vary significantly from quadrant to quadrant.

3. Video Concatenation

Before the morphed frames can enter the *Blob Analysis* subsystem (Figure 4) the sub-frames must be concatenated to form a complete frame again. Since each quadrant overlapped the adjacent quadrants, some pre-processing was necessary before concatenation. The subsystem used for concatenation is presented in Figure 18.

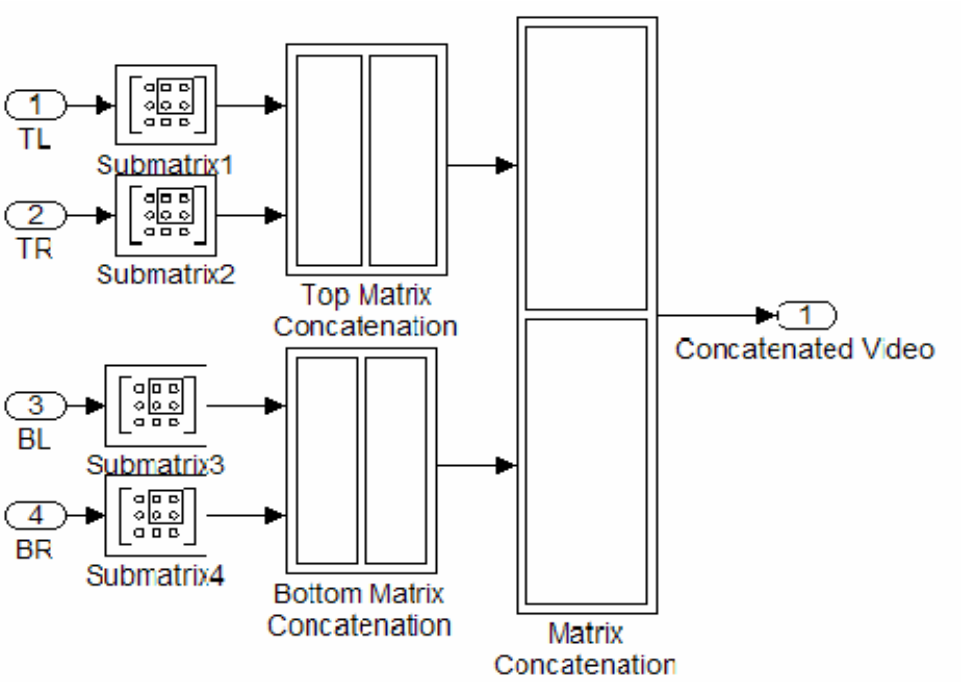

Figure 18. Simulink Block Diagram of Video Concatenation Subsystem

Four S*ubmatrix* blocks are used prior to horizontal concatenation of the top left and top right quadrants, as well as the bottom left and bottom right quadrants. These blocks return sub-frames that contain no overlap such that concatenation will result in frames which are of the same size as the original. Following the horizontal matrix concatenation, the resulting sub-frames are half of the original frame size. These sub-frames are concatenated vertically to create a full frame. Next, a *Compare to Constant* block is applied at the output of the *Video Concatenation* subsystem to ensure the frame is a binary image containing Boolean values of zero and one.

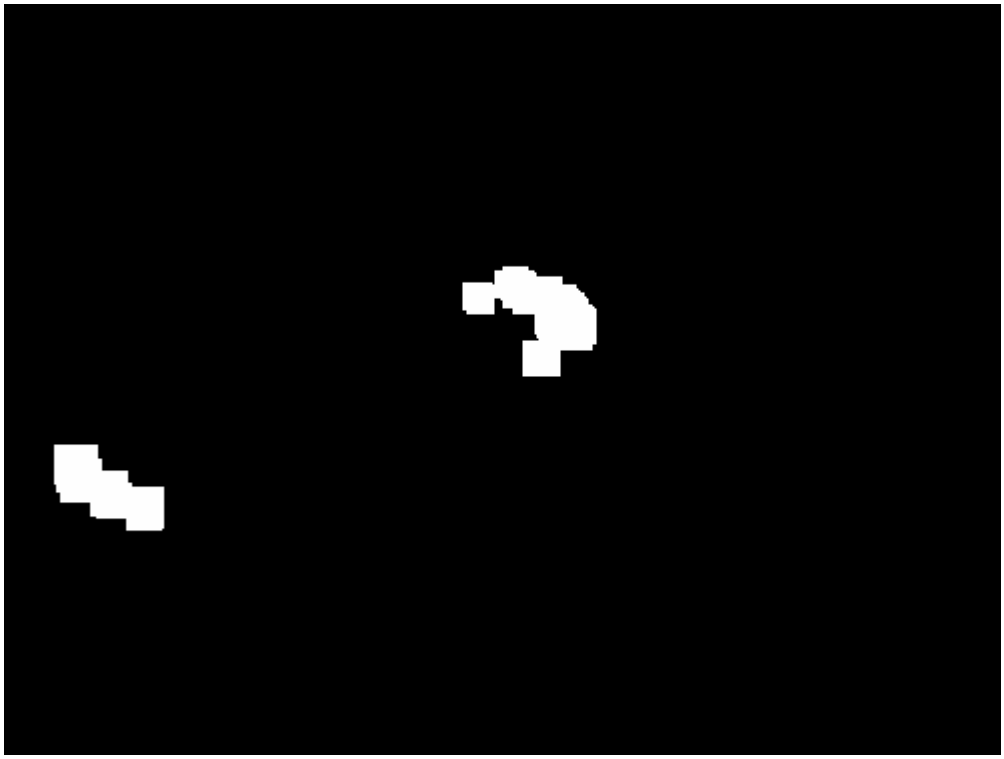

Figure 19. Sample Frame after Morphological Operations

A sample frame at the output of the *Morphological Operations* subsystem is presented in Figure 19. Comparing this frame to the frame it was generated from (Figure 12) shows that the erosion operation removed one of the areas of detected motion, thus deeming it not a contact. Further dilation has created solid blobs out of the other two areas of detected motion and these two blobs will be tracked as contacts.

E. BLOB ANALYSIS

The *Blob Analysis* block (Figure 4) processes the binary frames at the output of the *Morphological Operations* subsystem for use in both system monitoring and Kalman filter tracking. Several parameters are set in the *Blob Analysis* block to improve system functionality: minimum blob size and maximum number of blobs.

The minimum blob size provides an added level of protection against erroneous blobs by specifying a minimum size that a blob must be in order to be tracked. Thus, any blob smaller than the minimum size requirements is not tracked. The minimum blob size was set to approximately 250% of the size of the largest dilation structural element (225 pixels) or 560 pixels.

The maximum number of blobs provides a bound on the number of contacts that can be tracked by the system. For this study the maximum number of contacts was set to ten. This number was low enough so as not to be unwieldy, yet high enough to allow for the tracking of multiple contacts. This number was also used to determine the maximum number of Kalman filters to make available for tracking in the *Kalman Filter Tracking Implementation* discussed in Chapter IV.

1. System Monitoring

As a debugging tool, the output of the *Optical Flow Analysis* subsystem (Figure 1) is a convenient place to analyze how the system is performing. The *Blob Analysis* block has an option which when selected produces the coordinates of a bounding box around each blob on a frame by frame basis. By selecting this option, these bounding boxes can be superimposed on the video sequence and the user has a simple method of monitoring system performance. The details of this monitoring process will be explained further in Chapter V. A sample frame with bounding boxes superimposed is presented in Figure 20.

Figure 20. Sample Frame with Bounding Boxes Superimposed

2. Kalman Filter Tracking

The output of the Blob Analysis block is also used as an input to the *Kalman Filter Implementation* (Figure 1.) The *Blob Analysis* block has two options that were selected for use in the *Kalman Filter Implementation*. The number of blobs found is provided as the input *count* to the *Kalman Filter Implementation*. The location of the center of mass of each blob is also provided as the measurement input *z* to the *Kalman Filter Implementation*. The exact implementation of the Kalman filter algorithm will be discussed next in Chapter IV.

IV. KALMAN FILTER IMPLEMENTATION

Aside from post-processing, there are two main blocks in the Simulink Block Diagram in Figure 1. The second of these blocks, the *Kalman Filter Implementation* block, will be analyzed in this section. The purpose of the Kalman Filter Tracking Implementation is two-fold: to correctly assign a tracking filter to each of the measurements entering the system from the Optical Flow Analysis block and to alert the user when a contact has been lost (a significant event has occurred.)

A. OVERVIEW

An *Embedded MATLAB Function* block was used in the Simulink Block Diagram shown in Figure 1, to implement the Kalman Filter Tracking algorithm. This type of block is often used when the function to be performed is more easily expressed in MATLAB's symbolic language than in Simulink's graphical language [8]. While most of the functionality present in MATLAB can be utilized in an Embedded MATLAB Function, there are some limitations to its use. Two types of limitations of interest are limitations on defining the type and size of variables and limitations on indexing operations.

1. Imbedded MATLAB Functions Constraints

Embedded MATLAB functions require that the type and size of a variable must be defined prior to being called for the first time [8]. In short this means that a variable can never be defined as an empty matrix. It also required that careful thought be given to how to handle the data in this thesis. That is, even though at any given time we may be tracking less than ten contacts, which is the maximum number of contacts assumed to exist in the video at a given time, the Embedded MATLAB Function must be written in such a way that it can accommodate the maximum number of possible contacts. As a result, ten filters were created in

the function and the number of necessary filters to access in a given frame was determined based on the number of contacts at the output of the *Optical Flow Analysis* subsystem.

The limitations on indexing operations are related to the limitations on the type and size of variables. In short, Simulink requires that the indices of a variable be defined explicitly when it is indexed within a loop.. That is, this variable must be defined in such a way that a known quantity of memory can be allocated to it prior to execution of the loop. This constraint sometimes created the need for *workaround* techniques when indexing certain variables in this thesis. These *workaround* techniques may appear as inefficient when looking at the code present in the Appendix. However, throughout this chapter, the reasons behind these *workarounds* will be conspicuously documented to benefit the reader.

The rest of this chapter presents the details of each section of the tracking algorithm. Figure 21 presents a flow diagram which illustrates the progression of the code for each frame. This flowchart and the code present in the Appendix should be used as references when reading the remainder of this chapter.

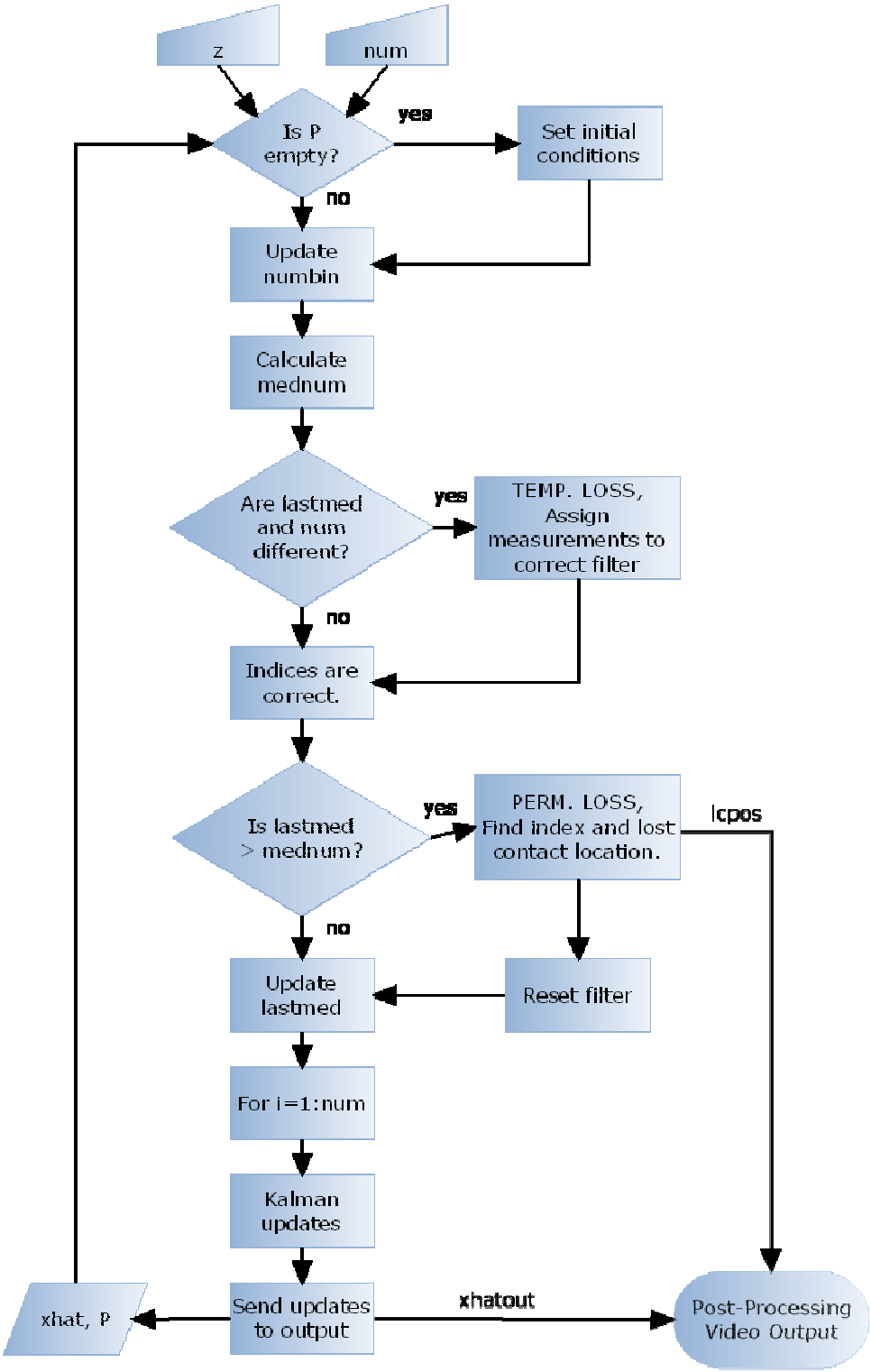

Figure 21. Flow Diagram for Embedded MATLAB Function Kalman Filter Implementation

B. INITIALIZATION

The initialization portion of the Kalman Filter Tracking function first includes the definition of several persistent variables. Persistent variables are variables which are local to the function in which they are defined. However, their value is retained between calls to the function. This functionality gracefully complemented the reliance of the Kalman Filter on the position estimates from the previous frame. Persistent variables allowed for a type of feedback system that operated within the constraints of the Embedded MATLAB Function block.

Thus, the algorithm checks whether one of these persistent variables is empty prior to processing each frame Note that persistent variables are empty only when the algorithm first starts, i.e. when the first frame of data is processed. In such a case, initial conditions must be set for several variables. The rest of this section will be devoted to discussing the specific initial conditions set for each variable of interest.

1. State Estimates

The state estimates for each of the ten Kalman filters are referred to as the variable *XHAT*. *XHAT* is defined as a matrix of size six by ten, where each column corresponds to a specific state estimate for a given filter. The first two rows of the column vector represent the coordinates (row, column) of the position estimates. The next two rows are the velocity estimates, and the last two rows are the acceleration estimates. Thus, a one-filter *XHAT* has the form

$$XHAT = \begin{bmatrix} \text{row position} \\ \text{column position} \\ \text{row velocity} \\ \text{column velocity} \\ \text{row acceleration} \\ \text{column acceleration} \end{bmatrix}. \tag{15}$$

Note that the initial positions were centered in the middle of the frame, so as not to bias initial conditions. Thus, the initial row and column positions for each filter were set to 144 and 192, respectively, for a frame size of 288 pixels by 384 pixels. Velocity and acceleration pixels were set to small non-zero values of 0.1. Thus, after initialization, the initial six-by-ten matrix, *XHAT*, took the form

$$XHAT = \begin{bmatrix} 144 & 144 & ... & 144 \\ 192 & 192 & ... & 192 \\ 0.1 & 0.1 & ... & 0.1 \\ 0.1 & 0.1 & ... & 0.1 \\ 0.1 & 0.1 & ... & 0.1 \\ 0.1 & 0.1 & ... & 0.1 \end{bmatrix}. \tag{16}$$

2. State Covariance Matrices

In addition to state estimates, each filter is defined by its own state covariance matrix of size six by six. The matrix *PP* implemented in the algorithm is a side-by-side concatenation of ten of these covariance matrices to form a matrix of size six by sixty. Each of these matrices is a diagonal matrix, as defined in Chapter II. The first two elements on the diagonal correspond to the covariance of the position estimate. The next two elements correspond to the covariance of the velocity estimate. The last two elements correspond to the covariance of the acceleration estimate.

The initial conditions for each of these matrices are set arbitrarily. Given the position of the contacts are known with greater accuracy than both the velocity and acceleration, the values increase along the diagonal. If we assume the position of the contact is known within three pixels, the position covariance was initialized at 3^2 or nine. Similarly, assuming the velocity is known within five pixels per second and the acceleration is known within seven pixels per second

squared, we obtain covariance values for velocity and acceleration of 25 and 49, respectively. After initialization, the six by sixty covariance matrix, *PP*, takes the form

$$PP = \begin{bmatrix} 9 & 0 & 0 & 0 & 0 & 0 \ldots 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 \ldots 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 25 & 0 & 0 & 0 \ldots 0 & 0 & 25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 25 & 0 & 0 \ldots 0 & 0 & 0 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 49 & 0 \ldots 0 & 0 & 0 & 0 & 49 & 0 \\ 0 & 0 & 0 & 0 & 0 & 49 \ldots 0 & 0 & 0 & 0 & 0 & 49 \end{bmatrix}. \quad (17)$$

Throughout the operation of the algorithm, these covariance matrices will be updated to reflect a more accurate representation of the covariance of the state estimates.

3. Number of Contacts Vector

The number of contacts in each frame needs to be monitored in order to keep track of changes in the contact behavior. However, changes cannot be accurately monitored on a frame-by-frame basis because of the fluctuations in the number of contacts at the output of the *Optical Flow Analysis* block. As a result, it was necessary to monitor the number of contacts over several frames and monitor *trends* in the number of contacts over those frames. The number of contacts was monitored over the last second of video (25 frames) in order to provide enough data to monitor these trends.

A vector used to monitor the number of contacts in the present frame and the 24 previous frames was created and referred to as *numbin*. Appropriate initial values for *numbin* must be set when the initial frame is processed. Thus, each value of *numbin* was set equal to the number of contacts in the first frame, *num,* to allow for tracking of all contacts found in the first frame. In other words, upon initialization, *numbin* is a vector of length 25 with all 25 values in the vector equal to *num*.

4. Median Number of Contacts

The median number of contacts in the last frame is a variable created to aid in the identification of lost contacts. It is referred to as *lastmed*. Note that there is no previous frame from which to obtain a value for *lastmed* when the algorithm processes the first frame of video. As a result, the parameter *lastmed* is initialized to the value of the number of contacts in the first frame, *num*.

5. Indexing Vector

An indexing vector *index* is used to keep incoming measurement values, i.e., locations, (z), state estimates (XHAT) and state covariance matrices (PP) properly associated. The indexing vector, *index*, of size ten by one is set to

$$index = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}' \tag{18}$$

to ensure each initial location is assigned a different filter.

C. CONTACT MONITORING

In this section we will discuss how contact behavior is monitored by the algorithm. The first sub-section will discuss how the algorithm determined how many filters to activate based on the number of contacts at the output of the Optical Flow Analysis Block. The second sub-section will discuss how each of these filters is assigned to the correct contact. The last sub-section will discuss how the algorithm handles a lost contact.

1. Median Filter

Despite the copious amounts of tuning discussed in Chapter III, the number-of-contacts output (*num*) of the *Optical Flow Analysis* block is susceptible to some degree of fluctuation. Note that these fluctuations are normally of no more than one contact in either direction, and usually do not last much longer than one frame. A persistent increase (or decrease) in the number of contacts indicates a gain (or loss) of contact. A change in the number of

contacts reflects an anomaly in the data at the output of the *Optical Flow Analysis* block when the behavior is not persistent. As such, a median filter of length 25 was designed to monitor the behavior of *num* over the current and the previous 24 frames to minimize the impact due to temporary fluctuations. The median filter allows temporary fluctuations due to anomalies in the *Optical Flow* Analysis block to be ignored and only persistent changes in the number of contacts to be tracked. The output of the median filter was also used to determine the number of Kalman filters needed for each frame of video.

The median operation was chosen instead of a traditional mean for two reasons. First, recall that the input and outputs to the median filter are always integer values. This behavior was desired as the output of the filter is used as an indexing operator for various loops throughout the function. The median filter allowed for correct indexing without the implementation of a rounding function.

Second, the median operation ensured that there was no need to prioritize which contacts would be tracked. The benefit of this behavior may be overlooked at first, however it proved quite powerful. As an example, assume that for 25 successive frames only one contact was present in the video. Next assume that for the next 25 successive frames nine contacts were present in the video. A simple moving-window average of length 25 over this length of time would yield values spanning from one to nine. For the time when the moving-window average filter yielded an output of five, for example, an undesirable result is produced. The algorithm must decide to which five of the nine contacts to assign a filter. A median filter over the same length of time would only have two possible outputs, either one or nine. The median filter eliminates the decision of which contacts to track by tracking either the initial contact, or the subsequent nine. While the example presented here is highly unlikely given the constraints of the problem, the same example can be shown to be true for smaller differences in the number of contacts (e.g. two and four).

Initializing this filter at the start of the algorithm is done by selecting each coordinate of the vector *numbin* to be equal to the value of *num* at the first frame.

This choice was made as it allowed for the median to be calculated at the first frame without error. For each subsequent frame, the last 24 entries in the vector are copied to the first 24 entries in the vector and the value of *num* for the current frame is inserted as the last value. The median value of the newly updated *numbin* is calculated as the parameter *mednum*. As mentioned, *mednum* is used later to determine the number of Kalman Filters needed in each frame.

2. Filter Indexing

After the initialization step is complete, the algorithm must assign a filter to each of the contacts in the measurement vector (*z*). This step is accomplished by comparing each incoming position in *z* to the previous estimated position values present in *XHAT* which have remained persistent from the last frame. It should be noted that throughout the function, it is assumed that the number of contacts from frame to frame changes by at most one in either direction. This assumption was deemed acceptable given the low contact density of the data used and the high frame rate of the video (25 fps).

a. Number of Contacts is Equal to the Median Number of Contacts Over the Last 25 Frames

No further indexing is necessary when the estimated variable *num* is equal to *mednum*, due to the initial conditions discussed above and the behavior of the *Blob Analysis block* (Figure 4). Note that a distinct filter is assigned to each contact because the parameter *index* was initially defined to include ten distinct values.

Note that the correct filter is still matched to the correct input measurement when values in *index* change from their initial conditions, because the *Blob Analysis* block (Figure 4) always orders its outputs in increasing order by row coordinate. Keeping the values in *index* constant in this case makes the assumption that one contact does not leave the scene at exactly the same time as another contact enters it. Given the frame rate of the data as 25 fps, the

likelihood of this event is negligible and this assumption is deemed acceptable. In the unlikely event that this assumption *is* false, the Kalman filter is flexible enough that it will equilibrate to the state of the new contact within a nominal amount of frames.

b. Number of Contacts is Greater than the Median Number of Contacts Over the Last 25 Frames

Indexing becomes necessary when *num* is found to be greater than *mednum*. While the added contact will not be immediately tracked due to the Median Filter described above, it is important that the new contact be identified and only the contacts present in the last frame be assigned filters. Thus, each measurement in the vector *z* is individually compared to all of the previous estimated positions in *XHAT* to assign the correct incoming measurement to the correct filter. This step is accomplished by calculating the vector norm of the difference between each measurement in *z* and all previous position estimates in *XHAT*. Next, the index of the estimate corresponding to the minimum norm, and consequently the index of the filter used to determine said estimate, is determined.

Given a contact will not realistically move more than a few pixels in the time between two frames, each of the previous position estimates will be correctly matched to the corresponding incoming measurement via the vector norm minimization calculation. For the situation where *N* contacts increase to *N+1* contacts, the new contact is assigned the filter of index *N+1.* This filter is defined by initial conditions described above. If this contact remains in the scene long enough for *mednum* to increase to *N+1*, the new contact will be tracked using the filter of index *N+1* under the conditions described in Subsection 2.a. above.

c. Number of Contacts is Less than the Median Number of Contacts Over the Last 25 Frames

Indexing also becomes necessary when *num* is smaller than *mednum*, which reflects a temporary loss of contact. As a result, the number of active filters is larger than the number of incoming measurements. Indexing occurs for the *N-1* measurements by computing the vector norm between each measurement and the *N* position estimates as described above.

Next the incoming measurements vector, *z*, is redefined to include a measurement for the contact which has been lost in the frame. In effect, the incoming measurement matrix of size two by *N-1* is redefined as a matrix of size two by *N*. The added measurement is obtained by copying the position estimate from the previous frame which corresponds to the lost measurement. This procedure results in the filter being continually updated if the contact is only lost for one or a few frames. If the loss of contact is persistent, (in this case for more than 13 frames in a 25 frame span of time) the contact is deemed lost and *mednum* assumes the value of *N-1*. Further processing of the lost contact will be described in the following sub-section.

3. Permanent Contact Loss

If a loss in contact is persistent for 13 out of 25 consecutive frames, the parameter *mednum* decreases to a value of one less than *lastmed*. This change is deemed a significant event and thus an alert will be sent to the user. In practical application, this situation could indicate that a contact has left the room, hidden behind an occlusion, or stopped moving. In order for the user to have some indication of which behavior has led to the loss of contact, the last known position of the contact will be marked on the current video frame. In the worst case scenario, this will be the position of the contact 13/25 frames ago, or approximately 0.5 seconds ago. By visually inspecting the current video

sequence the user should be able to identify which of the above behaviors has caused the contact to be lost.

The index of the lost contact is obtained via the method described above. The last position estimate which corresponds to this index is copied to the variable *LCpos*. This variable is used to create a mark on the video sequence corresponding to the lost contact in a process which will be described in Chapter V. Since the contact has now been lost, the filter corresponding to the lost contact is reset to its initial conditions. Finally, whether or not a contact has been lost, *lastmed* is set to the value of *medcount* so that it can be used in the processing of the next frame of video.

D. KALMAN TRACKING UPDATES

Before the Kalman tracking updates can be applied to each filter, the indexing vector is utilized to obtain the desired state estimates and covariance matrices from *XHAT* and *PP* respectively. A “for loop” was used to update each Kalman filter individually, where the vector *index* is used to access the correct state estimate in XHAT. Note that each column of the correct covariance matrix in *PP* is assigned separately to work around some of the indexing issues with Embedded MATLAB Functions as discussed above.

The individual state estimates and covariance matrices are assigned the variable-names *xhat* and *P*, respectively. Next, the Kalman equations are implemented, as described in Chapter II using the values *xhat and P*. After all equations are implemented, the updated values of *xhat* and *P* are copied to their corresponding locations in the variables *XHAT* and *PP*. Again to combat issues with indexing, the values copied to *PP* are copied one column at a time.

E. OUTPUT

The only relevant outputs from the *Kalman Filter Implementation* are the position estimates present in *XHAT* and the position of any lost contacts, *lcpos, in this* thesis. As such, the output of the Embedded MATLAB Function implementation consists of two variables. At each frame a two-by-ten matrix of position vectors (*xhatOut*) is produced. For *N* contacts, the first *N* position vectors in the matrix *xhatOut* correspond to the position of the *N* contacts. The remaining $10-N$ position vectors correspond to the initial position conditions as defined earlier in this chapter.

Also present at the output is a position vector of length two corresponding to the position of a lost contact. When there is no lost contact, *lcpos* assumes the value $\begin{bmatrix}0 & 0\end{bmatrix}'$. When a contact has been lost, *lcpos* assumes the value of the position vector corresponding to the last known location of the contact. The post-processing of these two output variables will be discussed further in Chapter V. In theory the persistent variables defined earlier are also outputs of the *Kalman Filter Implementation* block. These outputs however are of no practical use to the user and are only accessed by the algorithm upon processing of the next frame of video.

THIS PAGE INTENTIONALLY LEFT BLANK

V. POST-PROCESSING/RESULTS

In this chapter post-processing techniques used to monitor system performance are described. Results which describe the performance are also presented.

A. POST-PROCESSING

Aside from the two blocks discussed in Chapters III and IV, the final block in the subsystem created was the *Post-Processing/Video Output Subsystem* (Figure 1.) This block was used to process the outputs from both the *Optical Flow Analysis* and *Kalman Filter Tracking Implementation* blocks and verify correct behavior of said blocks.

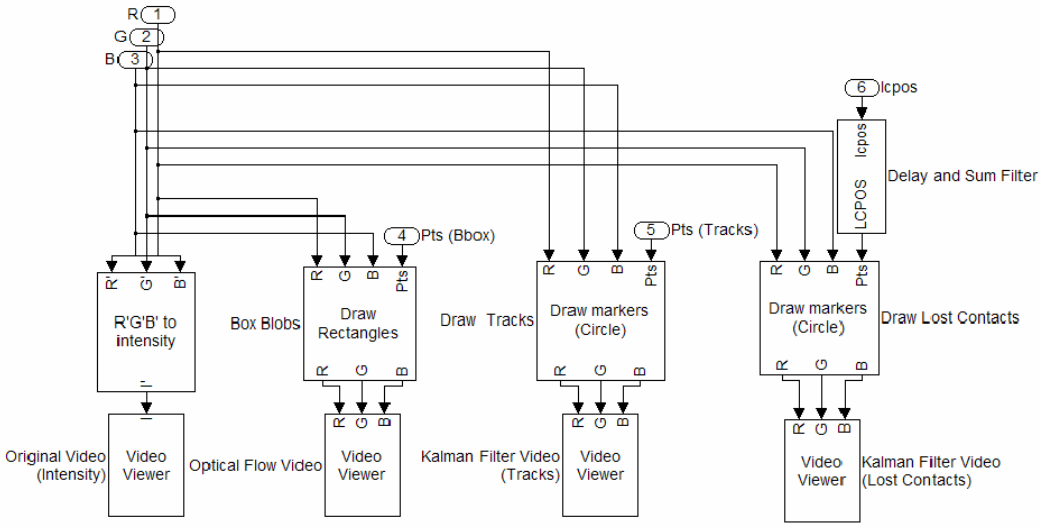

Figure 22. Simulink Block Diagram of Post-Processing Subsystem

Figure 22 presents a view of the inner workings of the *Post-Processing/Video Output Subsystem.* The first video output block is simply an output of the input video in the intensity color-space. The rest of this section will be dedicated to describing the task performed by each of the remaining video viewers.

1. Optical Flow Analysis

The second video viewer shown in Figure 22 is the *Optical Flow Video Viewer*. This video viewer takes four inputs. The first three inputs are the input video in RGB color-space. The last of these inputs is labeled *Pts (Bbox)*. This input is identical to the *BBox* output shown in Figure 4. The *Blob Analysis Block* shown in Figure 4 produces coordinates for a bounding box around every blob in each frame. The *Draw Rectangles* block in Figure 22 draws rectangles on the video frame as defined by these coordinates.

In short, the *Optical Flow Video Viewer* allows the user to ensure that the *Optical Flow Analysis* subsystem is operating properly. The user can see in real-time which contacts in the video are being sent to the *Kalman Filter Tracking Implementation* block by monitoring the *Optical Flow Video Viewer.* If bounding boxes are not consistently surrounding obvious contacts in the image, then the *Optical Flow Video Viewer* serves as an indication that the *Optical Flow Analysis* subsystem is not working properly for the data being used. On the contrary, the user can deduce that the *Optical Flow Analysis* subsystem is working as desired when contacts are consistently surrounded by bounding boxes. Figure 20 shows a sample frame output of the *Optical Flow Video Viewer* with bounding boxes superimposed on contacts in the scene.

2. Kalman Filter Tracking

The third video viewer shown in Figure 22 is the *Kalman Filter Tracking Video Viewer*. This video viewer takes four inputs as well. The first three inputs are the input video in RGB color-space. The last of these inputs is labeled *Pts (Tracks)*. This input is identical to the *xhatOut* port at the output of the *Kalman Filter Tracking Implementation*. This output is a matrix of size two by ten where each column is a position vector corresponding to either the position of a contact

or the initial conditions set at the center of the frame. The *Draw Markers* block in Figure 22 draws ten circular marks on the video frame as defined by each of these ten position vectors.

In short, the *Kalman Filter Tracking Video Viewer* allows the user to ensure that the *Kalman Filter Tracking* algorithm is operating properly. The user can see in real-time which contacts in the video are being tracked by the *Kalman Filter Tracking Implementation* block via markers drawn on the video frames. The *Kalman Filter Tracking Video Viewer* serves as an indication that the *Kalman Filter Tracking Implementation* subsystem is not working properly when markers are not consistently following obvious contacts in the image. On the contrary, the user can deduce that the *Kalman Filter Tracking Implementation* subsystem is working as desired when contacts are consistently tracked by markers.

Figure 23 shows a sample frame of the *Kalman Filter Tracking Video Viewer* output. The marker drawn at the middle of the frame (indicated by an arrow) corresponds to the initial conditions of the eight inactive filters. Although only one marker is visible, in actuality there are eight markers overlapping each other at this position. The other two markers are correctly tracking contacts in the scene.

Figure 23. Sample Frame of Kalman Filter Tracking Video Viewer

3. Lost Contact Notification

The last video viewer shown in Figure 22 is the *Lost Contact Video Viewer*. This video viewer takes four inputs as well. The first three inputs are the input video in RGB color-space. The last of these inputs is labeled *lcpos*. This input is identical to the *lcpos* port at the output of the *Kalman Filter Tracking Implementation*. This output is a position vector which takes either the value *[0,0]'* or the position of a lost contact. Given the lost contact position is only produced for the frame in which the contact is lost, some processing is necessary before it can be sent to the *Draw Markers* block. The *Delay and Sum Filter* sums the value of lcpos over the last 50 frames. It is assumed here that two contacts are not lost within two seconds (50 frames) of each other. Given when no

contact is lost the output is simply a vector of zeros, the filter allows the position of the lost contact to be sent to the *Draw Markers* block for 50 consecutive frames.

B. RESULTS

This section is devoted to discussing the results of the three major areas investigated in the *Post-Processing/Video Output* subsystem. The first sub-section describes the results of *Optical Flow Analysis*. The second sub-section describes the results of *Kalman Filter Tracking*. The third sub-section describes the results of *Lost Contact Notification.*

1. Optical Flow Analysis

Results for the *Optical Flow* Analysis portion of this thesis were subjective in nature. The goal of the *Optical Flow Analysis* subsystem was to segment the video sequence into binary frames containing blobs which could be tracked as contacts. Jacinto C. Nascimento and Jorge S. Marques [9] have presented metrics for performance evaluation of object detection algorithms. This field is very underdeveloped, and little information is available on how to monitor the performance of algorithms like the *Optical Flow Analysis* subsystem. As such, the point of the results for this subsystem was simply to provide a proof of concept. That is, the *Optical Flow Analysis* subsystem did provide a consistent stream of contacts to be tracked by the *Kalman Filter Tracking* algorithm. The process of objectively defining results of the *Optical Flow Analysis* will be left for further study.

2. Kalman Filter Tracking

In order to determine how well the *Kalman Filter* tracked contacts throughout the scene it was necessary to create a ground truth to compare against the *Kalman Filter Tracking* algorithm output. To achieve this, several artificial videos were created. These videos consist of one or more white blobs

moving through a black background. These videos simulate the video sequence found at the output of the *Optical Flow Analysis* block under ideal conditions. Results for both single contacts and multiple contacts are presented in the following sub-sections.

a. Single Contact

A single contact was tracked under three different conditions. The first condition was that of constant direction and speed. The second condition was that of constant direction and varying speed. Finally the contact was tracked under constant speed and varying direction.

For each condition, two figures are shown. The first plots the vector norm of the difference between the known position (*z)* and the Kalman Filter estimate (*xhat*) plotted against time. The second is a three-dimensional plot showing the position of the contact (*z)* as well as the position of the track (*xhat)* against time. In all three cases the first plot shows that the norm approaches zero in much less than a second. This is reflected in the fact that the second plot for all three cases shows overlapping tracks for *z* and *xhat* after a nominal amount of frames. The output from the *Kalman Filter Tracking Video Viewer* is available at www.nps.navy.mil/faculty/fargues/theses/dsemko/.

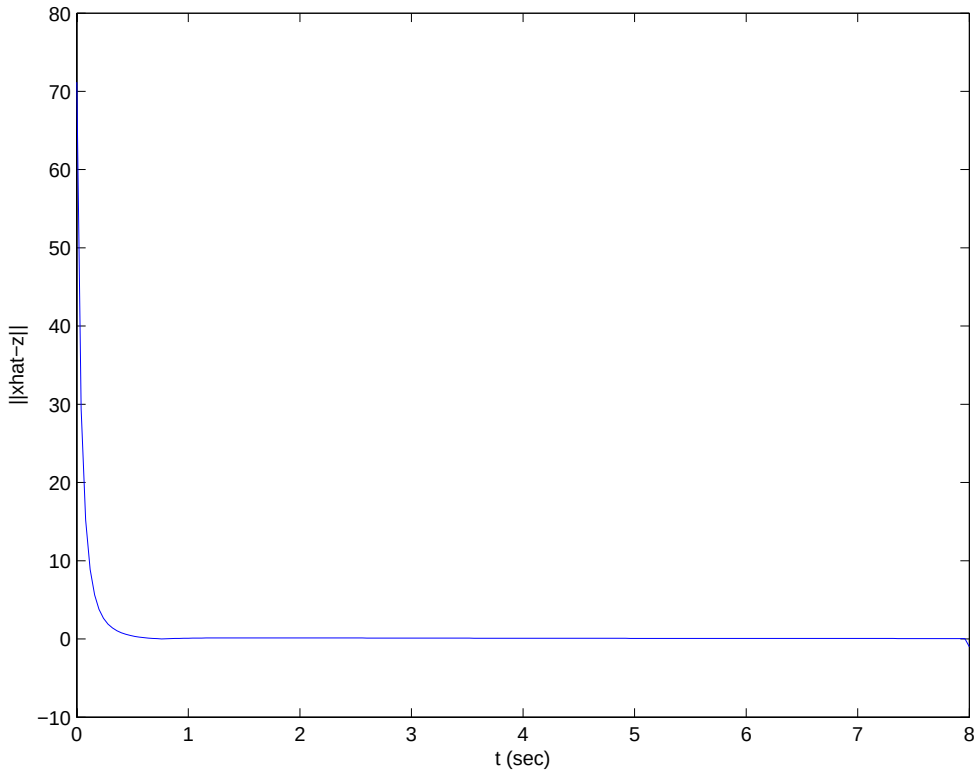

Figure 24. Vector Norm Plot for One Contact (Constant Speed and Direction)

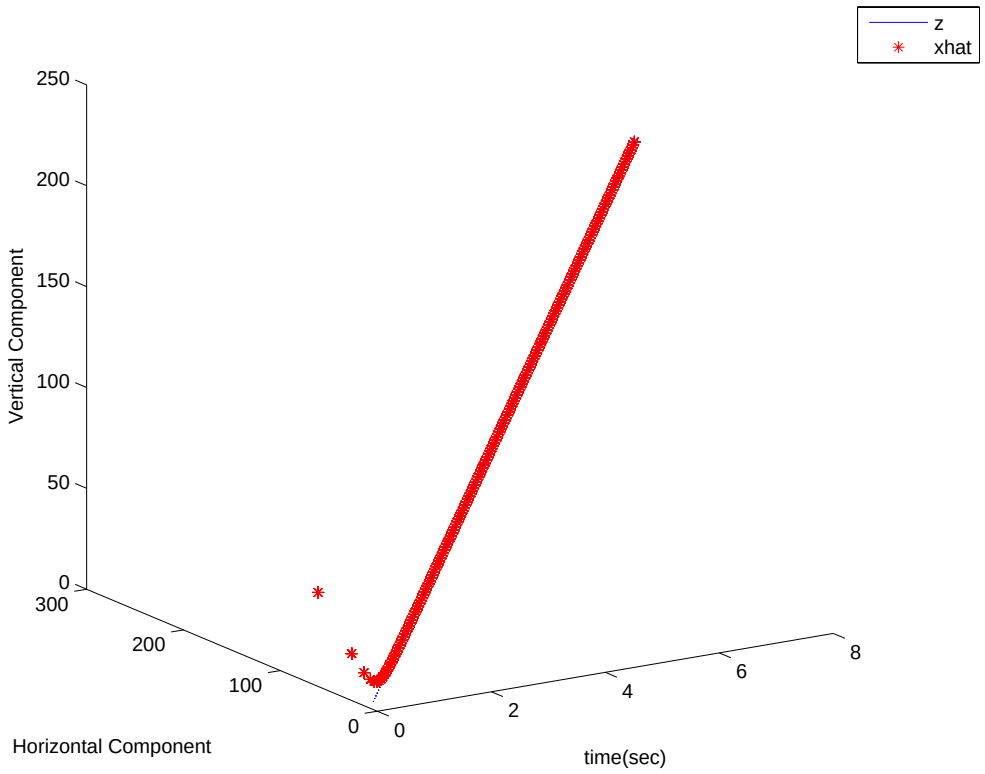

Figure 25. Position Track for One Contact (Constant Speed and Direction)

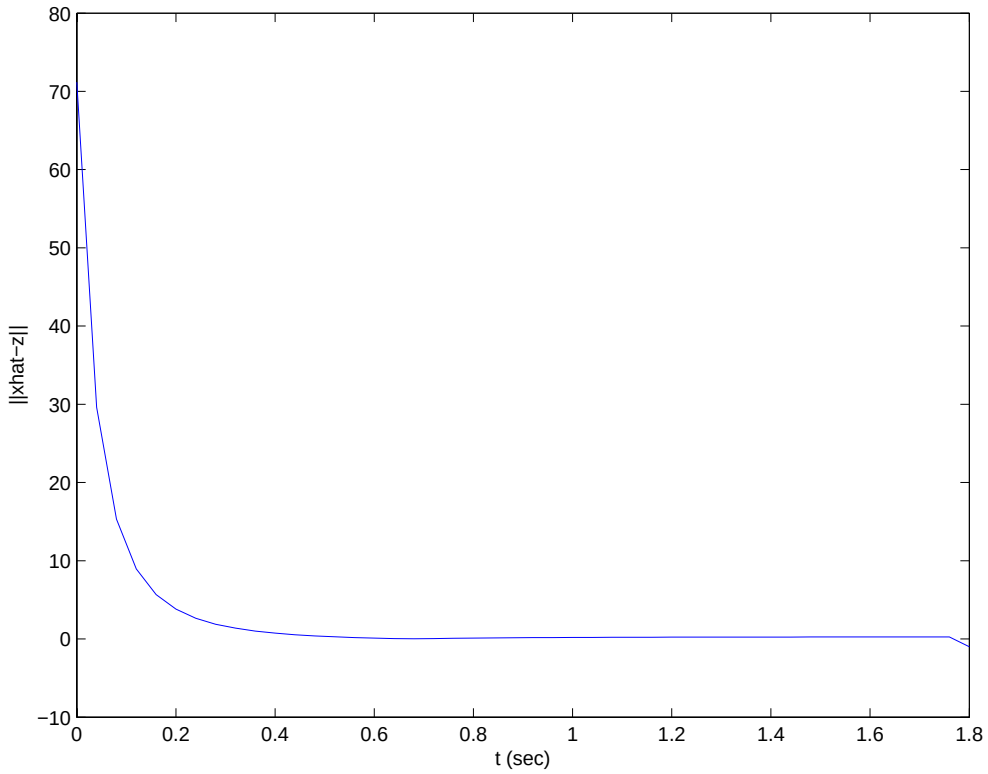

Figure 26. Vector Norm Plot for One Contact (Varying Speed and Constant Direction)

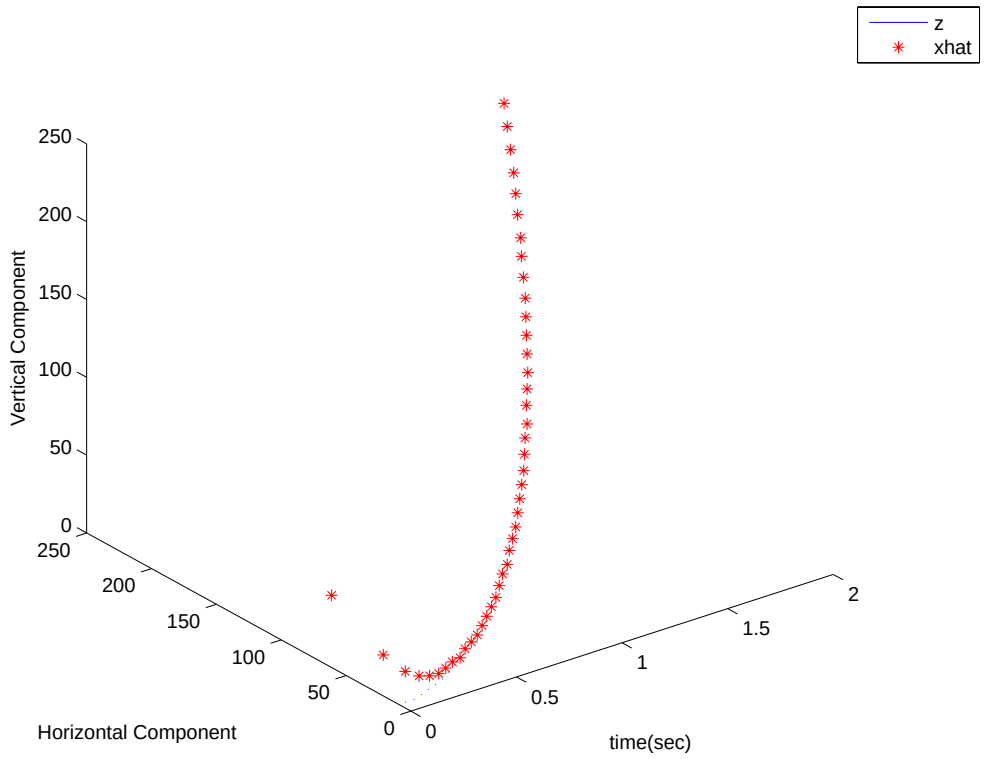

Figure 27. Position Track for One Contact (Varying Speed and Constant Direction)

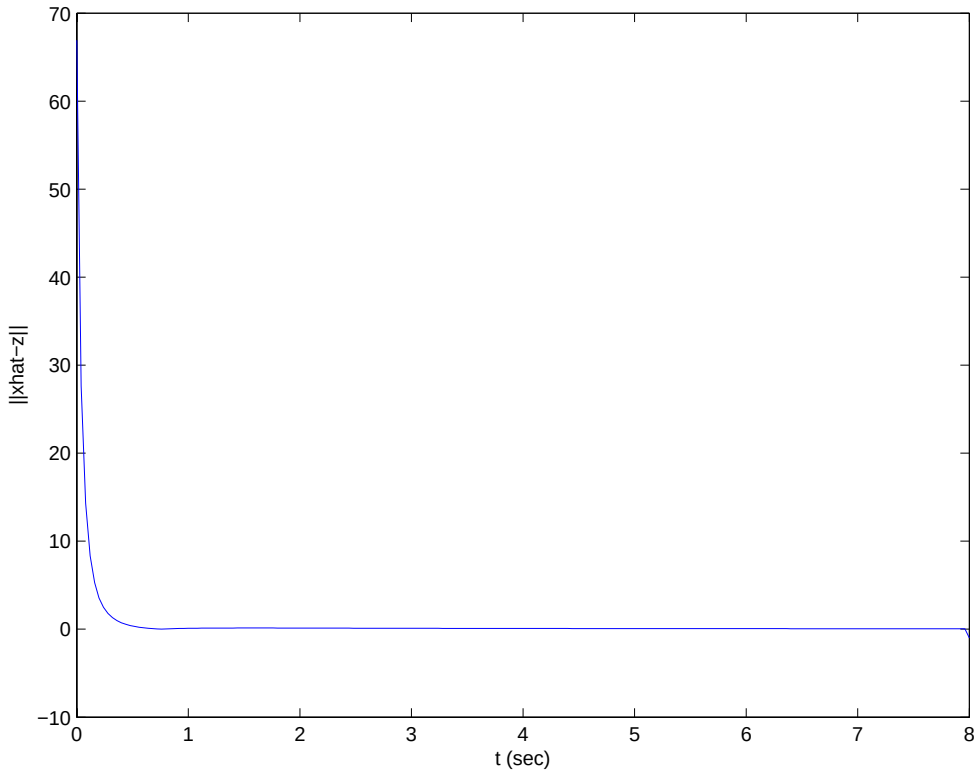

Figure 28. Vector Norm Plot for One Contact (Constant Speed and Varying Direction)

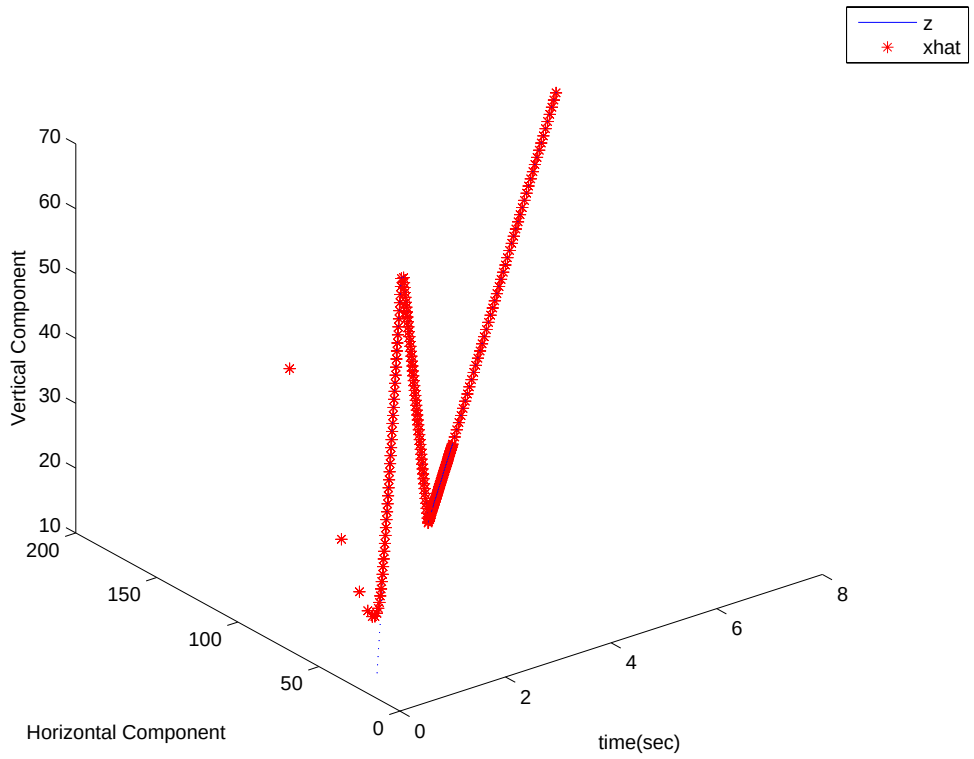

Figure 29. Position Track for One Contact (Constant Speed and Varying Direction)

b. Multiple Contacts

Three separate contacts were tracked simultaneously. One of these contacts had a constant speed and constant direction. The second contact had a constant speed and a variable direction. The third contact had a variable speed and a constant direction. Again, a plot of the vector norms of each of the known locations (z_k) and the estimated positions ($xhat_k$) are shown in Figure 30. Each subplot corresponds to the norm vector between the known location and its respective estimated position. As in the case of a single contact, the value of this norm approaches zero within a nominal number of frames. As in the case of a single contact, the position track plot shown in Figure 30 concurs with the vector norm plots as the tracks for both the known locations and their respective estimated positions overlap after a nominal amount of frames. The output from the *Kalman Filter Tracking Video Viewer* is available at www.nps.navy.mil/faculty/fargues/theses/dsemko/.

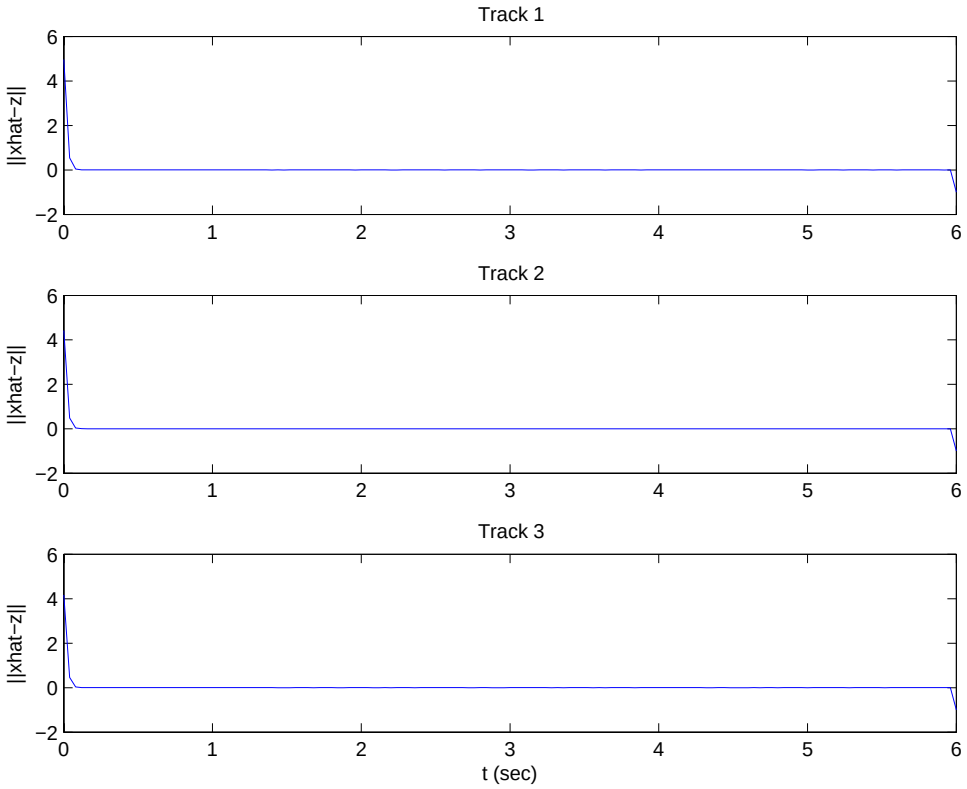

Figure 30. Vector Norm Plot for Multiple Contacts Under Various Conditions

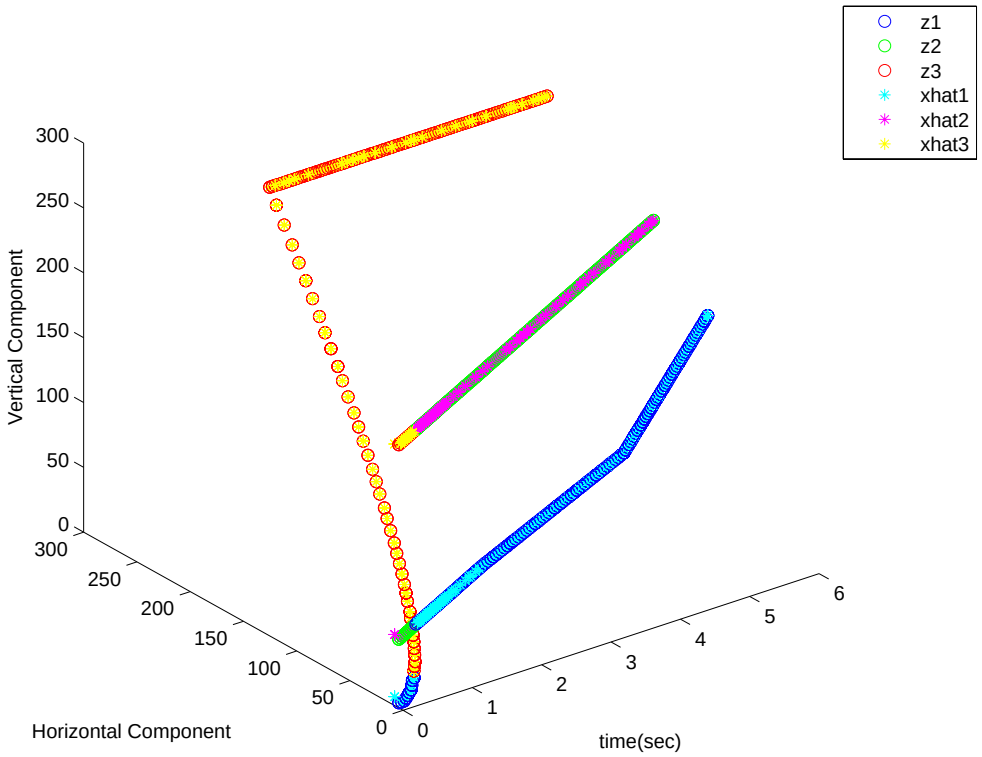

Figure 31. Position Tracks for Multiple Contacts under Various Conditions

3. Lost Contact Notification

Again, the results for the *Lost Contact Notification* behavior were very subjective in nature. Proof of concept was obtained by creating artificial data (as discussed in the previous sub-section) and visually determining whether the lost contact notification operated properly. Two videos (from the *Kalman Filter Tracking Video Viewer* and the *Lost Contact Video Viewer)* showing the correct behavior of this portion of the algorithm are available at www.nps.navy.mil/faculty/fargues/theses/dsemko/. These videos show correct tracking of a contact until it is deemed permanently lost, at which point the lost contact notification appears on the video.

4. CAVIAR Data Results

In addition to the artificial data constructed for analysis in the previous two subsections, the algorithm was used on the data from the CAVIAR study [3]. A sample video showing the behavior of the algorithm under actual conditions can be found at www.nps.navy.mil/faculty/fargues/theses/dsemko/. While, as expected, the algorithm does not perform as well as the case with artificial data, tracking of contacts is maintained somewhat consistently throughout the video.

VI. CONCLUSIONS

The problem of tracking objects in a scene hinges on two main factors: the speed to process the video in real-time and the accuracy to distinguish between contacts moving in the scene. In this thesis we have shown that through use of *Optical Flow Analysis* algorithms as well as *Kalman Filter Tracking* algorithms it is possible to track contacts through a scene in real-time and identify some basic contact behaviors.

The *Optical Flow Analysis* algorithms used in this thesis presented the largest barrier to fast and accurate operation. Because the *Optical Flow Analysis* is an iterative process to determine the best approximation to the optical flow between video frames, there is an inevitable trade-off between accuracy and speed. In order to accurately distinguish contacts moving in a scene, the computational time of the algorithm must be increased to allow for more iterations. If the required increase in processing time is too large, the algorithm cannot possibly operate in real-time.

The *Kalman Filter Tracking* algorithm implemented in this thesis was novel in its ability to process more than one contact at a time and assign the correct filter to its respective contact. Using artificial data simulating ideal output from the *Optical Flow Analysis*, the algorithm developed performed encouragingly. The algorithm was able to correctly track contacts moving under a variety of conditions and notify the user when a contacts track had been lost. Using real data, in conjunction with the *Optical Flow Analysis* algorithm, the *Kalman Filter Tracking* algorithm was able to track contacts with moderate consistency. Due to the success with artificial data, any inconsistencies in tracking can be traced to the fluctuations in performance of the *Optical Flow Analysis* algorithm.

Finally, the *Lost Contact Notification* algorithm was able to correctly identify the last known location of a contact and provide a visible alert to the user. This behavior was admittedly simple, but did indicate a possibility for further

advancement. It is conceivable that further work could lead to more complex notification systems that would further ease the workload placed on the user when viewing the surveillance video.

Future research in the area of surveillance video processing should be focused in two directions. First, due to the fluctuation in performance of the *Optical Flow Analysis* algorithm, significant work should be done to determine an objective measure of performance for this algorithm. The question of whether or not *Optical Flow* is the proper algorithm to use for segmentation should also be examined. Other algorithms may exist that could be better suited to accurate, real-time processing of video. These algorithms could provide similar outputs to be used in conjunction with the *Kalman Filter Tracking* algorithm presented in this thesis which has shown much promise.

Second, research should be focused in the area of determining contact behavior. While we have shown the ability to detect a loss of a contact, there are many more ways to classify significant behavior. For example, future research could explore areas such as the merging of contacts into groups and also the dividing of groups of contacts into separate contacts. Building off of the work in this thesis, there are a number of different ways to classify contact behavior which could further benefit the user and should be explored.

APPENDIX: KALMAN FILTER TRACKING CODE

```
%kalman_10
%The following code processes the output measurements from the Optical
%Flow Analysis block (z) and the number of contacts (num) at the output
%of the same block and performs a Kalman Filter Tracking operation on
%each measurement to determine an estimated position (xhatOut).  The
%code also monitors changes in the number of contacts and produces the
%last known location of a contact (lcpos) if a contact is lost.

function [lcpos, xhatOut]=kalman_10(z,num)

%The following variables are set as persistent in order to make them
%available frame after frame.
persistent XHAT
persistent PP
persistent lastmed
persistent numbin
persistent P
persistent index
persistent XHATtemp

%Initial conditions for various variables are set here for use throughout
%the embedded MATLAB function
if isempty(PP)
   XHAT = 0.1*ones(6,10);            %Estimated pos./vel/accel.
   XHAT(1,:)=144*ones(1,10);         %Estimated pos. (row)
   XHAT(2,:)=192*ones(1,10);         %Estimated pos. (column)
   pp = diag([9 9 25 25 49 49]);     %Initial P-covariance matrix (6x6)
   PP=[pp pp pp pp pp pp pp pp pp pp];%10 P matrices side by side
   numbin=num*ones(25,1);            %Bin tracks num over succ.frames
   lastmed=num;                      %Bin median
   P=pp;                             %Initial value for P
   index=[1:10]';                    %Index vector for filter assignment
   XHATtemp=zeros(6,10);             %Temporary estimated pos.
end

%Numbin is continually updated to include information from the current
%frame and 24 previous frames.  Median value for number of contacts (num)
%is calculated as mednum.
numbin(1:24,1)=numbin(2:25,1);
numbin(25,1)=num;
mednum=median(numbin);
```

```
%Each measurement (z) is compared to the position estimates which have
%remained persistent from last frame.  Each measurement is paired with
%its closest position estimate and the index of said estimate is recorded.
%Two “for loops” are used to ensure that no two successive measurements are
%matched with the same filter and position estimate.  The first loop
%calculates the index for the first measurement and the second calculates
%it for all subsequent measurements.
if mednum~=num
   normMIN=10e10;
   for j=1:lastmed
      normCALC=norm(z(:,1)-XHAT(1:2,j));
      if normCALC<normMIN
         index(1,1)=j;
         normMIN=normCALC-eps;
      end
   end

   for k=2:num
      normMIN=10e10;
      for j=1:lastmed
         normCALC=norm(z(:,k)-XHAT(1:2,j));
         if normCALC<normMIN && index(k-1)~=j
            index(k,1)=j;
            normMIN=normCALC-eps;
         end
      end
   end
end

%The measurements are adjusted here to account for the possibility that
%a contact was lost or gained for less than 13 frames.  Any measurement
%that is missing is replaced with the position estimate from the previous
%frame.
ztemp=XHAT(1:2,:);
for i=1:num
   ztemp(:,index(i,1))=z(:,i);
end
z=ztemp;
```

```
%If lastmed is greater than mednum, then a contact has been missing for
%13 out of the past 25 frames and the contact is deemed lost.  The index
%of that contact is determined and subsequently the position corresponding
%to that index is sent to the post-processing block.  The filter which
%corresponded to the lost contact is re-set to the initial conditions.
%*If LCindex is less than or equal to zero then there is an error in the
%index assignments and no lost contact information is sent to the post-
%processing block.
if lastmed>mednum
   sumSOME=0;
   sumALL=0;
   for i=1:mednum
      sumSOME=sumSOME+index(i,1);
   end
   for j=1:lastmed
      sumALL=sumALL+j;
   end
   LCindex=sumALL-sumSOME;
   if LCindex>0
      LCpos=XHAT(1:2,LCindex);
      for i=1:10
         XHATtemp(:,i)=XHAT(:,index(i,1));
      end
      XHAT=XHATtemp;
      XHAT(:,num+1)=[144;192; 0.1; 0.1; 0.1; 0.1];
      index=[1:10]';
   else
      LCpos=[0;0];
   end
else
   LCindex=0;
   LCpos=[0;0];
end

%lastmed is updated for use in the processing of the next frame of video.
lastmed=mednum;
```

```
%The Kalman filter is implemented based on the number of adjusted data
%points that are present at the input of the block.  Each data point is
%processed separately and filterindex is used to index the correct pos.
%estimates and covariance matrices.
for i=1:mednum
   if index(i,1)>0
      filterindex=index(i,1);
   else
      filterindex=10;
   end
      xhat=XHAT(:,filterindex);
      low=filterindex*6-5;
      P(:,1)=PP(:,low);
      P(:,2)=PP(:,low+1);
      P(:,3)=PP(:,low+2);
      P(:,4)=PP(:,low+3);
      P(:,5)=PP(:,low+4);
      P(:,6)=PP(:,low+5);

      % 1. Compute Phi, H, Q, and R
      Phi = [1 0 1 0 0.5 0; 0 1 0 1 0 0.5; 0 0 1 0 1 0; 0 0 0 1 0 1; 0 0 0 0 1 0; 0 0 0 0 0 1];
      H=zeros(2,6);
H(1:2,1:2)=diag(ones(1,2));H(1:2,3:4)=diag(ones(1,2));H(1:2,5:6)=0.5*diag(ones(1,2));
      Q =diag(0.5*ones(1,6));
      R =diag(ones(1,2));

      %2. Compute Kalman Gain and Update State Covariance
      K=P*H'*((H*P*H'+R)^-1);
      P = (eye(6,6)+K*H)*P;

      % 3. Propagate the track estimate::
      xhat = xhat+K*(z(:,i)-H*xhat);
      xhat=Phi*xhat;

      % 4. Update Covariance Matrix
      P = Phi*P*Phi'+Q;

      % 5. Refill filter bank
      XHAT(:,filterindex)=xhat;
      PP(:,low)=P(:,1);
      PP(:,low+1)=P(:,2);
      PP(:,low+2)=P(:,3);
      PP(:,low+3)=P(:,4);
      PP(:,low+4)=P(:,5);
      PP(:,low+5)=P(:,6);

end
```

```
%Updated position estimates and lost contact position (if any) are sent to
%the post-processor.
lcpos=LCpos;
xhatOut=XHAT(1:2,:);
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Wei Niu, Long Jiao, Dan Han and Yuan-Fang Wang, “Real-Time Multi-person Tracking in Video Surveillance,” Proceedings of the 4th International Conf. on Information, Communications and Signal Processing, Singapore, December 2003.

[2] Weiming Hu, Tieniu Tan, Liang Wang and Steve Maybank, “A survey on Visual Surveillance of Object Motion and Behaviors,” *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. C-34, No. 3, pp. 334-352, 2004.

[3]”CAVIAR: Context Aware Vision using Image-based Active Recognition,” http://homepages.inf.ed.ac.uk/rbf/CAVIAR/ [website], Last Accessed: June 2007.

[4] B.K.P. Horn and B.G. Schunk, "Determining optical flow." *Artificial Intelligence*, vol 17, pp. 185-203, 1981.

[5] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME - Journal of Basic Engineering*, Vol. 82, pp. 35-45, 1960.

[6] Robert Grover Brown and Patrick Y.C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*, Third edition, John Wiley & Sons, New York, 1997.

[7] Conrado Aparacio, “Implementation of a Quaternion-Based Kalman Filter for Human Body Motion Tracking using MARG Sensors,” Master’s Thesis, Naval Postgraduate School, Monterey, California, 2004.

[8] “Simulink Help,” http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/index.html?/access/helpdesk/help/toolbox/simulink/slref/embeddedmatlabfunction.html [website], Last Accessed: June 2007.

[9] Jacinto C. Nascimento and Jorge S. Marques, “Novel Metrics for Performance Evaluation of Object Detection Algorithms,” Proceedings of the 6th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, Prague, Czech Republic, 2004.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Jeffery B. Knorr
   Chairman, Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Monique P. Fargues
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

5. Roberto Cristi
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California