Fall 11-21-2018

Frequency Domain Decomposition of Digital Video Containing Multiple Moving Objects

Victor M. Stone
University of New Mexico - Main Campus

Victor Monroe Stone
Candidate

Electrical and Computer Engineering
Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Marios Pattichis, Chairperson

Christos Christodoulou

Balasubramaniam Santhanam

Constantinos Pattichis

Frequency Domain Decomposition of Digital Video Containing Multiple Moving Objects

by

VICTOR MONROE STONE

BS, Physics, Arizona State University, 1970
MS, Computer Science, California State University, Chico, 1982

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico
Albuquerque, New Mexico

December, 2018

DEDICATION

This work is dedicated to some of my staunchest supporters, my mother, Melba W. Stone, my father, Selden E. Stone Jr., and my eldest brother, Selden E. Stone, III. My only regret is that they didn't live to see its successful completion. My hope is that as they enjoy their eternal rest they can rejoice in this accomplishment along with the rest of the family still here.

It is also dedicated to my long-suffering family, who were with me all along for the ride through the rough places as well as the smooth ones.

"Soli Deo Gloria"

ACKNOWLEDGEMENTS

I wish to thank Mo Jamshidi, my first advisor, for starting me out on this journey. He believed in me and intervened to give me a chance at the start and saved the dream.

I wish to thank Marios Pattichis, my second advisor, who took me under his wing and put me on a new path when it seemed impossible to continue. His guidance enabled me to complete the work successfully and he saw me through to the end. He believed in me, and it is appreciated.

I also wish to thank my family, who have had to put up with me being in school most of their lives. Their support was essential to whatever success I may have achieved. They never doubted, always believing in me.

Of course, I must thank my wife Virginia, whose love and faith in me never wavered, even when her patience did. She was faithful to the promise she made to my father thirty years ago to "get me through it." Well, sweetheart, you did it.

Frequency Domain Decomposition of Digital Video Containing Multiple Moving Objects

by

Victor Monroe Stone

BS, Physics, Arizona State University, 1970
MS, Computer Science, California State University, Chico, 1982
PhD, Engineering, University of New Mexico, 2018

ABSTRACT

Motion estimation has been dominated by time domain methods such as block matching and optical flow. However, these methods have problems with multiple moving objects in the video scene, moving backgrounds, noise, and fractional pixel/frame motion. This dissertation proposes a frequency domain method (FDM) that solves these problems. The methodology introduced here addresses multiple moving objects, with or without a moving background, 3-D frequency domain decomposition of digital video as the sum of locally translational (or, in the case of background, a globally translational motion), with high noise rejection. Additionally, via a version of the chirp-Z, fractional pixel/frame motion detection and quantification is accomplished. Furthermore, images of particular moving objects can be extracted and reconstructed from the frequency domain. Finally, this method can be integrated into a larger system to support motion analysis.

The method presented here has been tested with synthetic data, realistic, high fidelity simulations, and actual data from established video archives to verify the claims made for the method, all presented here. In addition, a convincing comparison with an up-and-coming spatial domain method, incremental principal component pursuit (iPCP), is presented, where the FDM performs markedly better than its competition.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

There is much interest in identifying, quantifying, and extracting objects within video sequences, as evidenced by the amount of space allocated to the subject in textbooks and papers on video processing [1, 2, 3, 4, 5, 6]. This subject is important to video surveillance, military and civil aerial reconnaissance, and security. The crop of methods currently in vogue each have their own strengths, but they all share certain weaknesses – difficulty with moving backgrounds, platform motion, fuzzy focus, issues with noise, smooth surface texture, scene clutter, and high computational costs.

1.1 Motivation

Current methods do not fare well when confronted with background motion, jitter in the platform, focus issues, and optical noise. Block matching and optical flow [3] are very popular methods, and they also have similar issues. To illustrate this, two examples are presented in way of explanation. A 32 frame video segment was created using a 64x64 pixel frame. The blob was moved 1 pixel to from left to right. It was first submitted to a classical block matching algorithm which had been modified so that lines proportional to the magnitude would be displayed and exaggerated so they would be visible. It was then presented to a Horn-Shunck optical flow algorithm [7], modified in the same way so the motion vectors are visible. The results are shown in Figures 1.1 and 1.2, below. This examples show both block matching and optical flow failing to properly process the blob beyond the edges. The failure is due to the lack of features inside the blob.

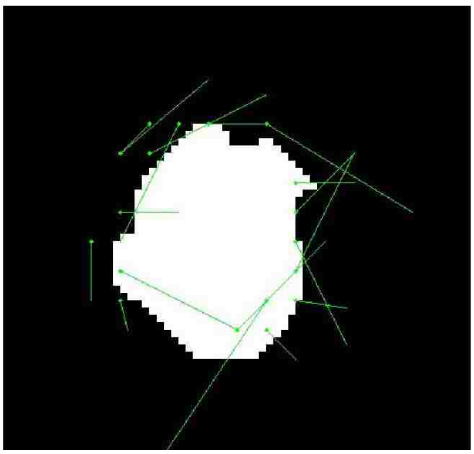

Figure 1.1 Block matching failing to estimate the correct motion

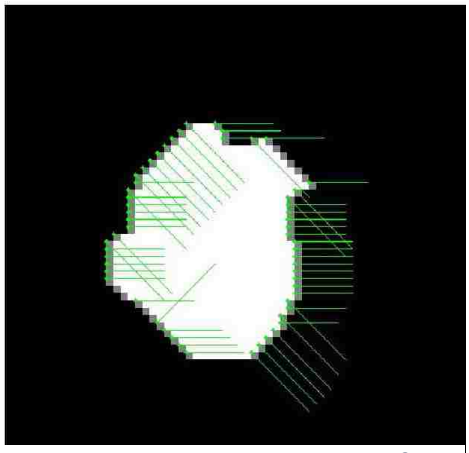

Figure 1.2 Horn-Shunck optical flow failing to estimate the correct motion with default parameters (MatLab implementation)

As can be seen, the vectors from the block matching don't seem to have detected the motion properly. Optical flow did better, but there are still several stray estimates. The other issues mentioned are equally troublesome.

1.2 Related Research

Motion detection and quantification fall into two classes – spatial domain algorithms and frequency domain algorithms. The vast majority of extant methods are spatial domain algorithms, including everything from background subtraction to analytical methods like PCA. Frequency domain methods are not at all well represented in the literature, indicating the degree of difficulty encountered when working in that domain.

1.2.1 Spatial Domain Methods

Existing methods of motion estimation have been dominated by spatial domain techniques, as evident from the references previously cited above [1, 2, 3], most notably block matching and optical flow. Some techniques use analytical techniques, such as principal component analysis (PCA), as seen in [8], but it remains a spatial domain

technique. When multiple objects are expected, these methods often require either that the scene be subdivided so as to isolate the different objects from each other or to use some kind of classification technique to identify multiple objects. Generally, these methods assume that the short video segments along with the typically short sampling interval provided by the frame rate allows the assumption of linear motion without rotation and for uniform illumination and texture. All methods have shortcomings. As block matching and optical flow are probably the most popular methods, they will be examined further here. As PCA methods are gaining popularity they will also be discussed.

Block matching starts by breaking up the reference frame into small blocks (on the order of 8x8 or 16x16 pixels), which are then compared to the analogous blocks in the next frame. The method assumes that the motions are simply linear translations of a few pixels per frame in x and y; although just finding the translation of the reference block that matches the target block is usually sufficient, adding cross-correlation to the search makes it very robust. This also is its weak point. First, the exhaustive search of the target frame is computationally expensive. Fractional pixel motions can be part of the search, but it is even more computationally expensive, so sub-pixel resolution is usually not done. Block size is also an issue, as the smaller the block size, the more computation required. If the block is too big, then it will include background; if the background is changing as well, the technique will find it difficult to identify what is going on with the mixed blocks. This problem also occurs when two moving objects are passing by each other. The technique also depends on the texture of the object. If the object is uniform, and the blocks are small, the technique will detect the edges, but otherwise fail. The

estimate will be noisy, as the boundary of the estimate will be blocky. Finally, the technique is not very robust in the face of noise; it depends on the image being unchanged, except in location, frame to frame, and noise changes the texture of each block in a random manner, making comparisons more difficult.

Optical compares reference pixels to target pixels, one to one, rather than block to block. Optical flow uses statistical and optimization techniques to determine the motion of the pixel field, from which the motion within the video is deduced. The assumption here is that the intensity is consistent. Sub-pixel motion is easier to estimate than with block matching. However, optical flow is not very robust in the face of noise for the same reasons as block matching. Worse, the video sequence can't be denoised, because denoising alters the intensity and texture of the video frame by frame, violating the assumption of consistent illumination intensity. As computationally expensive as block matching is, optical flow can be even more costly because it works at the pixel level, not with larger blocks. Optical flow also suffers from the same issue as block matching when confronted by a uniform object, not having a texture to differentiate one pixel from another, even though motion estimation will likely work at the edges.

Before moving on, it must be acknowledged that PCA based methods must be mentioned. The current popular method is called "Incremental Principal Component Pursuit via Alternating Minimization" (iPCP) [9, 10] which uses principal component analysis to model the foreground (i.e. the moving object) as well as the background. As an analytic, spatial domain method, it eschews the frequency domain, but it is a very effective technique. The main idea is that the slow or static pixels are assumed to be background and the fast pixels are assumed to be in the foreground and therefore the moving object.

It uses this information to inform and update models of the foreground and background in real time and thus separate the moving object from the background. This implies a restriction to a single moving object. The method doesn't explicitly provide output products other than video output showing the foreground and background, but the data is present within the method. Teasing out specific objects is problematic. Code was obtained from the author's web site and slightly modified to provide the data to allow for a direct comparison detailed in Chapter 5.

1.2.2 Frequency Domain Methods

The obvious alternative to spatial-domain methods is to use the frequency domain methods. However, it would appear that the alternative path is more daunting, based on the lack of papers presenting methods based on the frequency domain. It is perhaps the perceived difficulty involved in using such techniques that discourages researchers from trying.

There are only three quality papers on frequency domain methods worth mentioning. They are examined here. The papers present one of three approaches, two of which address the unique properties of the 3-D spectrum where the spectrum of a moving object in a video sequence will reside on a plane skewed at angles determined by its velocity. The rarity of reported research speaks to the difficulty of extracting motion estimates from the spectrum.

One of the earliest examples of the use of frequency domain methods is reported by Kojima, Sakurai, and Kishigami [11]. It addresses the tilted plane property of the spectrum by using a frequency domain filter bank tuned to the estimated direction of motion. Here, the notion that a direct search of the spectral volume would be too difficult

computationally is expressed. Simulations using one and two moving objects were reported, as well as results with a video of a moving car on a fixed background. The method depends on detecting the lines of intersection in the 0^{th} spatial frequency plane and two other spatial frequency planes to establish a general direction and magnitude of the moving object to set the initial parameters of the filter bank. When there are multiple objects, each tilted plane must be deduced and a filter bank must be computed to produce the estimate of motion for each object. The difficulty here is that the resolution of the estimate is limited to the number of filters in the bank. Also, there is the computational cost of discerning the initial orientation of each plane and then separating them from each other. Noise rejection was cited as a topic of future work.

This paper is a good start, but as they use a hybrid method based on a set of filters, its resolution limited by the size of the set. Their method can be overwhelmed by background. They also have no demonstrated robustness in the presence of noise. This paper's method was likely dictated by the computing power available in 1993. The method introduced in this dissertation is limited only by the resolution of the transform used. The proposed method will also handle moving backgrounds.

The second method, reported by Briassouli and Ahuja [12], uses phase correlation between individual 2-D frequency planes and least-squares model fitting to detect and track motion as part of a hybrid method. The results reported include both image sequences created by combining various real images and actual video sequences were used. All examples appear to have had a static background. Some examples had two moving objects. Velocity estimates, robustness in the presence of occlusion, and the ability to extract images of the moving objects were demonstrated using synthetic and

real video sequences. The ability to overcome partial occlusion and to detect and measure rotation was reported. The authors discussed noise issues, but it appears that they generally assumed that the videos would be denoised before any other processing was done.

Briassouli and Ahuja method nibbles around the edges of the problem by using the fast Fourier transform (FFT) to inform a least squares/correlation motion model fitting technique rather than doing a head-on estimate. The reported method results in some sensitivity to noise and background. To their credit, the authors deal with rotation; this is more difficult than just finding the object and assuming steady-state velocity with no rotation or zoom. Their method may inform future work on the method presented here to extend it to deal with rotation, as well. It is likely that this work was also limited by the computing power available in 2008, as well. The method which is the subject of this dissertation essentially directly deduce the orientation of the plane, and thus obtains a superior estimate, unencumbered by the need to denoise or remove the background.

The third method, offered by Alexiadis and Sergiadis [6], ultimately uses the tilt of the moving object plane in the motion estimate, but takes a distinctly different tack to identify them, in that the researchers use a hyper complex Fourier representation of color video coupled with an iterative, fuzzy optimization based method to detect and quantify moving objects within the video. Results were presented using both synthetic sequences assembled from still images and actual video. Sequences with static and with moving backgrounds were included as examples. This method was used to decompose the images and separate the moving objects from the background. The results reported are obtained at the expense of considerable computational cost, but are still impressive. They

were able to deal with longer sequences where acceleration and zoom were present. Reversibility of the image decomposition was not demonstrated, however, and no results with noise were presented. Noise rejection was to be the theme of future work.

As impressive as their results are, the authors of paper three achieve their result at the cost of an algorithmically complex method that uses the 3-D hyper complex spectrum to inform the fuzzy clustering estimator. They do have the advantage of being able to deal with acceleration and zoom; the methodology used in this paper may point to a way deal effectively with both by incorporating elements of this paper into the method presented in this dissertation. It would seem that an algorithmically simpler direct estimator using optimization is more appealing, as provided by the method presented in this dissertation. The method presented in this dissertation also has demonstrated robustness in the presence of noise, the ability to effectively use expanded frequency resolution to improve the estimate, and the ability to perform a "perfect" decomposition that is reversible without loss.

1.3 Thesis Statement

This dissertation presents a novel and elegantly simple frequency domain based method for decomposition of digital video and analyzing translational motions mechanized by simple optimization as good as, if not superior to current spatial domain methods. It is effective over a wide range of resolutions, including fractional pixel/frame accuracy. Examples will be presented using both integral fast Fourier transform (FFT) based 3-D spectra and enhanced, fractional pixel resolution 3-D spectra based on the chirp-Z transform. It will be shown that the method is robust in the face of significant levels of

noise. It will be shown that a variant of the core algorithm can be used to extract and reconstruct of objects (or even the background) based on its corresponding motion. It will be shown that the method can be used in a practical setting by offering a synthetic object tracking application. Finally, a comparison of the tracking accuracy of the frequency domain based tracking application and an application based on the up-and-coming incremental principal component pursuit method.

1.4 Contributions

The contributions presented in this dissertation include:

- Development of an elegantly simple algorithm that reliably identifies and quantifies one or more moving objects in a video segment using 3-D Fourier transforms and a straightforward optimization methods.
- 3-D frequency domain decomposition of digital video as the sum of locally translational motions.
- The algorithm has been extended to provide for fractional pixel/frame resolution using a chirp-Z transform.
- Demonstrated strong noise rejection.
- Development of a method to extract and reconstruct the image of the moving object using a variant of the algorithm.
- Development of a virtual tracking method to showcase the practicality of the algorithm.

1.5 Dissertation overview

This section contains a brief description of the all the work that is presented in this dissertation.

- Chapter 2 presents the details of the core algorithm and how it works, along with directly related methods.
- Chapter 3 presents details the properties inherent in the algorithm – robust noise rejection and the ability to deal with multiple objects, and the related methods of image extraction and tracking.
- Chapter 4 presents results of trials with real data,
- Chapter 5 presents results of a direct comparison of the accuracy of the tracking algorithm versus tracking by the iPCP method.
- Chapter 6 gives a summary and possibilities for future work.

Chapter 2

The Core Algorithm, the Foundation of a Family of Methods

The core algorithm opens the door to several interesting methods. Without the core, the others would be impossible. One of the daughter methods, image extraction, is so tied to the basic concept that it will be presented in this chapter, as well. The basic algorithm is an integer based method; however, a related method using a chirp-Z transform to provide fractional pixel/frame resolution will also be covered. It is virtually identical, except for the use of the chirp-Z instead of the DFT.

2.1 Frequency Domain Representation of Translation Motion

The development of the 3-D Fourier transform presented here is loose paraphrase of the discussion found in Wang, Ostermann, and Zhang [13]. For a more complete treatment, please refer to a multidimensional signal processing book such as [14], to which Wang, Ostermann, and Zhang themselves refer. An object within a video frame can be identified by the coordinates of its pixels. The coordinates of a given pixel are going to be given in terms of x, y and f, for its (x, y) position within the frame f. For the continuous case, which is developed below, "f" is replaced by "t" for time, as the continuous case ignores the fact that a video stream is made up of discrete frames without loss of generality. The position of the entire object can be discussed in terms of the position of a selected pixel within the object. For the purposes of a transform, the individual object is not a concern, but the entire frame, and, indeed, the entire video sequence, is considered. So it is obvious that we are dealing with three levels of abstraction: the pixel, the object, and the frame. Fortunately, anything deduced about an

individual pixel also applies to the object and the entire frame itself. With a moving object, or even a moving frame as with background motion, the pattern changes in a way determined by the type of motion and captured in the spectrum, as will be shown.

Consider the case of a single moving object on a neutral background. Let the initial pixel pattern of the object be initially O_0(x, y, t). The object is moving at constant linear velocity $\boldsymbol{v}$ whose components are v_x and v_y, respectively. Assume uniform illumination. These assumptions simplify the discussion again without loss of generality.

Starting with the classical Fourier integral, the form for a 3-D video segment *o*(x, y, t) is

$$O(f_x, f_y, f_t) = \iiint o(x,y,t)\exp\left(-j2\pi\left(f_x x, f_y y, f_t t\right)\right)dxdydt \ . \tag{2.1}$$

Here, $O(f_x, f_y, f_t)$ is the Fourier transform of $o(x,y,t)$, and f_x, f_y and f_t are the frequency components. To show the most salient property to be noted, it is necessary to relate the transform back to a reference frame, which here is the initial frame. The position of the scene, in terms of the initial position of the object, is

$$o(x,y,t) = o_0\left(x - v_x t, y - v_y t\right) \tag{2.2}$$

This initially seems contrary to common sense, but it is a necessary tactic to make the derivation show what it must show. As can be seen, $x\text{-}v_x t = x_0$ and $y\text{-}v_y t = y_0$ so that the object's pattern is translated back to the initial position. Substitute equation 2.2 into equation 2.1 to get:

$$O(f_x, f_y, f_t) = \iiint o_0\left(x - v_x t, y - v_y t\right)\exp\left(-j2\pi\left(f_x\left(x - v_x t\right), f_y\left(y - v_y t\right), f_t t\right)\right)dxdydt \tag{2.3}$$

Let $x_0 = x - v_x t, y_0 = y - v_y t \Rightarrow dx = dx_0, dy = dy_0$. Thus, $x = x_0 + v_x t, y = y_0 + v_y t$. Substituting, we obtain

$$O(f_x, f_y, f_t) = \iint o_0(x_0, y_0) \exp\left(-j2\pi\left(f_x(x_0 + v_x t), f_y(y_0 + v_y t)\right)\right) dx_0 dy_0$$

$$\bullet \int \exp\left(-j2\pi\left(f_x v_x + f_y v_y + f_t\right)t\right) dt \ . \tag{2.4}$$

The first factor in equation 2.4 (the double integral) is, in fact, the 2-D Fourier transform of the initial frame. Using the substitution is what makes this possible. Equation 2.4 becomes

$$O(f_x, f_y, f_t) = O_0(f_x, f_y) \int \exp\left(-j2\pi\left(f_x v_x + f_y v_y + f_t\right)t\right) dt \tag{2.5}$$

where O_0 is the 2-D transform of the initial frame, as stated above. The last step involves an identity involving the delta function. Since

$$\int_{R^K} \exp\left(j2\pi f_0^T x\right) dx = \delta(f_0), \tag{2.6}$$

equation 2.5 becomes:

$$O(f_x, f_y, f_t) = O_0(f_x, f_y)\delta\left(f_x v_x + f_y v_y + f_t\right) \tag{2.7}$$

The implication of this is that temporal frequencies will only exist where $f_t = - f_x v_x - f_y v_y = 0$. This causes the 2-D transform plane of the initial scene to be skewed at an angle through the spectral volume such that the plane is normal to the vector (v_x, v_y, 1). This phenomenon is illustrated in Figure 2.1.

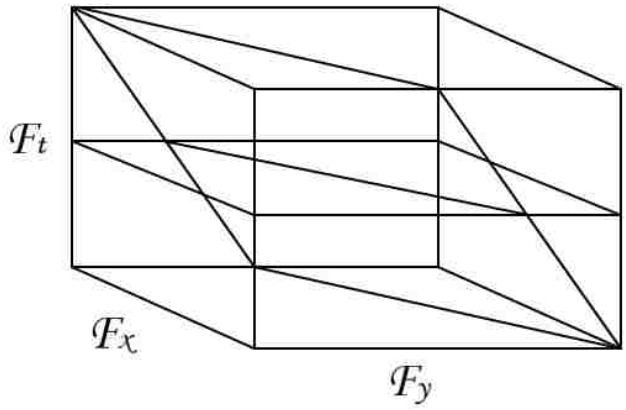

Figure 2.1 Spectral volume showing skewed plane from moving object

Ultimately, video is digital and discrete, not continuous, as the picture is presented as discrete frames, no matter how continuous it appears to the viewer. Digital video is even more so, as each pixel on a given line is distinct from its neighbors, more so than the modulated amplitude trace used in traditional analog television. The duration of the video is also finite. This leads directly to the notion that a video signal is, of necessity, cyclical, as far as the transform is concerned. Every frame has finite boundaries and, as such, forces the transform to consider it a cyclical event, even if though it really isn't from frame to frame. This is actually good news. The extension to discrete is supposedly intuitive, but it is not straight-forward; fortunately, it can ultimately be bypassed, as the discrete 3-D Fourier transform can be formed by successive application of a 1-D FFT

2.2 The Core Algorithm

The equation for the plane tilt is equation 2.7. This plane will have more spectral intensity on it than any other potential planes through the volume, barring other moving objects in the video segment. If one can project the plane through the spectral volume and then sum the intensity found on that plane, he will end up with the total intensity

contributed to the spectrum by the object. The intensity can be used to identify the moving object's plane, and, thus, identify the velocity of the moving object.

Here is the challenging part of the procedure. To find the plane of the moving object, every possible plane within the spectral volume must be examined, which means that every possible velocity pair must be examined. Even though there is no theoretical limit on the size of the video, either in duration or frame size, as a practical matter the velocity pairs to be examined must be bounded heuristically along with the length of the video clip to keep the problem tractable. For instance, most of the objects which were the subjects of this research moved at less than 5 pixels/frame, very few moving faster than that because they couldn't move that far in $1/30^{th}$ of a second, defined by the 30 frame/second video frame rate. Other frame rates and other applications could require a different set of heuristics. In any case, it would be reasonable to assume that a plane with a moving object would hold more intensity than planes with only noise.

Every possible velocity pair (v_x, v_y) defines a unique plane will be normal to the vector (v_x, v_y, 1). It doesn't matter whether or not it holds the spectrum of a moving object. This property allows one to examine that plane for spectral values in the following manner, given a certain amount of setup. The necessary set up starts with remembering that an FFT of N spatial points will generate N corresponding spectral points, and is generally shifted so that the values are arranged from frequency $-\pi$ to $(\pi - 2\pi/N)$. The reason for stopping one value short of π is due to the mechanics of the FFT. To access the spectral address (ω_x, ω_y, ω_p) corresponding to the position (x, y, p) in plain indices in the spectral volume, one creates a scaling vector *S* of length N+1 that will translate the

plain index to the spectral volume's index value ω. The scaling vector is filled with N+1 evenly spaced values from $-\pi$ to π.

Recalling that to determine if the spectral point in question is on the plane defined by the velocity pair of the search, one simply computes the dot product between the vector between the origin and the point to be tested and $(v_x, v_y, 1)$. If the dot product is 0 or less than some threshold ε, that point in the volume is considered on the plane associated with (v_x, v_y). A logical data structure mimicking the spectral volume (that is, the same dimensions) is used to record the ones and zeros. It stores the mask used to remove the values not on the plane. At the end of the exhaustive but bounded search, the logical data structure holds a plane of ones representing the plane corresponding to the velocity vector being tested. The mask is then used to select only the values on the putative plane of motion; the intensity values selected are then summed and stored in a 2-D array, the indices of which represent the velocity vector pairs within the bounds. This intensity map is for all intents and purposes a map of motion within the volume, because the any motion corresponding to that velocity shows up there. If there is no motion, the sum of the intensities will be low; if there is, the sum will be higher. Thus, where motion exists, there will be a peak in the map at the velocity of the moving object. Searching for the peak(s) in the map requires an optimization technique which allows the data to inform the observer of the presence of motion and its magnitude and direction. We will next describe the steps mathematically.

The 3-D FFT of the video segment is given by Equation 2.8 below:

$$\tilde{I}\left(\omega_x,\omega_y,\omega_t\right)=\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}\sum_{p=0}^{P-1}I\left(r,c,p\right)\exp\left(\left(-j2\pi r\omega_x/R\right)+\left(-j2\pi c\omega_y/C\right)+\left(-j2\pi p\omega_p/p\right)\right)$$

where the image size is R rows by C columns by P frames.

Let $\Omega=\left(\omega_x,\omega_y,\omega_t\right)$ be an element of a spectral space. The integer search space will be restricted to:

$$0\le\omega_x\le R-1,0\le\omega_y\le C-1,0\le\omega_t\le P-1\ . \tag{2.9}$$

In the 3-D frequency plane, the set of spectral frequencies associated with the motion are given by the 3-D discrete-space plane approximation given by:

$$S\left(v_x,v_y\right)=\{\Omega\,|\,\Omega\bullet\left(v_x,v_y,1\right)\le\varepsilon\} \tag{2.10}$$

which makes the motion map be given by the sum of spectral magnitude values over the plane given by:

$$M\left(v_x,v_y\right)=\sum\nolimits_{\Omega\in S\left(v_x,v_y\right)}|\,\tilde{I}\left(\Omega\right)|\ . \tag{2.11}$$

To detect the peak(s), we need to consider the local maxima given by:

$$\max\nolimits_{v_x,v_y}M\left(v_x,v_y\right)\quad . \tag{2.12}$$

There are some practical issues to that have to be accounted for. An exhaustive search of any kind on any sort of sizable data structure is costly. In this case, a quick complexity analysis highlights the problem. The cost of the 3D FFT comes from the need to compute along the rows, columns, and then frames through time. For an NxNxP video volume, the computational complexity of the search is $O(PN^2\log(N))$. The overall

complexity is dominated by the search procedure. The cost of the core search is contingent on how large the spectral volume is and how extensive the velocity band is. There is one dot product/compare/store for each point in the volume for each possible velocity pair. This speaks to the necessity of limiting the range of the velocities examined. As it is, continue with the assumption that the frame size is N by N over P frames. Also assume that the range of the velocities is minus v to plus v in both directions. To search every velocity pair, that would require an examination of N^2P points by $(2v+1)^2$ velocity pairs. For simplicity, let the velocity factor be V^2. Then. In asymptotic notation, the search is $O(V^2N^2P)$. A calculation of this order is daunting, at best. Therefore, ways to speed up the computations were sought.

Three practical steps are the first resort. First, process the video in gray scale. Color would require three times the processing, and there is nothing to be gained by processing in color. Second, the size of the video must be limited. Cropping the frame size provides a virtual digital zoom, making the object of interest larger and allowing it to make a larger impact in the spectrum. Likewise, limiting the length of the video segment reinforces the assumption of linearity. Third, limiting the range of the velocities being examined as mentioned above further assists in reducing the load. The first algorithmic speedup applied is obtained by observing that the spectrum (the magnitude of the spectrum is used for this process) is symmetrical. In this case, the negative temporal half of the volume does not need to be used; the result obtained will be the same. This cuts the computational burden in half. That's some help, but more can be done. Next, it is must be recognized that the origin itself is not interesting – the only value there is "DC offset", a bias often called in video cameras the pedestal. It is of no interest, but

including it would create a huge spike at the origin which would obscure any motion nearby, so it is ignored. Likewise, values on the 0^{th} temporal plane are of no interest, because that is where spectrum of non-moving objects (such as a static background) reside. By skipping over the 0^{th} plane and the origins of the other planes, the computational burden is cut down a bit more, but it's still $O(V^2N^2P)$. The answer ultimately became the availability of multicore processors. Using a multiprocessor approach to this otherwise daunting problem helped greatly. The core algorithm is well suited for a multiprocessor approach; parallelizing it and compiling the core algorithm into a MEX routine ultimately yielded a 360+ times speedup for one example. The actual speedup will vary with the size of the video.

In the early stages of this research, simple sequences of moving blocks were generated to calibrate the core algorithm and insure its proper functioning. Using one of the simple examples should illustrate the process. Figure 2.2a-h shows a simple 2x2 block on an 8x8 pixel frame moving from left to right diagonally down, 1 pixel/frame right in x and 1 pixel/frame down in y. The motion map in Figure 2.3 clearly shows the motion peak at (1, -1). For illustrative purposes, an isometric rendering of the motion map is presented as Figure 2.4 The pseudocode for the program implementing the algorighm is presented in Figure 2.5. It should be pointed out that this includes the keyword "parfor," which instructs MatLab to run this loop on multiple processors in parallel. Turning it into a MEX file changes nothing in the MatLab code, except for a few lines defining the passed parameters to the C compiler.

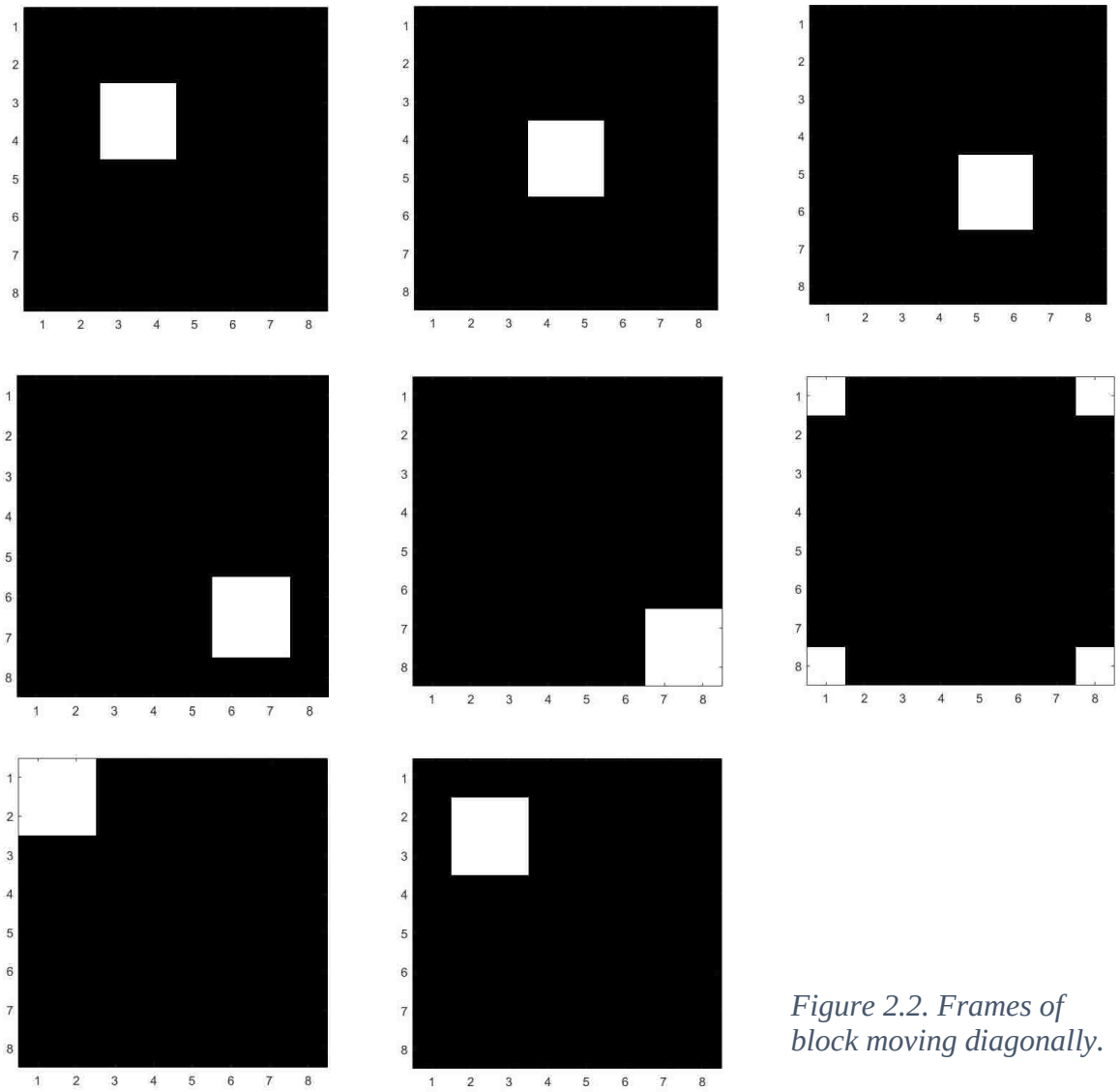

Figure 2.2. Frames of block moving diagonally.

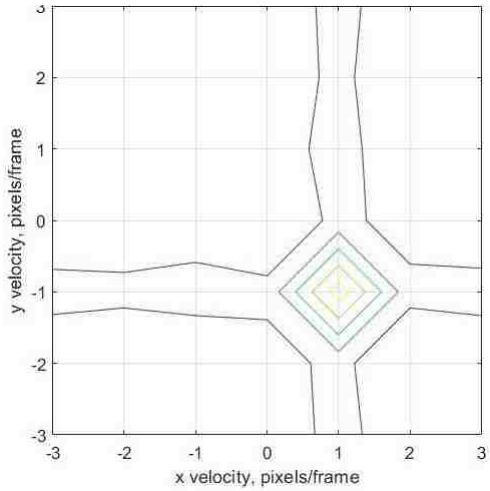

Figure 2.3 Motion map for diagonally moving block

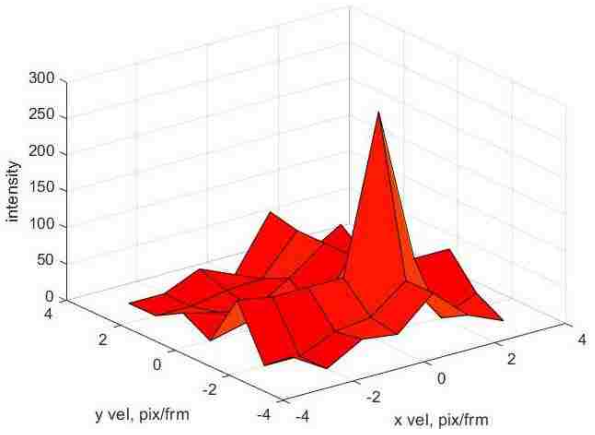

Figure 2.4 Isometric view of motion map.

Figure 2.5 Pseudocode for the latest version of the motion mapping routine

```
Motion Map
Inputs: Spectrum, range, resolution;
Output: Map

Extent = range*resolution       //determine size of map
Map = zeros (Extent ,Extent)    //allocate map
vx_max = vy_max = Extent
vx_min = vy_min = -Extent
Thresh = constant               //set the threshold as specified; it's "hardwired"
Size(Spectrum) = (M,N,P)        //get size of spectral volume
halfP = floor(P/2)              // set up for the positive half spectral volume
halfSpec = Spectrum(M, N, halfP)    // this makes the positive spectrum volume
        Probe = zeros (M, N ,halfP)    //preallocate probe volume

// Build scaling vectors
X = {N+1 values from -pi to pi}
Y = {M+1 values from -pi to pi}
Z = {halfP+1 values from 0 to pi}

//the main loop – parfor means "parallel for" to allow loops to be run in parallel
parfor Vx = vx_min to vx_max      // test each velocity vector
  for Vy = vy_min to vy_max
    if (Vx == 0) and (Vy == 0)       //skip the origin
      Map(0,0) = 0
    else
      zero Probe            // must be done each time
      for p = 2 to halfP    // planes – skip the 0th plane
        for m = 1 to M      // rows
          for n = 1 to N    // columns
                            // this simply means that the indices also pull in the
                            // scaling values
            Probe(m, n ,p) = ([X(n),Y(m),P(p)]dot[Vx,Vy,1])<Thresh
          end
        end
      end
      Map(vx, vy) =sum (abs(Probe*halfSpec))    // sums up the intensities
    end
  end
end
```

At this point, the work is almost done. The map is readily read by humans, but to make the method useable for further processing by a larger system, a routine to detect the peaks must be invoked. This a relatively straightforward process. The map is first normalized. A 2-D storage array the same size of the map is allocated. The normalized map is then examined point by point to see if any given point is a local peak, that is, above a threshold of 0.9 and it is higher than its 8 neighbors. If it is, its normalized cumulative intensity is stored in the array at the same location as the peak in the map; otherwise a 0 is stored. The indices of both arrays correspond to velocities. This way, only the largest values will be found in the auxiliary array; these large values indicate objects at the velocities indicated by the indices of the peaks. The velocities are stored in an array along with the cumulative intensities, used for diagnostic purposes. The array is then sorted in descending order so that the most prominent peak, corresponding to the object of interest, is first in line. Practice has shown that this peak is the object of interest and the next peak (assuming that there are no other specific moving objects) will be the background, if it's moving, too. Sometimes, even a moving background is so diffuse that it will not show up. The pseudocode for the peak detecting routine is given below as Figure 2.6. This routine effectively implements Equation 2.12.

Figure 2.6 Pseudocode for the peak detection routine.

```
Detectpeaks
Inputs: map, scalefac;
Output: vel

[R,C] = size(map);          // build data structures for the search
answer = zeros(R,C)
map = map/max(map)      // normalize the map to 0-1 range
thresh = mean(map)*0.9  // set the threshold to ignore any lower values
parfor i=2:R-1                  // go through the map to find peaks; ignore the edges
   for j=2:C-1
      if map(i, j) > {all of it's 8 neighbors}
         answer(i, j)= map(I ,j)
         if (answer(i, j) < thresh)
            answer(i, j)= 0
         end
      end
   end
end
B = floor(C/2)      // determine scaling of the map
B = B*scalefac     // scale factor is to adjust scale if a fractional pixel/frame
          spectrum
                    // is used
ruler = {C values from –B to B}     // set up index to velocity ruler
numpeaks= 0                          // initialize the counter
for i=2:R-1
   for j=2:C-1
      if answer(i,j) != 0
         numpeaks = numpeaks+1
         vel(1,numpeaks) = ruler(j)        // x in col space
         vel(2,numpeaks) = ruler(i)        // y in row space
         vrl(3,numpeaks) = answer(i, j)   // save the relative intensity
      end
   end
end
vel = sort(vel,-3)      // present in descending order
```

2.3 A Direct Corollary of the Core Algorithm – Fractional Pixel/Frame Resolution

A chirp-Z based 3-D transform was implemented to support the work described in this dissertation. It was successfully tested using the method above against various synthetic video sequences moving at rates as low as 1/8 pixel/frame. This 3-D chirp-Z is implemented a row or column at a time, as it is done with an FFT, as each dimension is separable. As a concession to speed and the fact that increased temporal resolution is unnecessary, the third (temporal) dimension is performed with an FFT. The dimensionality of the spatial planes changes as the computations proceed, so storage allocation takes that into account. The only change required in the energy mapping routine was to include variable scaling of the spectral values to accommodate the sub-pixel/frame resolution of the spectral volume. Using the expanded spectrum yields higher resolution results allowing direct estimation of velocities normally inaccessible to other techniques except by extrapolation.

The actual chirp-Z is accomplished with a rather basic form of the algorithm. The chirp-Z, in general, allows arbitrary resolution and spacing. It also encompasses the simpler FFT when the phase factors are appropriately chosen. Although the chirp-Z can deliver a spectrum with arbitrary spacing, or even compute a Laplace transform, it is restricted here to generating evenly spaced frequency values on the unit circle to allow for easy computation of the values of the resulting frequencies. The development of this form was pioneered by Bluestein [15] and generalized by Rabiner, Schafer, and Rader [16]. To describe the form we use, one starts with the usual 1-D DFT. Here we have the output spectral value X_k as the sum of the products of the phase factors as calculated in the exponential and the time domain sequence:

$$X_k = \sum_{n=0}^{N-1} x_n \exp\left(-j2\pi\left(nk/N\right)\right) \quad \text{k=0…K-1.} \tag{2.13}$$

For an input sequence of N points, an output of N unique spectral values will be generated. If we desire more spectral values, we must generate appropriate phase factors. Thus, if K spectral values are desired, and it is desired that they will be evenly spaced (which makes it easier to deal with the scaling), then we simply provide for K phase factors and modify Equation 2.6 as follows:

$$X_k = \sum_{n=0}^{N-1} x_n \exp\left(-j2\pi\left(nk/K\right)\right) \quad \text{k=0…K-1, N<K} \tag{2.14}$$

As deceptively simple as substituting *K* for *N* appears, this will generate K phase factors instead of N. That is, if K is larger than N, there will be more phase factors, spectral lines that are more closely spaced, and thus more resolution. Of course, this formulation will also allow larger than integral resolution, too, if that is of interest. The pseudocode for the actual "chirpZ2dplus" routine is shown below in Figure 2.7

Figure 2.7 Pseudocode for the actual 3-D chirp-Z used here

chirpZ2dplus
Input: A holds the video segment, Res holds the resolution (1/2, ¼, etc.)
Output: B holds the complex spectrum.

```
Calculate scale factor: M  = 1/Res
[R,C,P] = size(A)                    // get the size of the input
Allocate size of output as B = zeros(R*M, C*M, P)
Compute w = exp(-j2pi/(2*R))   // Assume square region: R = C
B1 = zeros (R, C*M, P)
B2 = zeros (R*M, C*M, P)
Loop through all the rows, B1= czt(A,R*M,w,1)
Loop through the columns B2 = czt(B1, C*M,w,1)
Loop through the plane axis B = fft(B2)    // fft used on the temporal axis
Return the answer
```

The actual computation of the chirp-Z in packages like MatLab is generally performed using FFTs by invoking Bluestein's identity, which is $nk = -(k-n)^2/2 + n^2/2 + k^2/2$. This laborious identity actually serves to decompose Equation 2.14 into

$$X_k = \exp\left(-j\pi k^2/K\right)\sum_{n=0}^{N-1} x_n \exp\left(-j\pi n^2/K\right)\exp\left(-j\pi (k-n)^2/K\right) \quad \text{k=0…K-1.} \qquad (2.15)$$

This poses the transform as a convolution and a multiplication by a complex scale factor. The exponentials in Equation 2.15 are the phase factors or, here, sometimes called complex chirps, referring to the frequency chirp sometimes encountered in radar processing, because plotting them will result in a sinusoidal chirp. This gave the transform its name. The process is as follows in Figure 2.8.

Figure 2.8 Pseudocode for Bluestein's method

Generate the first chirp vector $\exp\left(-j\pi n^2/K\right)$ over *N*.
Perform the multiplication of the vector $\vec{x}$ by the above chirp vector.
Take the FFT of the result.
// The above steps can be done once, as they do not change with *k*.
for *k=0* to *K-1* **do**
    Generate the second chirp vector $\exp\left(-j\pi (k-n)^2/K\right)$ over *N* for *k*.
    Take the FFT of the sequence
    Multiply the transforms together.
    Take the IFT of the product. This completes the convolution computation.
    Perform the summation as indicated.
    Multiply the result by $\exp\left(-j\pi k^2/K\right)$.
end

This accomplishes Equation 2.15 with 3 FFT operations per iteration, which is generally more efficient than computing it directly. This is basically the same method used by software packages such as MatLab.

Computing a chirp-Z can also be done by multiplying the input vector by a matrix of pre-computed phase factors, as can be found in any signal processing textbook. This is an alternative way to perform Equation 2.15. The pseudocode for this method is presented in Figure 2.9. Such a version of the chirp-Z was implemented just in case it could prove useful. It proved unnecessary, but it has potential. Although this method requires a bit more storage, it has two potential advantages. First, when a large number of identical transforms need to be made, pre-computing the phase factors could save time over the direct computation. A second possibility is the situation where a subset of the spectrum is required or where a particular spacing is desired. In this case, the particular phase factors would be generated to do the desired function.

To compute the transform of a vector of *N* values resulting in *K* spectral values, the following method is used. Note that the phase factors are identical to the exponentials generated in Equation 2.15.

Figure 2.9 Vector multiply method pseudocode

```
Allocate array W= K by N   // Array to hold the phase factors
for n=0 to N-1
        for k=0 to K-1
                W_kn = exp(-j2π(nk/K))
        end
end
// present N as a column vector
K=W*N
```

The result is row vector of *K* spectral values. A 2-D or 3-D transform would be performed a row and column as a time, as indicated previously for the FFT. Although the chirp-Z is a bit more computationally intensive than when using an integral transform like the FFT, its overall complexity remains the same (see [41] for recent approach). Again, because only one 3-D transform needs to be computed, absolute computational efficiency is not an issue here.

A version of the blob presented to the block matching and optical flow algorithms was created moving ½ pixel/frame. As before, the frame is 64x64 pixels; with this example, there are 64 frames. In Figure 2.6 a-c, blocks 3, 4, and 5 are shown to illustrate how the ½ pixel/frame is implemented. Figure 2.7 is the resulting motion map. It is obvious that it works.

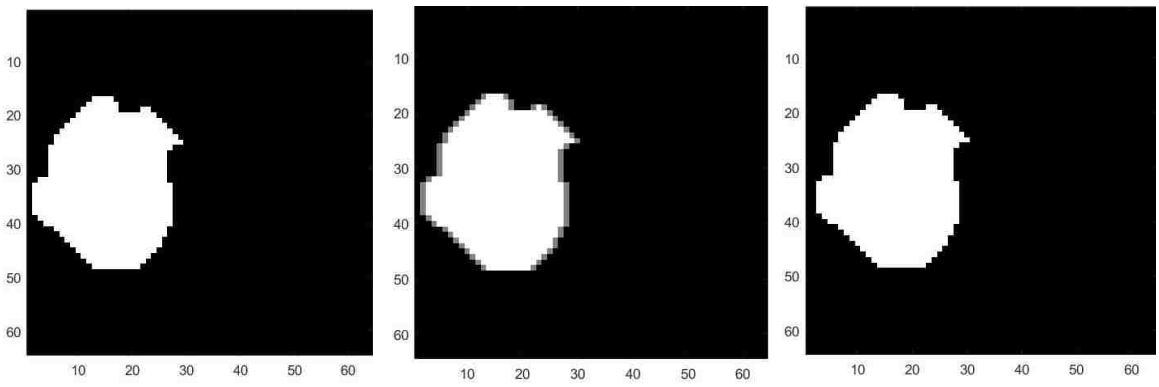

Figure 2.10. Frames 3, 4, and 5, blob moving 1/2 pixel/frame.

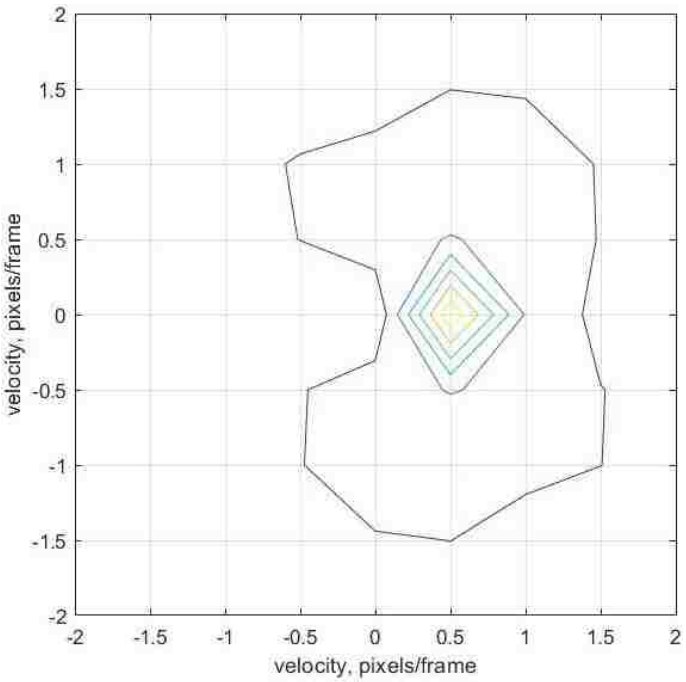

Figure 2.11. Motion map for slowly moving blob.

Chapter 3

Robust Noise Rejection, Multiple Object Detection and Quantification, Image Extraction, and Tracking

The core algorithm is an enabling technology for some interesting related methods and displays properties important to the successful deployment of any of these techniques. The first property is robust noise rejection. The second property is the fact that multiple objects traveling along the same path together (such as a convoy) will show up as a single "object." The first related method is the ability to reconstruct moving objects from their motion-tilted frequency planes. Second is a method to track the object of interest. Examples illustrating this will be shown using synthetic video data.

3.1 Robust Noise Rejection

Noise rejection is inherent in the core algorithm because of the fact that systemic noise will be scattered uniformly throughout the spectral volume while the moving object's spectral signature is concentrated on its plane. By filtering out everything but the image plane made by the object of interest, the only noise left will be the small portion of the noise coincidently on that plane. Of course, issues directly related to the object's image will remain, but that is usually not such a large amount to mask the motion. This will be illustrated by using synthetic data mixed with noise.

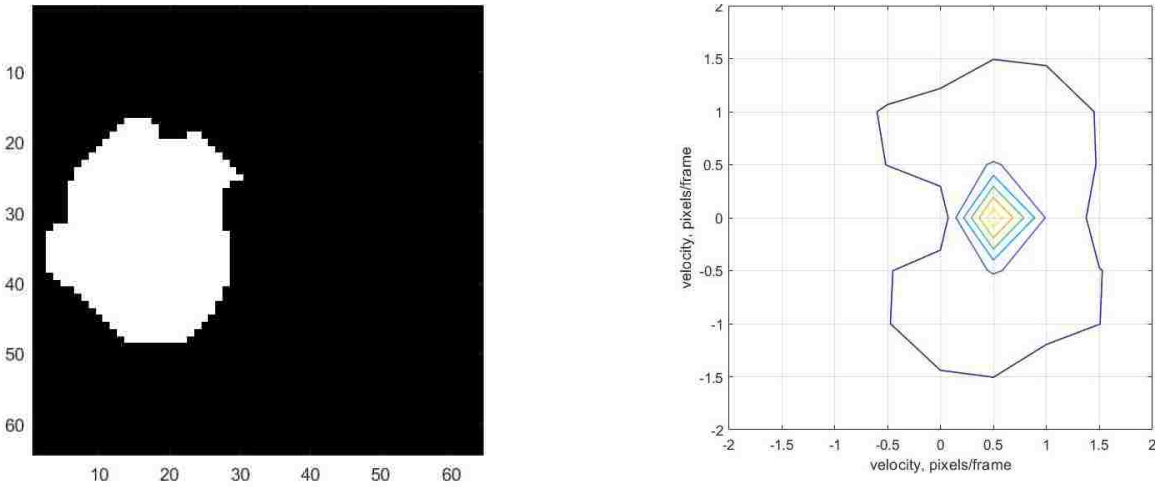

Figure 3.1 Large blob, frame 5, moving 1/2 pixel/frame l to r, with motion map, no noise

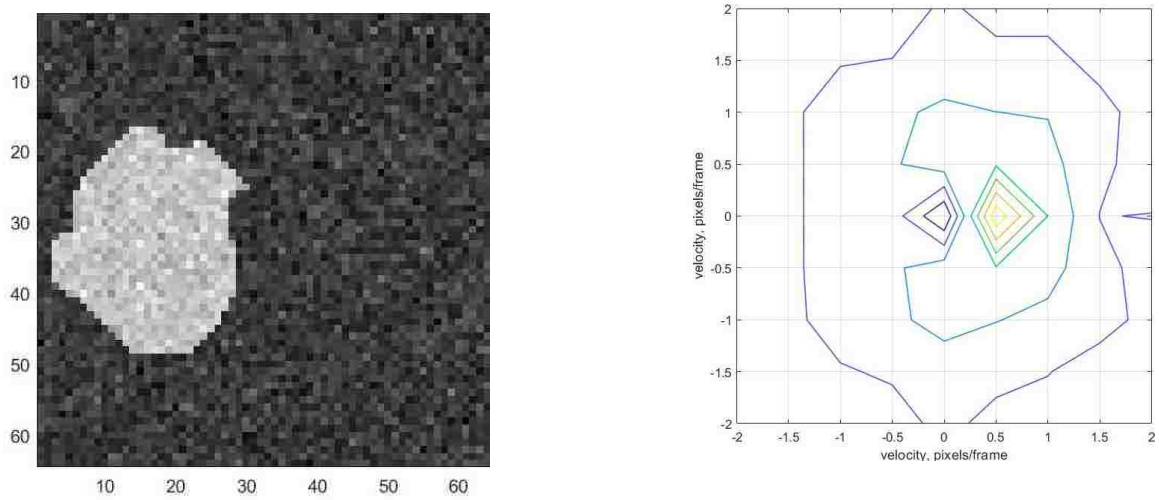

Figure 3.2 Large blob, frame 5, with motion map, 1-1 SNR additive Gaussian noise

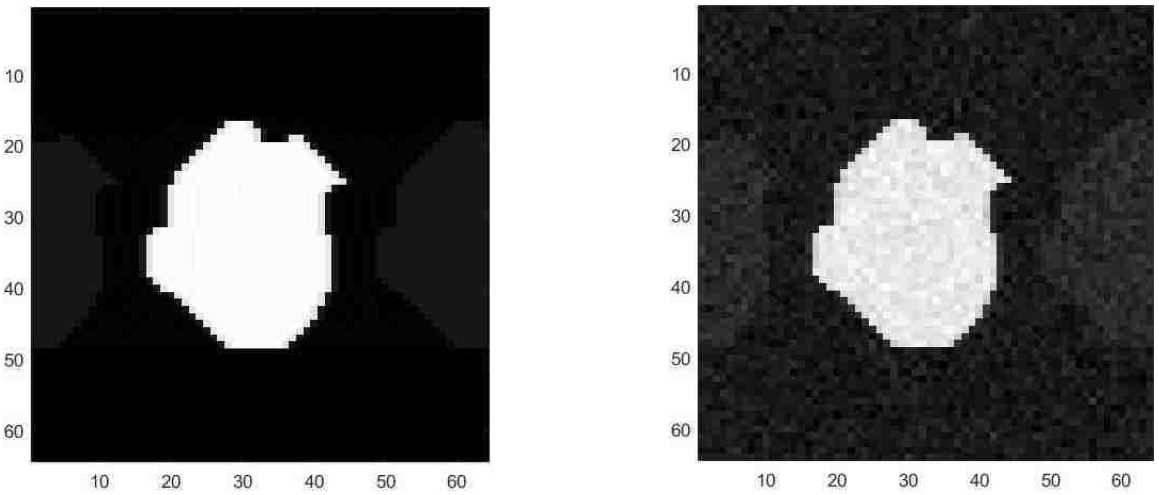

Figure 3.3 Extracted large blob images, no noise (l.), with noise (r.)

The example being presented above is the large, irregular blob used as examples with block matching (Fig. 1.1), optical flow (Fig 1.2), and fractional pixel/frame resolution (Fig. 2.9). Here, the blob is moving left to right at ½ pixel/frame. A Gaussian noise field was generated for each frame with an amplitude commensurate to the amplitude of the figure, so that the noise would have a signal to noise ratio (SNR) of 1 to 1. This noise was then added to a copy of the moving blob to generate a noisy version of the blob. Figure 3.1 presents the blob and its motion map without noise, and 3.2 shows it with the 1-1 SNR Gaussian noise added. As can be seen, with this larger structure, the noise floor is raised a bit, but otherwise, there is no real effect on the map. Figure 3.3 shows the blobs as extracted from the two videos, one without noise and one with noise.

3.2 Multiple Object Detection and Quantification

A natural result of the core algorithm is that objects moving together at the same velocity, as in a convoy or a formation of vehicles, will be mapped on the same 3-D frequency plane. Another, not so immediately obvious aspect, is that objects moving differently will inhabit their own planes, and will be detectable and quantifiable from their own peaks in the motion map. The algorithm even will detect a moving background, provided that it isn't diffuse, like a flat parking lot or a barren countryside. As above, this is best shown by synthetic examples.

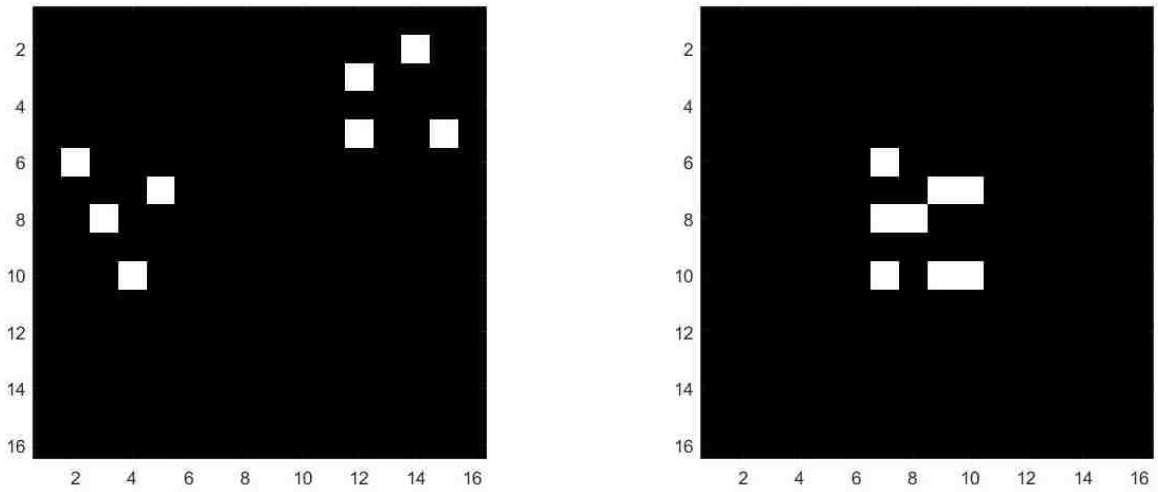

Figure 3.4 Two moving groups, frames 3 and 8 (at the merge)

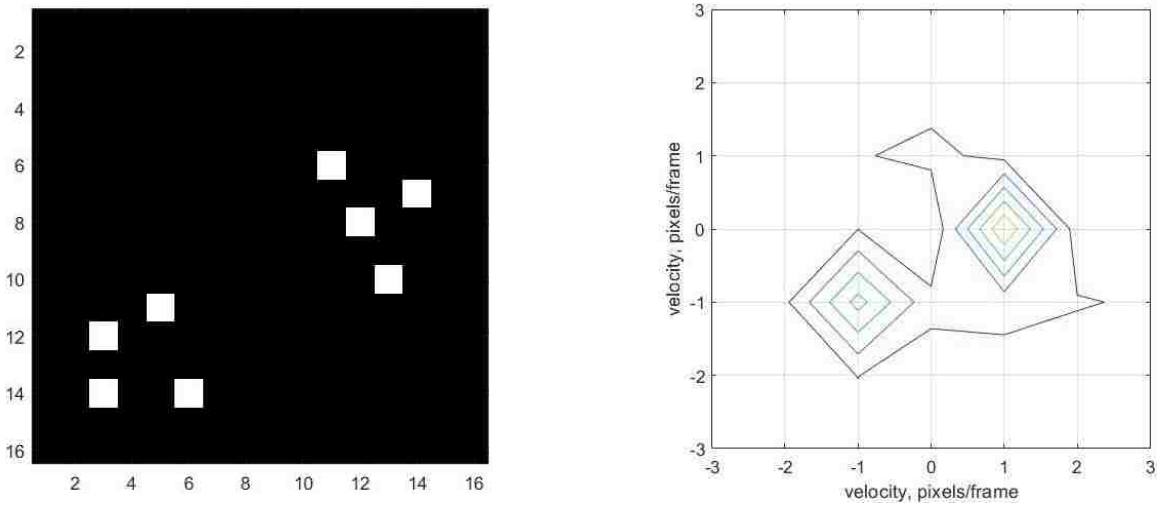

Figure 3.5 Two moving groups, frame 12 (after merge) and corresponding motion map

This demonstration is based on a simple synthetic video that portrays the motions of two formations of blocks shown in Figs. 3.4 and 3.5. The first formation is moving left to right, with the second formation moving right to left diagonally down, both moving one pixel/frame. Note the slightly higher noise floor, due to the fact that the blocks merge in the middle.

3.3 Image Extraction

Extracting the image, or segmentation, of a moving object is accomplished by isolating the plane in the original complex spectrum on which the object's energy resides by using a mask generated by the velocity of the object. Then using an inverse Fourier transform, the image of the object is recovered. Thus, the segmentation is not done as it is in the spatial domain.

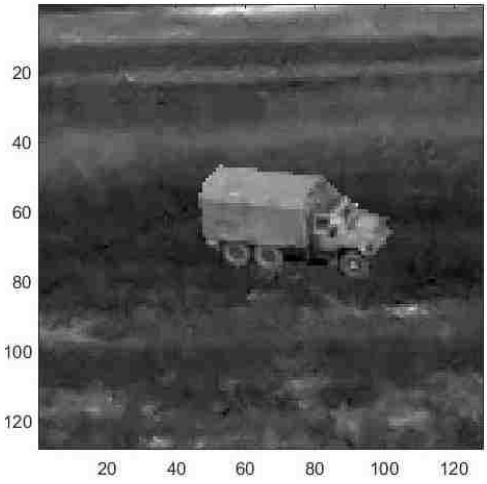

Figure 3.6 Original image from synthetic video sequence

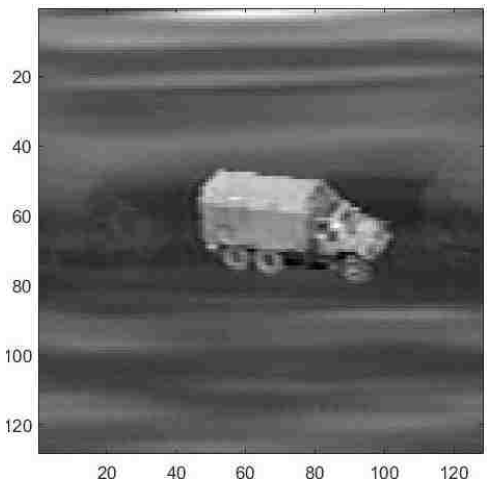

Figure 3.7 Image extracted from the video

A high fidelity simulation was implemented using actual photographs of terrain and an M-35 6x6 military truck to provide realistic test scenes. This simulator was used to generate the above video. The rate at which the truck moves across the terrain and the rate at which the background scrolls by is controlled by programmed parameters; thus all of the motion parameters are known. A more detailed description of the simulation is presented in Appendix A. The simulation parameters for this example were set at (2, 0) for the truck and (-2, 0) for the background. Frame size is 128x128; 16 frames were

generated. Figure 3.6 shows the original truck image in the simulation, while Figure 3.7 shows the extracted image. The high quality of the extracted image attests to the perfection of the simulation's imagery. Real life, as previously mentioned, has a way of blurring the extracted image.

How this algorithm works is as follows. First, the object of interest's velocity is required to extract the object's image from the video sequence in which it is embedded. This is required to build the mask to exclude the irrelevant spectral information from the spectral volume, leaving the object and whatever artifacts that are coincidentally on the object's plane. Once the mask is applied, the inverse transform is computed and the reconstructed image will be found on the 0^{th} plane in the new "video". The details of the algorithm is given below as Figure 3.8.

Figure 3.8 Pseudocode for the extract object algorithm

```
Extract_Object
inputs: video segment, x velocity, y velocity
output: extracted image

// compute complex spectrum of video segment
Spectrum = FFTshift(FTn(video segment))
// set threshold a bit looser to scoop up data that is close enough
Threshold = 0.2
//get size of spectral volume
(M,N,P)   = Size( Spectrum)
// build scaling vectors
X = N+1 values from -pi to pi
Y = M+1 values from -pi to pi
Z = P+1 values from -pi to pi
// allocate mask
Mask = zeros (M.N,P)
// build the mask
for p = 1 to P            // planes
   for m = 1 to M         // rows
```

```
    for n = 1 to N        // columns
      //this is set up to allow the indices to also pull in the scaling values
      //using a logical expression automatically gives a 1/0 answer
     Mask(m, n ,p) = ([X(n),Y(m),P(p)]dot[x velocity, y velocity,1])<Thresh
    end
  end
end
// apply the mask to the spectrum
Spectrum = (Spectrum.*Mask)
// take the IFT of the spectrum
OutSpec = IFTn(Spectrum)
// select the 0th frame – that holds the extracted image
extracted image = 0th video frame
```

3.4 Tracking, After the Fact

Tracking in video is usually considered a real time activity, with the tracker computing deviations from the center of the field of view using a centroid, edge or correlation algorithm. These errors are then fed into a tracking loop to correct the aim of the camera's gimbal to allow it to follow the action. This is not the case here, however. Because the image of the object of interest can be segmented out, it can also be followed through the video sequence. The path of the object may be of great interest to military analysts or law enforcement investigators trying to reconstruct an incident.

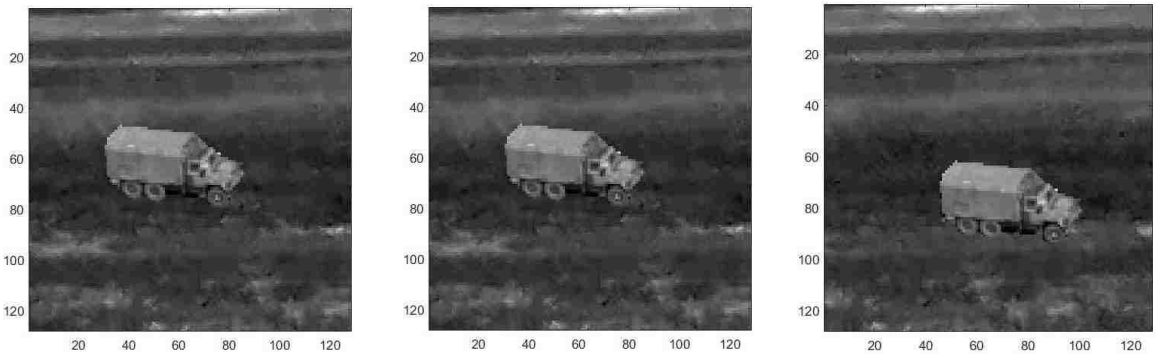

Figure 3.9 Truck simulation baseline, frames 1, 9, and 16

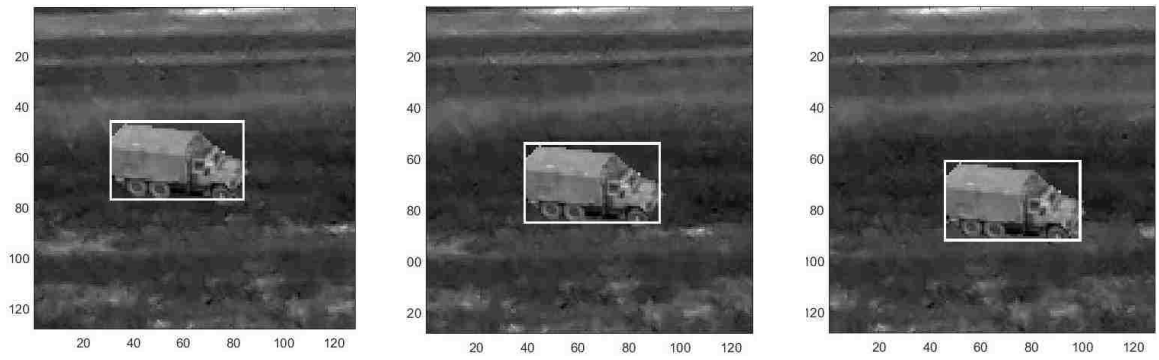

Figure 3.10 Truck simulation with tracking boxes, frames 1, 9, and 16

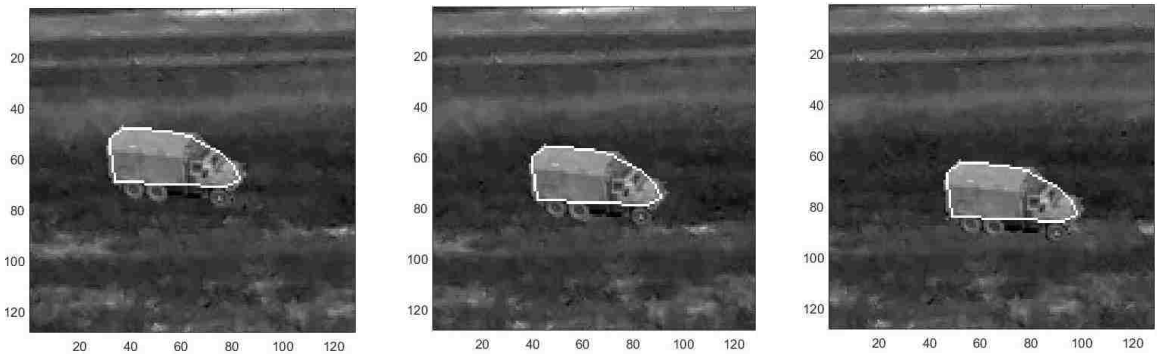

Figure 3.11 Truck simulation with convex hull, frames 1, 9, and 16

What the user of this method does first is take the extracted image and stretch the histogram, then normalize the image gray scale values between 0 and 1. The method is then invoked, passing to it the original video segment, the conditioned extracted image, and the previously measured velocity of the object of interest. The routine calls a function to place the bounding box or convex hull on the image. The final step is to superimpose said boxes or hulls on each frame of the output video using the appropriate function, computing the location of the object in the next frame assuming constant linear motion. The resulting modified video segment is returned to the user.

To illustrate the result of this process, Figure 3.9 shows a segment of the synthetic truck simulation, this particular one showing the truck moving to the right and down one pixel/frame each, with the background moving to the left at 2 pixels/frame. Figure 3.10

shows the same sequence with tracking boxes applied. Figure 3.11 shows the same sequence with the convex hull applied. Pseudocode for the tracking method is given below as Figure 3.12 a-d. Sufficient detail is provided for understanding the flow of data; the actual MatLab code is to be found in Appendix B, should one care to examine it. The pseudocode is of the latest version of the tracking box method. The convex hull method is essentially identical to the box method, except that the bounding box routine is replaced by a convex hull routine.

Figure 3.12a Tracking method main routine

```
Follow_Object_Report_Modified
//This is the most advanced version of the tracking program. The modification
//is adding the output of the box array to allow for the comparison of this
//method to the IPCA method.
Inputs:
        invid = original video,
        exobj = extracted image (previously normalized and adjusted),
        xvel = x velocity,
        yvel = y velocity
Outputs:
        outvid = output video (with applied boxes),
        boxes: box array}

//Make beginning of boxed, output video by copying input video
outvid = invid
//Get the size of the video
(R, C, F) = Size (outvid)
//call bounding box function to get the specs for the bounding box
BB  = get_object_bounding_box (exobj)
//Bias the location of the bounding as indicated by the velocity vectors
BB(1) = BB(1) - (F/2 * xvel)
Bb(2) = BB(2) + (F/2 * yvel)  // yvel is up for humans, but in videos y is down
//Loop to add the boxes to the output video
For f = 1 to F
        boxes(:, f) = BB
        //call the box adding function
        outvid(:, :, p) = add_standard_box(outimg(:, :, p), BB)
```

```
        //update the box position
        BB(1) = BB(1)+xvel
        BB(2) = BB(2)-yvel
end
```

Figure 3.12b Function that calculates the tracking box

```
Get_object_bounding_box
//This function makes use of MatLab image processing functions to find and
// erect the bounding box.  An alternative version makes convex hulls.
Inputs: exobj = extracted object
Outputs: box = bounding box specification

//Turn the exobj into a binary image.  It's supposed to be gray scale expanded
//normalized so that the max value is 1.  The threshold is set heuristically so that
//only the extracted object is left.
B = make_bin_img(exobj, threshold)

L = bwlabel((B)        //Label binary image blobs
P = regionprops(L)     //properties include centroid and bounding box params.
NB = length(P)         //Determine number of regions

//If more than 1 blob, concentrate the blobs into the ones most likely to be
//part of the object.
If NB = 1
        //use the one blob
        Box = P(1).BoundingBox
else
        //Sort the list descending by size of blob.  This is a simple home-made
        //bubble sort – no need to bore the reader with something this trivial.
        P = sort_blobs_desc(P)

        //To come up with a sane way to select blobs that are part of the object,
        // choose the largest dimension of the largest blobs bounding box.
        SL = {the largest dimension of the largest blobs bounding box}

        for i = 2 to NB  //minor loop to exclude blobs that are too far away
                D = {distance to the center of this blob from the biggest blob}
                If D > 3*SL
                        {delete this blob from the list}
                end
```

```
        end

        L = bwconvhull(B)           //Throw a convex shell around the blob(s)
        P = regionprops(L)          //Get the region properties of the hull
        box = P(1).BoundingBox      //extract the bounding box specs
end
Expand the box by 2 pixels each direction
```

Figure 3.12c Function that produces the binary image for the bounding box

```
Make_bin_img
//This little number turns a stretched and normalized gray scale image into a
//binary image for the MatLab routines to properly find the brightest object
//and measure its properties.  Very likely, the image ends up a blob.  A threshold
//is provided by the caller to adjust the cut-off level as desired.

input: inimg: input image, thresh = threshold
Output: bw: resulting binary image}

 (r, c) = size(inimg)     //r and c are rows and columns, respectively
bw = zeros(r, c)          //preallocate space for the binary image

//loop through the image, applying the threshold to the pixels
for i = 1  to r
        for j  = 1 to c
                //This logical automatically yields 1 for pixels >= the threshold, 0
                //for the rest of them.
                bw (i, j) = inimg(i, j) >= thresh
        end
end
```

Figure 3.12d Utility function to superimpose a box onto a frame of imagery

```
Add_standard_box
//MatLab's image processing routines use a standard definition for a boxes,
//which are often used to segment items in a digital image.  This utility routine
//applies the supplied box to the supplied image based on the box definition and
//location as specified in the definition.  Efforts are made to avoid the trivial,
//boring details.  Besides, some of this will be language dependent.
```

Input: inframe = input image frame, boxdef = box definition;
Output: outframe = output image frame}

//Copy the input frame into the output frame
outframe = inframe
//Get the size of the image.
(R, C) = **size**(outframe)
//Clip the boundaries of the box if they go past the edges of the frame
//Local corner variables are ulr = upper left row, ulc = upper left column,
//lrr = lower left row, lrc = lower left column. They are clipped as follows:
//
{Set the local corners to the box values; check the values against the frame and clip them if they extend beyond the edge of the frame.}
//Establish the maximum pixel value. Needs to get this so the box is visable.
maxpic = **max**(inframe)
//
{Draw the top, bottom, and sides on the frame}

Chapter 4

Some Results with Real Data

Up to this point, examples have been given with synthetic data. Here, the results of trials with real data are given. One source of data is the LIVE Video Quality Database from University of Texas, Austin (UTA) Laboratory for Image & Video Engineering (LIVE) (http://live.ece.utexas.edu/research/Quality/index.htm) [17, 18]. Another source is airborne video sensor data obtained from DARPA's VIVID database (http://vision.cse.psu.edu/data/vividEval/datasets/datasets.html) [19]. Neither database was created with research like this in mind, but there were sufficient examples useful for this work to glean the necessary real-world data for this work to proceed. Actual files will be identified as the narrative progresses.

4.1 LIVE Data

Standard, publically accessible imagery was sought to test the algorithm. One such source is the LIVE video database from UTA. Because these sequences were clean and clear, the first serious looks at real data were done with these examples. This is not to say that they are easy; some, the cars on the street and the pedestrians crossing the street are quite complex. All the sequences selected were converted to ".avi" format to enable processing. Six short sequences were selected for testing from five videos, based on their potential to provide information on the behavior of the algorithm. The peak detector was used to verify any peaks found. The videos chosen were all 25 frames/second withv768x432 pixel frame size. The "13" at the end of each file name specifies the use of this format. The short sequences used were cropped to 128x128 frame size and 0.62

seconds (16 frames) and converted to gray scale, as color is irrelevant. A segment length of 0.62 seconds is sufficiently short to enforce motion linearity. Segments, sources, and results are summarized below in Table 1. Significant results will be discussed in more detail below.

Table 1 Summary of Testing with LIVE Videos. Good – motion map shows expected result; Mixed – unexpected results.

Segment name	Source	Figure Reference	Results
Car changing lanes 1	rh13	Fig, 4.1	Good
Car changing lanes 2	rh13	Not shown	Good
Pedestrians crossing street	pa13	Fig. 4.2	Mixed
Jogger on canal path	pr13	Fig. 4.3	Good (Background motion only)
Docent and tapestries	sh13	Not shown	Good (Background motion only)
Tractor plowing field	tr13	Not shown	Good

Nominal results were those that generated a motion map showing the "right" answer, based on simple visual examination. Half of the samples exhibited the expected results. The pedestrians crossing and the jogger exhibited interesting results that were not expected. The docent's result, in light of the jogger's result, were in line with what was going on and were not unexpected.

4.1.1 Cars Driving Up the Hill

The cars up the hill sequence (rh13) provided two encounters suitable for use. The first of the two will be presented here as an example of a nominal result. Figures 4.1a, b, and c show the this encounter, with 4.1a and b showing the initial and final position of the car, and 4.1c showing the motion map with the 3 pixel/frame right clearly indicated. The motion is most evident from the position of the headlight. Note the clear peak in the motion map at (0. 3).

Figure 4.1a Car one, frame one

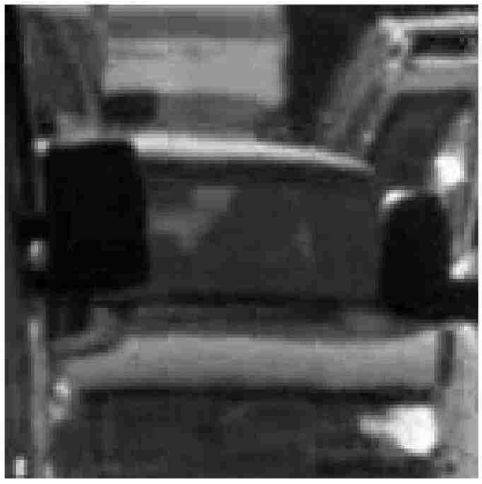

Figure 4.1b Car one, frame sixteen

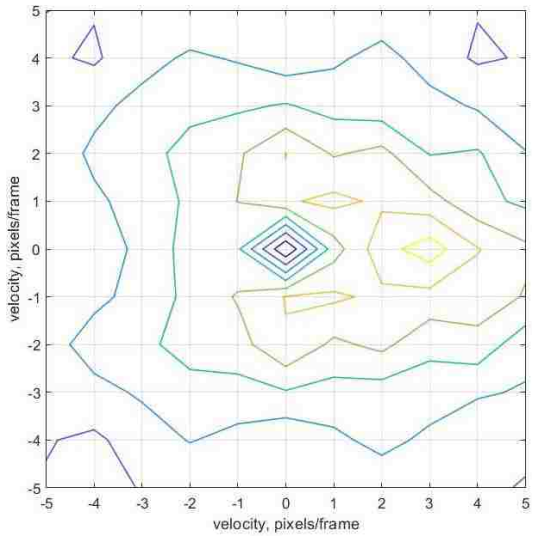

Figure 4.1c Motion map for car one lane change

4.1.2 Pedestrians Crossing the Street

The next video segment (Fig. 4.2) is a LIVE sequence from the pedestrians crossing the street (pa13). It is interesting for a number of reasons. It is the most complex of the LIVE sequences, and it presents a number of challenges. The desired objects of interest are a mature woman crossing close to the camera left to right and a young woman walking right to left in the background There is also a component of pan in this video, where the camera is panning very slowly left to right. On top of all this, there are several other pedestrians milling around in the scene. This sequence illustrates the difficulties encountered when working with real-world imagery that is cluttered with many moving objects, including the background. The challenge here is for the algorithm to find meaningful peaks in the motion map when there are many moving objects in the scene and sudden scene changes (people walking behind other pedestrians), generating what can only be characterized as "noise". Indeed, it is difficult to identify whose peak is whose, even when looking at the video. The third frame of the sequence clearly introduces the principle subjects of the sequence. This is illustrated by showing the third frame of the video sequence as Figure 4.2a, where we see the mature woman, a young woman, and another person in the background, along with the corner of the building. As the sequence progresses, the mature woman masks the young woman for a few frames, then as she moves on and unmasks the young woman, the man in the background comes in and out of view from behind the young woman, the ATM on the wall of the building is unmasked contributing its bright reflection, and a fourth person starts to enter the scene. This all contributes to the confusion seen in the motion map, which is given in Figure 4.2b. In spite of all this, the motion map clearly shows the young woman's motion at (-2,

0). The mature woman doesn't show up as prominently as one might think, as her image is blurred by her motion and by being in poor focus and thus she doesn't generate the energy in the spectral volume that it would if her image was crisper. Her impact on the motion map is apparent in the small peak at (8, 0). It's just one of many bumps and burbles in the map, except that it's a little higher than the rest. It's just not very obvious with all the other undulation going on. The extracted and reconstructed images of the young woman and the mature woman are shown in Figures 4.2c and 4.2d, respectively. It is important to note that there is a small lump at (-2, 2) which appears to correspond to the left to right and up pan (the background moves opposite to the camera's motion).

This scene would challenge any algorithm. A bit of luck was obtained as the young woman's image was crisp enough to make a clearly visible peak in the motion map. The mature woman's peak was identified in the map at the spot suggested by direct measurement. The background was also teased from the scene by direct observation. This brings out an important point. At times, a wealth of moving objects will mask each other in the map, as is the case here. If one has *a priori* knowledge of the velocity vectors or a way to measure the motion of a given object directly, its image can be extracted and reconstructed regardless of the clutter and confusion present in the motion by map using the related extraction method independently of the motion estimation algorithm. This could be useful in some instances.

Figure 4.2a Original image, frame three

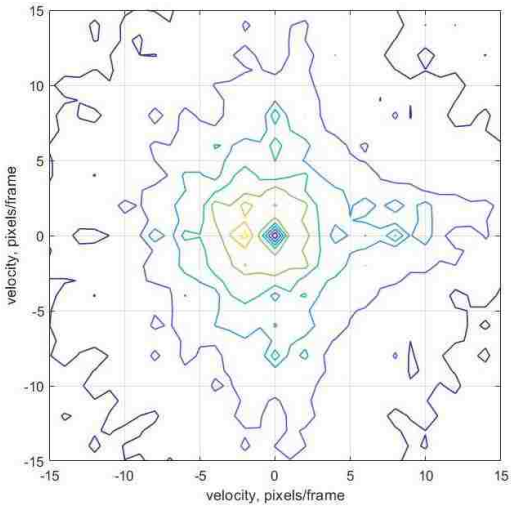

Figure 4.2b Motion map for Pedestrians

Figure 4.2c Reconstruction of young woman, moving left at (-2, 0)

Figure 4.2d Reconstruction of mature woman, moving right at (8, 0)

4.1.3 The Jogger

The jogging man sequence (Fig. 4.3) provides another example of the power of these algorithms. The pr13 sample shows a gentleman with an umbrella jogging along on a canal bank. The limitation here is the fact that the camera man panned along so smoothly and kept the jogger in the center of the frame so well that his motion relative to the frame could not be determined algorithmically. Under such circumstances, the moving object being tracked appears stationary as far as the video is concerned. However, the background will be clearly in motion, as it is here. By analyzing the data, the background can be extracted with the jogger essentially removed.

The first frame of the sequence is shown in Figure 4.3a, the motion map in 4.3b, and the extracted background in 4.3c. The right to left motion of the background is clearly seen in the map, as well as the absence of any motion on the part of the jogger. An artifact of the jogger remains as a ghost shadow none the less, as some of its data lays on the plane where the background's spectrum exists. This illustrates the fact that an object momentarily obscured by another can be recovered anyway, as its spectrum is extracted over several frames. This fact can be useful if the object of interest is the one being obscured, as illustrated both here and in the pedestrian example above.

Figure 4.3a Jogger sequence frame 1

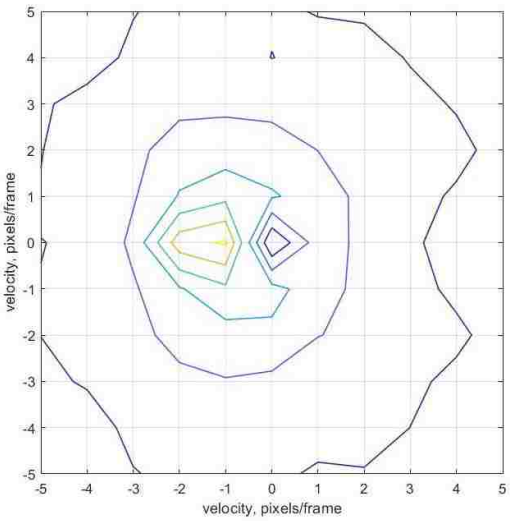

Figure 4.3b Jogger sequence motion map

Figure 4.3c Reconstructed background image of the Jogger sequence

4.2 VIVID Data - Actual Aerial Videos

The VIVID data base afforded the opportunity to examine video with multiple vehicles moving at different speeds and directions. At the time of downloading the DARPA VIVID database, there were 185 data events which were recorded by three cameras – a wide field color video, a narrow field telephoto color video, and an IR camera with a medium field of view. The cameras were mounted on the same gimbal, so they all saw the same events in somewhat different ways. For the most part, the wide field camera generally gave the best view for the purposes of these trials. The aerial platforms consisted of a UH-1 helicopter, an RC-26 Metroliner aircraft, and a C-23 Sherpa aircraft. It is not always obvious as to which aircraft is the platform, but they share some of the same obstacles in terms of platform vibration. While the RC-26 had a stabilized platform installed in a belly turret, the Sherpa had to make do with an open door on the side. The original purpose of these videos was to determine if specific activities could be elucidated from the videos. The videos were in color, were longer than required for these short sequence trials, and the size of the frame was a problem in the early trials, as the algorithm had not yet been parallelized and optimized for speed. Again, color was irrelevant, so the selected samples were rendered into gray scale, as it was in the case of the LIVE videos. The single car events were performed with 128x128 pixels by 16 frames

Unfortunately, only a small number of the files lent themselves to analysis due to various reasons, such as bad focus, poor lighting, clouds, foliage in the way, and the camera operator being instructed to follow a particular vehicle. The camera operator's skilled manual tracking rendered that vehicle essentially stationary and unsuitable for analysis.

That notwithstanding, there were spots out of ten files that were suitable and are presented here. If it was desired to get the actual velocity of a given vehicle, one must add the velocity of the frame along the ground created by the camera aircraft's motion; the observed velocities are only relative to the frame, and the frame is moving. Thus, velocities measured by the algorithm are not actual vehicle velocity but velocity relative to the frame only. A summary of the results with the VIVID data is presented below in Table 2. The really good, the good enough, and the problematic events are discussed below.

Table 2 Summary of Testing with VIVID Videos; Good - motion map as expected; Mixed – unexpected or anomalous result; Poor – did not work as expected

Segment Name	**Source File**	**Figure Reference**	**Results**
Car passing 1	V3V100003_007.avi	Fig. 4.6	Good (some platform motion)
Car passing 2	V3V100003_008.avi	Not shown	Mixed (one vehicle tracked too well)
Car passing 3	V3V100003_008.avi	Not shown	Good (some platform motion)
Car passing 4	V3V100012_009.avi	Fig, 4,9	Mixed (puzzling map)
Cars passing by each other 1	V3V100004_003.avi	Not shown	Good (some platform motion)
Cars passing by each other 2	V3V100004_005.avi	Fig, 4.4	Good (nearly perfect)
Cars passing by each other 3	V3V100003_008.avi	Fig. 4.8	Poor (excessive platform motion)
Cars turning 1	V3V100003_004.avi	Not shown	Poor (excessive platform motion)
Cars turning 2	V3V100003_007.avi	Fig. 4.7	Good (really interesting result-see below)
Car in and out of the shade	V3V100007_015.avi	Fig. 4.5	Good (really interesting result-see below)

4.2.1 Best Results

Not all of the results were stellar; a little more than half were good, or at least usable. Not to be redundant, the encounters illustrate the results when the conditions are favorable (good focus, low camera motion, etc.). Here are the best of the data sets;

4.2.1.1 Cars Passing Each Other 2

The second passing by event was taken from V3V100004_005.avi, frames 1270-1285, with the 128x128 pixel window size. The two sedans are passing each other, crossing the frame at about 45 degrees. The motion map says it all, with the top car moving diagonally right to left and down at (-1, -1) and the bottom car moving left to right and up at (1, 1). The 9th frame is shown as Figure 4.4a and the motion map as Figure 4.4b.

Figure 4.4a Second Passing By event, frame 9

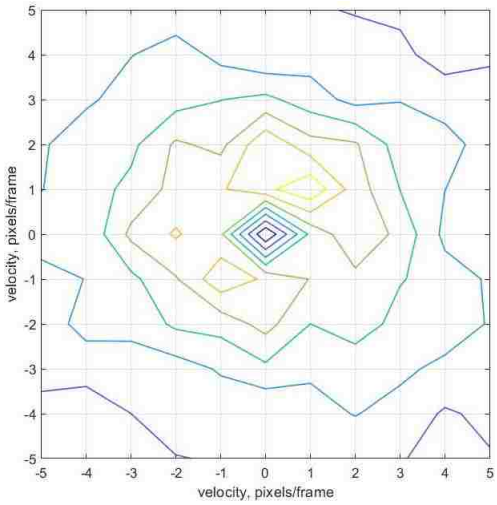

Figure 4.4b Second Passing By event motion map; both cars are evident at (1, 1) and (-1, -1)

4.2.1.2 Car In and Out of the Shade

The last example given here is that of a car driving through spotted shade, so that in any one frame, the car is illuminated with spots of light coming through the trees. This sort of illumination was originally thought to be problematic, but apparently it appears to be not a problem. The sequence comes from file V3V100007_015.avi, frames 500-515, again with a 128x128 pixel window. The spots of light move along the car as it drives by, with the rear deck generally illuminated all the time (see Figure 4.5a). The resulting motion map indicates motion at (2, 0) (see Figure 4.5b). The extracted image (Figure 4.5c) appears to be evenly illuminated and shows no evidence of the light and dark interplay we see in the video itself. This is similar to the events above in the previous section where previously blocked features are revealed because the various spots were "saved up" in the spectrum and then returned with the reconstructed image. Both of these examples are exceptionally clear, almost as clear as the truck simulation examples.

Figure 4.5a Car in and out of shade, frame 9

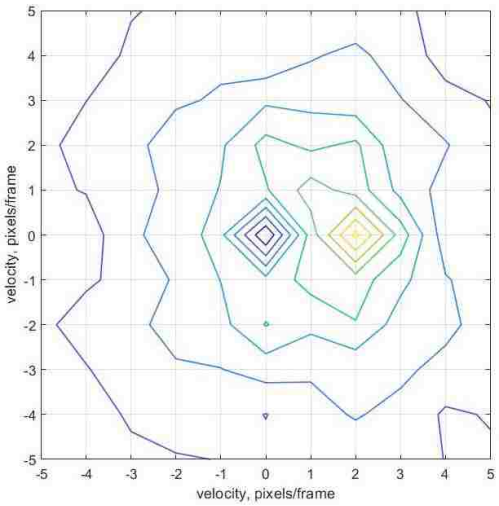

Figure 4.5b Motion map for Car in and out of shade

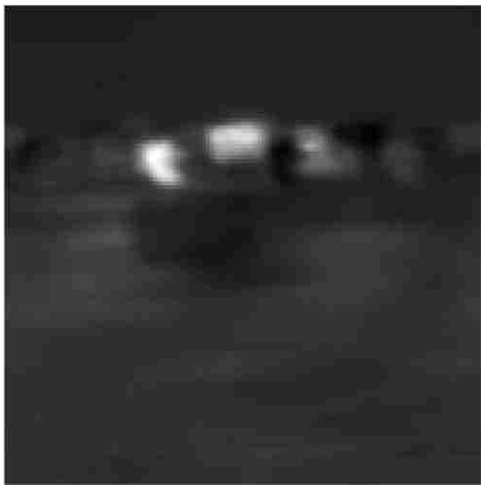

Figure 4.5c Extracted Image of Car in and out of shade

4.2.2 Good Enough Results

The "good enough" results would be sufficient to determine velocity when ground track velocity of the platform is included, the image(s) can be extracted, and so on. Instead of showing all four of the "OK" examples, the most representative two examples will be discussed.

4.2.2.1 First Passing Event

The first mediocre event selected is from file V3V100003_007.avi, frames 1765-1780 (16 frames), using a 128 pixel x 128 pixel window. The "top" car is passing the "bottom", both moving left to right. The background is flowing right to left, as expected, as the aircraft is moving left to right. The camera man is tracking the passing car, rendering it essentially stationary, while the bottom car is overtaken and passed. When analyzed, the motion map of the event shows two peaks: one at (-2, -2) and the other at (0, -1). The video clearly shows that the passing car is being tracked in azimuth, but not

elevation. As such, the peak at (0, -1) represents this car. The car being passed is represented by the peak at (-2, -2). Both appear to be going "backward," but that is due to the helicopter flying faster than the cars are driving. Background is nowhere to be seen, as the runway on which they are driving is virtually featureless and doesn't leave a clear signature in the spectral volume. It is also possible that the aircraft is moving too fast for the background to be analyzed by the algorithm. Frame 4 is presented as the example of the video sequence as Figure 4.6a, and the corresponding motion map as Figure 4.6b. This result is not the best, but it is usable.

Figure 4.6a First Passing event, frame four

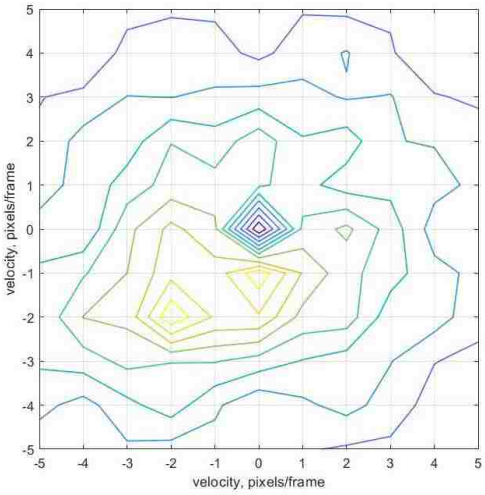

Figure 4.6b First Passing Motion Map; Passing Car at (0, -1), Passed Car Left Behind at (-2, -2)

4.2.2.2 Second Turning Event

The second turning event (Fig. 4.7) comes from file V3V100003_007.avi, frames 670-685, again with the 128x128 pixel window. What merits this event a second thought was that at first look, it seems that what the map is reporting is platform motion; it turns out that there is no platform motion at all. The helicopter seems to have paused or turned in

such a way that the camera was still, while the cars drove by. The video captures left turn, but this time both vehicles are on the bottom of the loop as they turn back up the runway. The leading car seems to be going up while the trailing car seems to be going left to right. The truth, as it turns out, is that the bright spots on the side of each vehicle are roughly in line and both are going left to right at the same speed. As such, they register in the spectrum as if they are going the same speed and direction (see Figure 4.7a).

There is only one peak, at (2, 0), in the map. The question could well be "Which car is represented by this peak?" in Figure 4.7b. The answer is "both." As indicated in the theory section, this peak should represent all motion going that direction at that speed, implying that the bright spots on both cars are captured in this one peak. This notion is verified in the extracted image at (2, 0) given in Figure 4.7c, where both vehicles are clearly visible. Mediocre? No – surprising, yes, and maybe misleading. Clearly usable, as evidenced by the extracted imagery, even if it's not perfect.

Figure 4.7a Turning event 2, frame 9

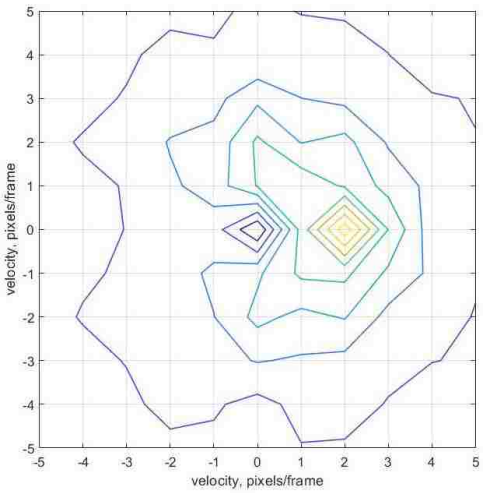

Figure 4.7b Motion Map for Turning event; both vehicles share the indicated motion

Figure 4.7c Image extracted at (2, -2)

4.2.3 Difficult Cases

These results are a bit problematic. One example will show how a clear result really might be misleading, the other will show that the answer may be there, but not immediately discernable.

4.2.3.1 Cars Passing by Each Other 3

The third and last passing by event is found in V3V100004_008.avi, frames 290-305, with a 128x128 pixel window (Fig. 4.8). What should have been an easy identification gets complicated by two factors. First, the white pickup is hardly moving. It shouldn't show up at all, except it has a slow drift down, throwing some energy into the spectrum on the (0, -1) plane. Second, because it's white, its trace in the spectrum is powerful, bleeding into the some of the adjacent spectral space and smearing all over the neighborhood. The dark Mustang doesn't have a chance to make a discernable peak in the plateau in the motion map made by the two vehicles. Throw in a little platform jitter, and we have nothing from the Mustang, and, when extracted, the reconstituted image pickup doesn't fare well (see Figure 4.8c). Frame 9 of this sequence is shown in Figure 4.8a, with the corresponding motion map in 4.8b. Because the energy in the spectrum of the truck was smeared all over the map there were no visible peaks. The peak detector program tried to deal with this situation and located a peak pickup at (0,-1). The motion map was essentially destroyed by the bright white truck, it would seem.

Figure 4.8a Third Passing By event, frame 9

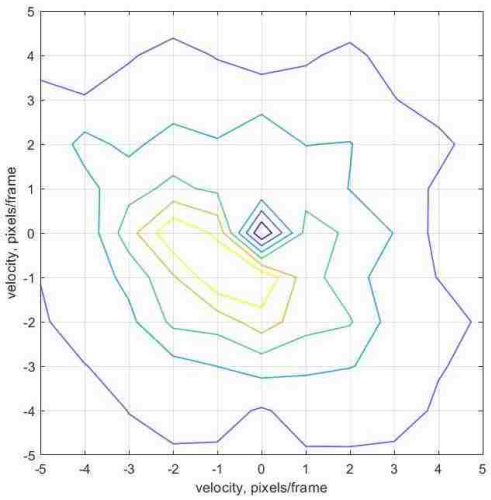

Figure 4.8b Motion Map for Third Pass By event; white pickup is not detectable without detect peak program; Mustang not detectable

Figure 4.8c Extracted image from Third Pass By at (0,-1)

4.2.3.2 Car Passing 4

The fourth passing event (Fig. 4.9) is a High Mobility Multipurpose Wheeled Vehicle (commonly called a Humvee) pulling a tank trailer and passing an M-35 6x6 truck. This sequence is from V3V100012_009.avi, frames 1365-1380 (16 frames), and a 128x128 pixel window. The camera operator tracked the M-35, letting the Humvee move on by. Frame 4, representing the video sequence, is seen Figure 4.9a. The motion map, given as

Figure 4.9b, shows this, as the Humvee clearly shows up at (-2, 2) (see Figure 4.9c), and the M-35 seems to be inconclusively smeared all over what might have been a clear background indication or the actual truck's motion. This could have been caused by aircraft motion from mild turbulence. This is disturbing, as one would expect the truck to show up somehow, because the camera did not perfectly track the truck. However, this is not the last word on this data sample, as will be shown below.

Figure 4.9a Fourth Passing event, frame four

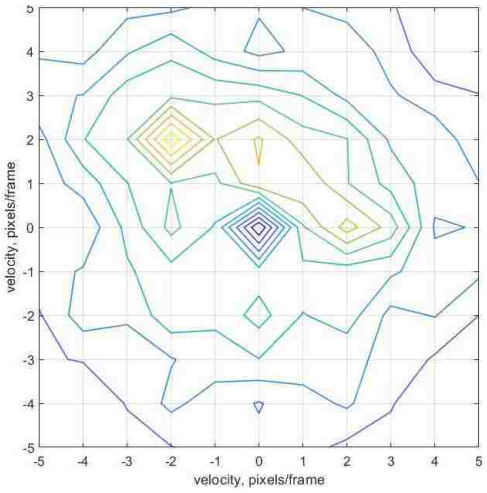

Figure 4.9b Fourth Passing motion map; passing HUMVEE with trailer at (-2, 2), passed M-35 lost in the noise

Figure 4.9c Extracted image at (-2 ,2)

4.2.3 Tracking Real Vehicles

The tracking program is now demonstrated on a real-world video. To show its effectiveness, it was applied to an aerial sequence from VIVID file V4V200005_023.avi, frames 2801-2816, with a window of 256x256 pixels, showing a van driving through an intersection. Figures 4.10 shows the "boxed" frames 1, 9, and 16 of the sequence. As with the "truck on the meadow" simulation, the box is not perfect because the process that calculates the box has only the binary image's blob made from the extracted image to go on. It does, however, demonstrate the effectiveness of the algorithm's products. For completeness, the convex hull was also placed around the image. The results are shown in Figures 4.11 for frames 1, 9, and 16. The method clearly works on real vehicles in real video, too.

Figure 4.10 Van through the intersection, frames 1, 9, and 16 with tracking box applied

Figure 4.11 Van through the intersection, frames 1, 9, and 16 with convex shell applied

4.3 Interesting Conjectures

Some topics came up during the evolution of this work. These topics might be called "odds and ends." They are worthy of discussion, taken up here in no particular order.

4.3.1 Convoys

An obvious application of the tracking method to follow convoys of vehicles of interest. Preliminary results obtained during the initial trials with blocks and blobs and later with groups of vehicles moving together validates the theory, which indicates that such a group, traveling together in a given direction with a common velocity, will show up in the motion map as a single object. In theory, they will show up together, as they occupy the same tilted plane in the spectral volume. The tracking algorithm was modified to accommodate an extended group and the trial proceeded. It must be admitted in advance that the original software should not be considered production quality, and the modifications to it to track a convoy is even less so. Even so, the "hack" on the original software works well enough to prove the point.

In the selected sample, we have a group of three vehicles traveling roughly vertically up, in respect to the frame, from extracted from frames 720 to 735 of V3V100003_005.avi (Figure 4.12, below). The observing aircraft is crossing right to left and up faster than the convoy is moving. Thus, the motion map of the clip (Figure 4.13) shows that the motion of the group relative to the frame is 2 pixels/frame right and 3 pixels/frame down. This was not unexpected. However, when the tracking algorithm attempted to separate the vehicles from the residual background, a problem was encountered with the residual background "stealing" the tracking gate (Figure 4.14). The solution to this problem was, for lack of a better description, applying a "flat topped barrel vault" mask (Figure 4.15)

with the profile derived from a Hanning window with an intervening flat spot at the apex to span the area of interest in the frame while suppressing the extraneous residual background. The orientation of the mask was determined by the program based on the velocity vector of the convoy. When the mask was applied, the tracking gate was applied to the proper objects and the tracking successfully accomplished (Figure 4.16). The orientation of the mask was determined by the difference between the x velocity vector and y vector. A simplifying assumption was made to the effect that the convoy was either going to be going roughly vertically or that it would be moving more or less horizontally relative to the frame and that orienting the mask appropriately would sufficient for proof of principle. As it turns out, the assumption proved effective.

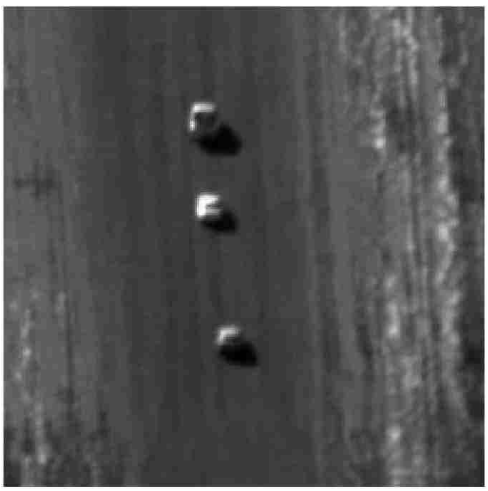

Figure 4.12 Original convoy sequence

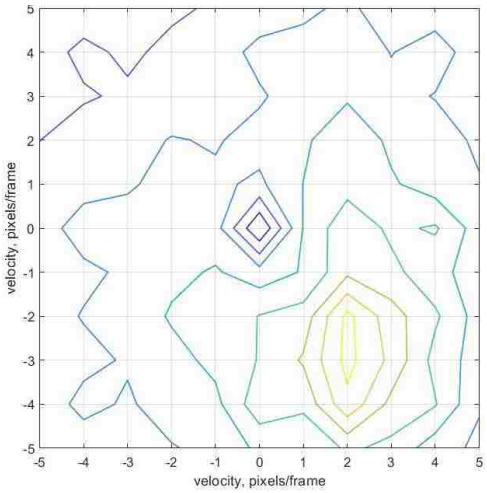

Figure 4.13 Motion map of the convoy

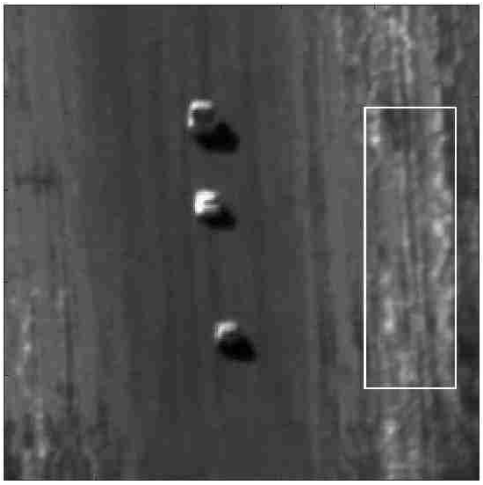

Figure 4.14 Initial result (without mask)

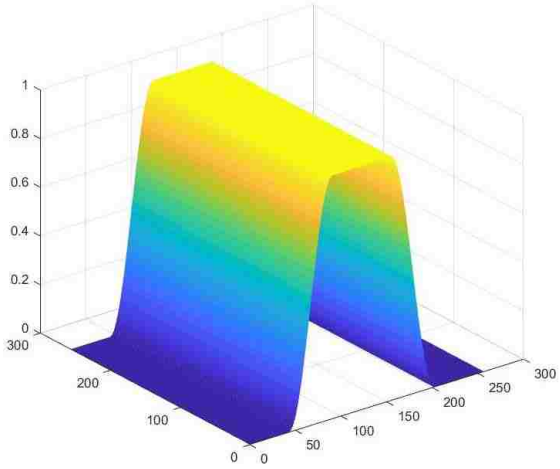

Figure 4.15 Flat topped barrel vault mask

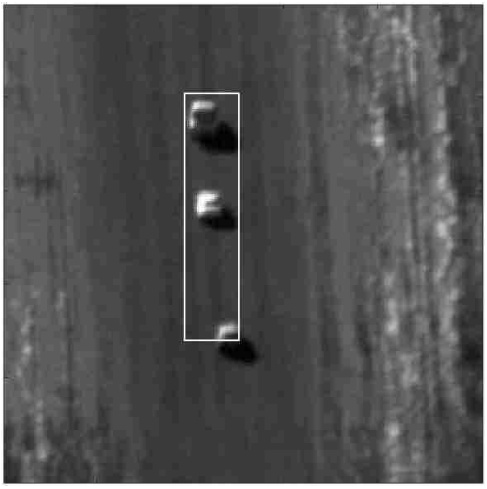

Figure 4.16 Final result (with mask)

4.3.2 Ridge Running

Often, a video sequence will have more than one moving object (aside from the moving background). In such cases, each of the moving objects will generally have their own readily discernable maximum in the motion map. However, when several vehicles are going in the same direction and at similar velocities, such as in a passing event, the maxima will be close together, perhaps even adjacent. Throw in noise and the ground motion, and sometimes a "ridge" in the motion map will arise masking the individual peaks. In Figure 4.17, presents again the scene from 4.9a, a Humvee pulling a trailer passing an M35. The resulting motion map in Figure 4.18 (same as 4.9b) clearly shows this situation. There is a main peak and right next to it is a discernable ridge. To find any of the objects masked by the ridge, one must go point by point along the ridge to find them. The main peak is at (-2, 2) and is unambiguous. Extracting the image at the indicated location yields Figure 4.19, which is clearly the passing Humvee. The map also shows a ridge from (0, 2) to (1, 1), ending at (2, 0). Investigating the ridge, we start with the left local peak at (0, 2). The image extracted at that point, Figure 4.20, shows fuzzy images of both vehicles. Enough of both vehicles leaked into the "in between" plane to make a weak appearance. Figure 4.21 shows the image extracted from (1, 1) which is clearly the M35 being passed. This is not a discernable peak, other than this point being on the ridge itself, as it sits in the saddle between the ends of the ridge. Finally, the image extracted from (0, 2) is given in Figure 4.22, where we have another case where we have fuzzy images of both vehicles. The image of the Humvee is fuzzier than the image of the M35 due to the fact that this point is further away from the optimal point for the Humvee than is the truck's optimal point.

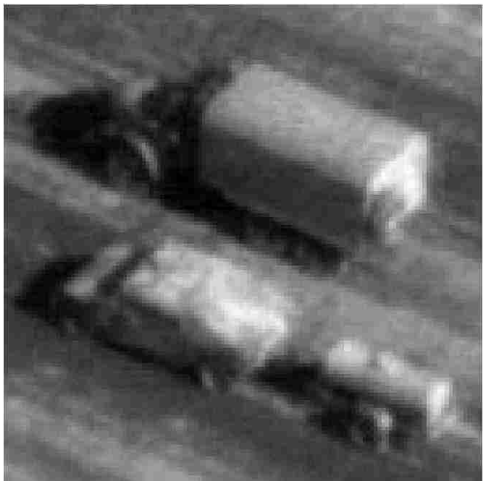

Figure 4.17 M35 Truck being passed by a Humvee with trailer

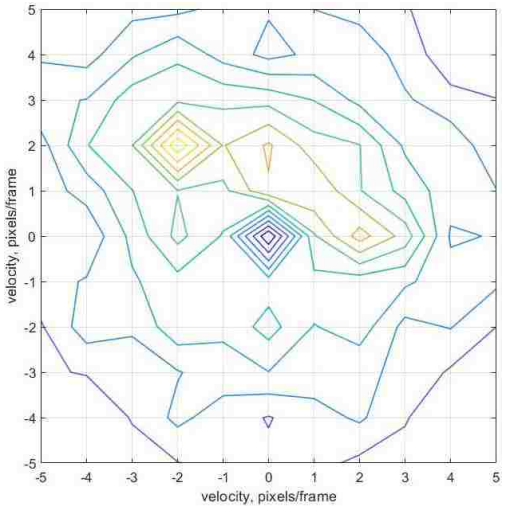

Figure 4.18 Motion map of passing event

Figure 4.19 Image extracted at (-2, 2)

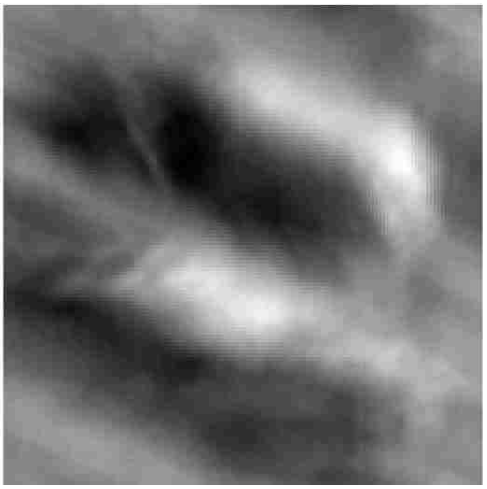

Figure 4.20 Image extracted at (0, 2)

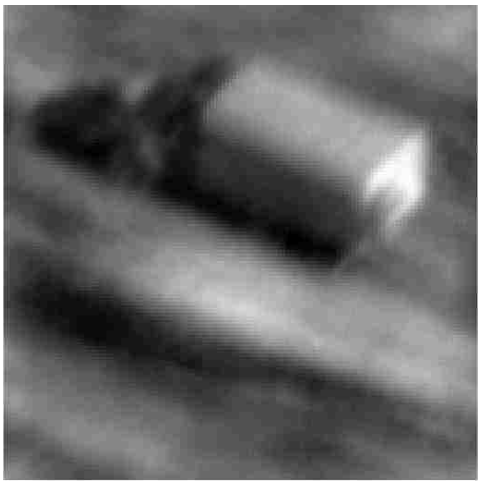

Figure 4.21 Image extracted at (1, 1)

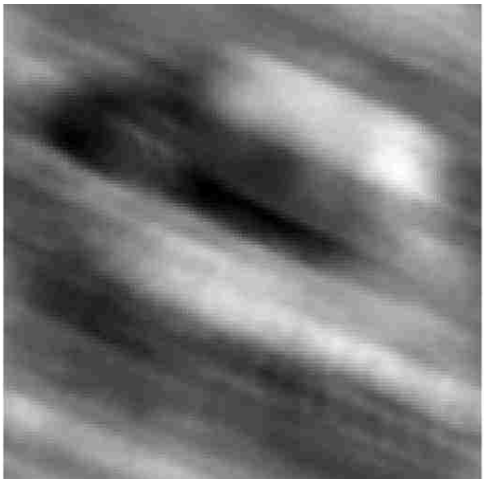

Figure 4.22 Image extracted at (2, 0)

4.3.3 Expanding the Filter

The extraction process consists of determining the plane on which the information for the object's image resides, and then using that information to create a mask to eliminate the extraneous data and allow the recovery of the best possible image of the object of interest. The mask forms a spatial filter in the frequency domain. When the perfect test images generated by the truck on the meadow were extracted from the spectral volume, for example, the results were sharp and clear. However, real data is not perfect, with the vehicle not moving at exactly an integral velocity, focus and platform vibration issues, and the presence of noise in the data resulting in a "fuzzy" image. It was speculated that expanding the mask to include adjacent planes might improve the quality of the extraction. This conjecture is not quite in the mainstream of this research, but it is interesting and closely enough related to warrant a few trials. Two cases were tried to obtain a preliminary result to see if there was any positive outcome – one trial with the four-neighbors and one with the 8-neighbors, the trials were performed by modifying the extraction software to construct the planes represented by the aforementioned neighbors and then including them in the mask. The results can be seen below. Using the data from the convoy trials, we start with Figure 4.23, the extracted M35, which is the baseline image. The extraction routine was then modified to include planes represented by the 4-neighbors in its filter. The resulting image, Figure 4.24, is not visibly improved. The routine was further modified to include the planes from the 8-neighbors. The result, Figure 4.25, also shows no improvement, leading to the conclusion that expanding the filter to include the neighboring regions is not effective in improving the extracted image.

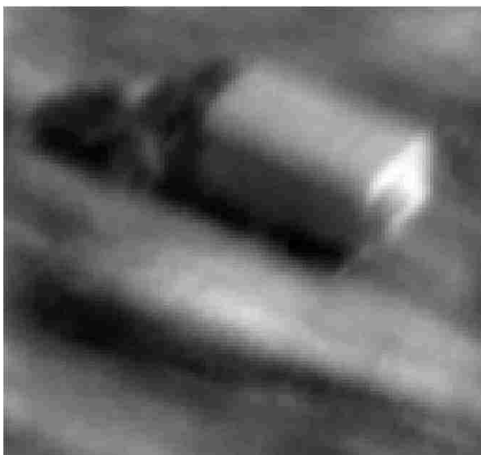

Figure 4.23 Original extracted image

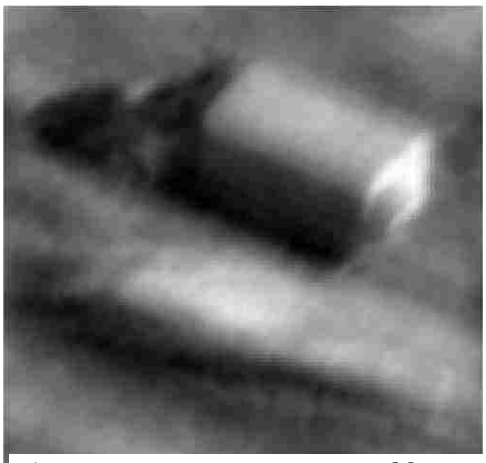

Figure 4.24 Image augmented by 4-neighbors

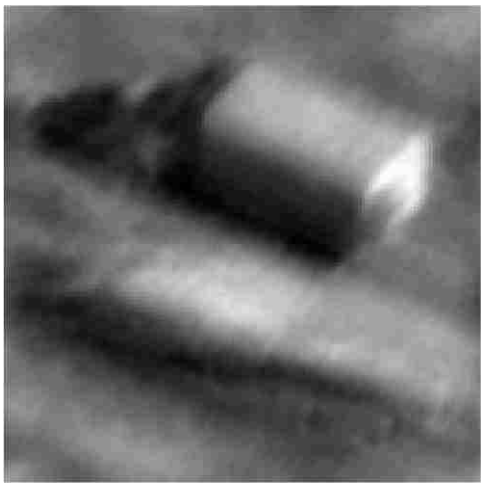

Figure 4.25 Image augmented by 8-neighbors

Chapter 5

Tracking Accuracy Comparison

An experiment was designed to verify the assumption of short-term linearity and accuracy of the algorithm and its associated tracking method. The plan was to track and apply boxes to longer sequences of vehicle motion using sequential 16 frame snippets (approximately ½ second each at 30 frames per second) and compare the results to the results from a well-regarded algorithm by Paul Rodriguez and Brandt Wahlberg called incremental Principal Component Pursuit (iPCP) [9, 10]. The dataset used consisted of long single car sequences from the VIVID database described in Chapter 4, augmented by data taken by a drone hex helicopter. The data set is show in Figures 5.1 – 5.5.

Figure 5.1 Car on Road 1 – V3V100003_005.avi from DARPA VIVID; frame size: 128x128 pixels. See Figure 5.7.

Figure 5.2 Car on Road 2 – V4V200005_022.avi from DARPA VIVID; frame size: 256x256 pixels. See Figure 5.8.

Figure 5.3 Car on Road 3 – V4V100013_053.jpg from DARPA VIVID; frame size: 128x128 pixels. See Figure 5.9.

Figure 5.4 Hexcopter Clip 1- Earthmover; frame size: 256x256 pixels. See Figure 5.10.

Figure 5.5 Hexcopter Clip 2- Car in parking lot; frame size: 448x128 pixels. See Figure 5.11.

An automatic object following program using the preceding routines was under development at the time this test was proposed, and it was modified to perform this comparison. This code is provided below in Appendix B. No pseudocode will be provided due to its complexity.

IPCP is robust and well behaved. As mentioned in Chapter 1, iPCP separates the video frame into background and foreground regions. As with the automatic tracking program, some minor modifications were made to allow access to internal data structures and

frame-by-frame results so a direct comparison could be made. The results from both algorithms would then be compared to ground truth data done by hand. The results for each method were plotted on the same graph for a given sequence, and the results for all the examples were combined and tabulated. As neither algorithm was originally intended to do object tracking, this is as fair a comparison of performance as possible that could be devised. Copyright issues preclude the publication of the modified iPCP code; it is available at the authors' web site.

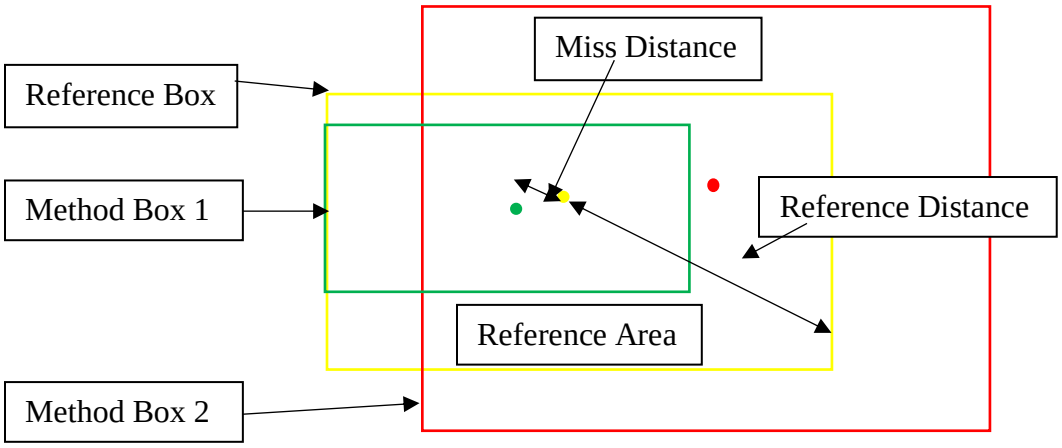

Figure 5.6 Scoring criterion for the comparison.

A very simple-minded scoring method was devised to compare the efficacy of the two methods. Code for this program is also given in Appendix B. As illustrated above in Figure 5.6, the result of each method on a sample sequence frame was used to generate a box on the frame, as in the synthetic tracking examples above. These boxes were then compared to a reference box inserted by hand. Scoring proceeds as follows. The area of the reference box is used as the standard for area and accuracy comparisons for that run. Area of the method boxes were compared to the reference box by forming a ratio of the

areas. Using this method always yields a positive, unambiguous result, unlike a simple arithmetic difference. That is,

$$Normalized\_Area\_Difference = Area\_of\_Method\_Box / Area\_of\_Reference\_Box. \quad (5.1)$$

Likewise, half the length of the diagonal of the reference box was used as a normalizing distance to determine the "miss distance" of the center of the method box to the center of the reference box, that is,

$$Ref\_Dist = \tfrac{1}{2}(Diagonal\_of\_Ref\_Box). \quad (5.2)$$

$$Miss\_Dist = norm(Method\_Box\_Center, Ref\_Box\_Center) / Ref\_Dist \quad (5.3)$$

By not using absolute areas and distances, the results of one run could be directly compared to another in terms of the common criterion. Thus, if a given method was working perfectly, it would generate a box that was centered on the reference box (zero miss distance) and was the same size as the reference box (size ratio of one). Of course, that never happens. The justification for the box size score is that it is an indirect measure of the quality of the extracted/reconstructed image. That is, if it is sharp, the box will closely conform to the shape of the original object, while if it's diffuse, it will be larger, and perhaps off center as well. Of course, the "boxing" algorithms used on either set of data could always be improved, but those that were used are probably good enough to give an informed comparison.

In the case of the example in Figure 5.6, the blue reference box is given with two other boxes, representing the competing methods superimposed on it. Method 1 represents a reconstruction where the "bright" spot being detected by the algorithm is smaller than the actual object, and the "miss distance" is less than the reference distance. As the box is

smaller than the reference box and fairly well centered, the result is that both scores are less than 1. Method 2 represents a reconstruction where the bright area is more diffuse, and, thus larger than the original. The box's miss distance is also less than the reference distance. The scores for this example would be less than one for the miss distance and greater than one for the box area. Remember, "perfection" would be a score of 1 for the size ratio and 0 on miss distance.

Results of real data comparisons are given below in Figures 5.7 through 5.11. Various sequences with cars on the road were analyzed with total sequence lengths of 2 to 3 seconds. A sliding interval of 16 frames was used, as indicated above. A reference sequence for each was manually generated and the results of the sequence as processed by the iPCP and FDM methods were compared to the reference and the results plotted. Sample fames from each video sequence is also presented with the reference, FDM, and iPCP generated boxes superimposed on the original object.

Figure 5.7a Car on Road #1, Frame 22

Figure 5.7b Image produced by iPCP segmentation model

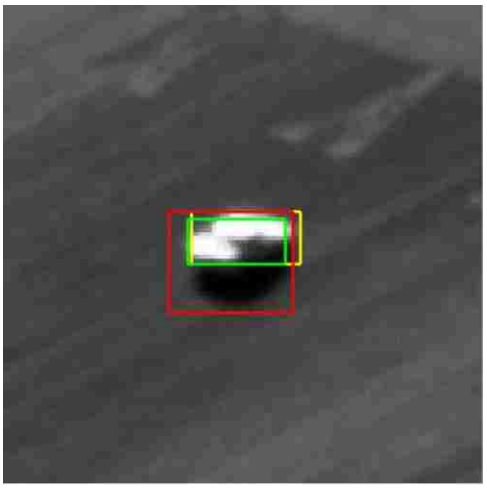

Figure 5.7c Original frame 22 with tracking boxes added for comparison

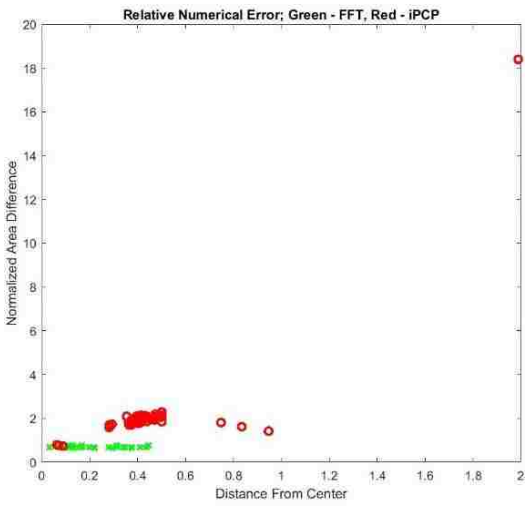

Figure 5.7d Relative error plot for Car on Road #1

Starting with Figure 5.7a, frame 22 of the Car on Road #1 sequence is presented, along with the image produced by the iPCP in Figure 5.7b. Figure 5.7c shows the color coded tracking boxes applied to the original frame 22 image to show how close the methods came to the position and size of the boxes placed by hand. This sequence is extracted from VIVID sequence V3V100003_005.avi, 0:48 to 0:53 seconds, frames 17-79 (32-79 for the FD method) with a frame size of 128x128 pixels. The iPCP method requires 15 frames to "prime the pump" with data, then the rest come out of the pipeline, hence the extra 15 frames to process the same number of frames as the FD method. The yellow box is the manually placed ground truth box, green is the FDM generated box, and the red box is the one placed using the iPCP data to generate the box. Figure 5.7d is a plot of the scoring data. Here, except for the one outlier in the iPCP score which forced the scaling to be so large, the results can be seen to be fairly close, with the FDM generated boxes edging out the iPCP results. This also shows that under favorable circumstances, both methods can produce similar results.

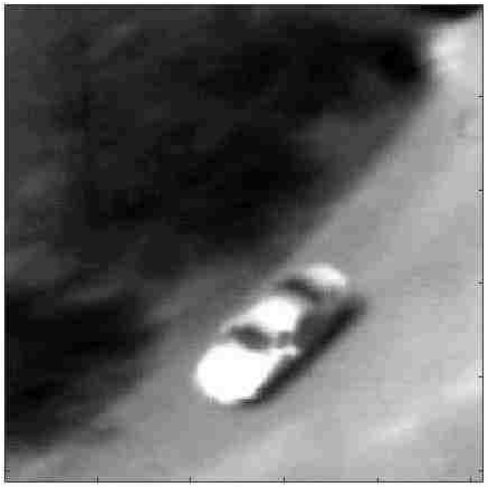

Figure 5.8a Car on Road #2, Frame 33

Figure 5.8b Image produced by iPCP segmentation model

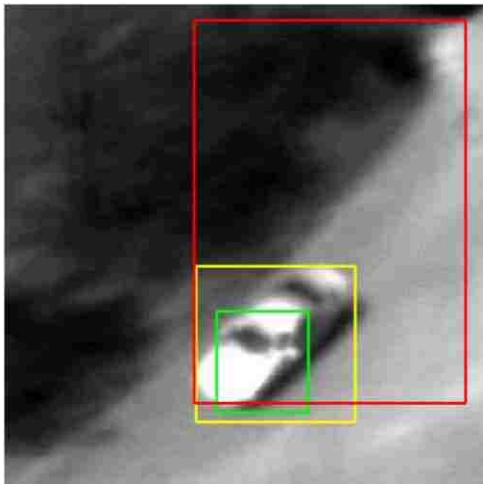

Figure 5.8c Original frame 33 with tracking boxes added for comparison

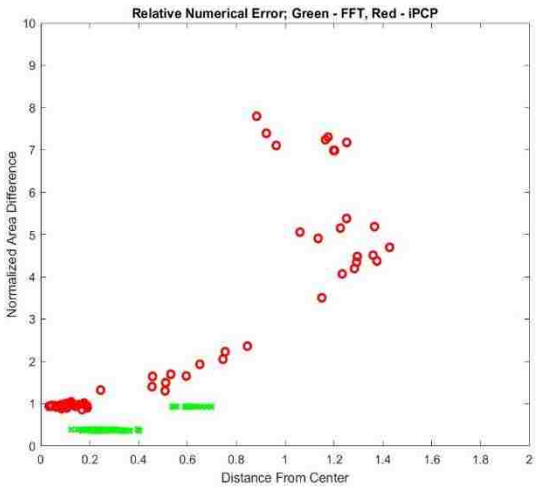

Figure 5.8d Relative Error Plot for Car on Road #2

Figure 5.8a shows the Car on Road #2 sequence; Figure 5.8b, as before, shows the iPCP model generated image, 5.8c the original frame 33 with the tracking boxes applied, and Figure 5.8d its error plot. This sequence is from VIVID sequence V4V200005_022.avi, seconds 0:00 to 0:04, frames 1-79 (16-79 for the FD method), with a window of 256x256 pixels. The results are not as good for the iPCP method as it was before, as can be seen in Figure 5.8c, considering the misalignment of the red iPCP box with the yellow ground

truth box. Bright spots dragged into the foreground by iPCP, seen in the upper right corner, snatch the block placing routine's attention and fool it into generating a larger box which is not always close to being coincident with the ground truth box

Figure 5.9a Car on Road #3, Frame 30

Figure 5.9b Image produced by iPCP segmentation model

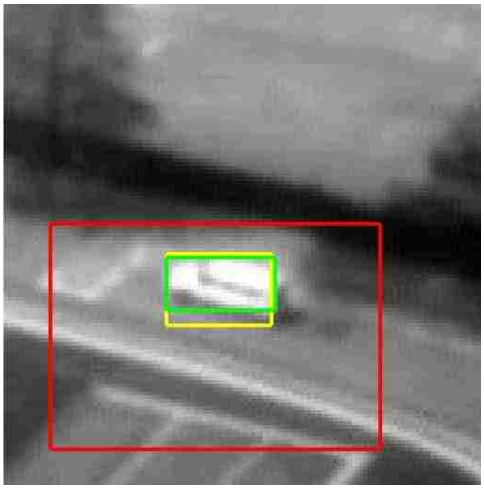

Figure 5.9c Original frame 30 with tracking boxes added for comparison

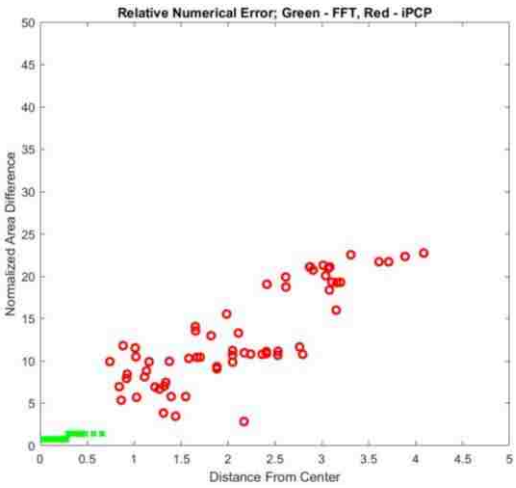

Figure 5.9d Relative error plot for Car on Road #3

ar on Road #3 is a sequence taken from VIVID file V4V100013_053.jpg, 1 minute, 41 seconds to 1 minute 43 seconds, frames 1 to 79 (16-79 for the FD method), with a window size of 128x128 pixels. The white van is flanked by light colored curbs and sidewalks which caused difficulty to the iPCP method, as it included much of the bright background clutter in its foreground rendering. The results are evident in Figures 5.9a through c. The red iPCP box is considerably larger than it ought to be, grabbed as it was by the light colored curbs, and it is clear from the plot that the centers of the iPCP boxes are unstable in relation to the center of the ground truth box.

The next two sequences were obtained by a fellow classmate, Jeffery Love, using a drone hexcopter with a video camera. The sequences consist of an earthmover driving through the brush by the Rio Grande, the other an SUV negotiating its path through a parking lot. Both videos are of excellent quality – the GoPro camera being superior to the cameras flown a decade ago by DARPA. The videos have come challenging issues though with terrain clutter, as will be discussed.

Figure 5.10a Hexcopter Clip 1, Frame 33

Figure 5.10b Image produced by iPCP segmentation model

Figure 5.10c Original frame 33 with tracking boxes added for comparison

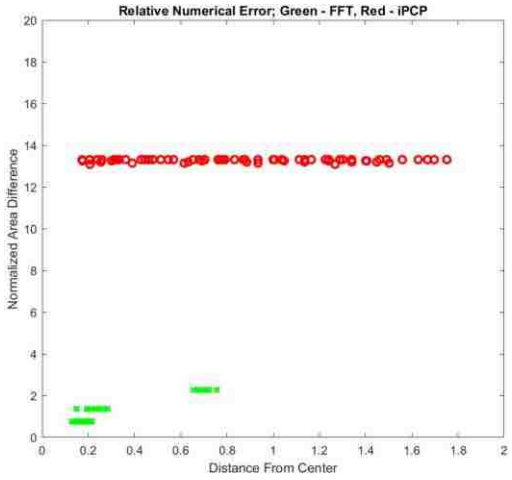

Figure 5.10d Relative Error Plot, Hexcopter Clip 1

Figure 5.10a is an earth mover driving through the brush by the river while doing erosion mitigation. It is flanked by reeds and rushes as commonly found on a river bank. This environment presented the most challenging background encountered in these trials. As can be seen in Figure 5.10c, the red iPCP box is fully expanded to the limits of the frame, which indicates that the weeds leaked into the foreground with sufficient energy to steal the box from the earth mover. The FD method had trouble with the earth mover as well, as it was an extended object with three bright spots sufficiently separated from each other to force the boxing software to choose the brightest and closest two spots (the cab and the front of the bed) to be boxed, leaving the rear of the bed out of the box, but even so, it did successfully capture most of the object. The result is that the size of the box was simply stuck on essentially the whole frame. Figure 5.10d records the analysis.

Figure 5.11a Hex Copter Clip 2, Frame 40, original, iPCP, and boxed

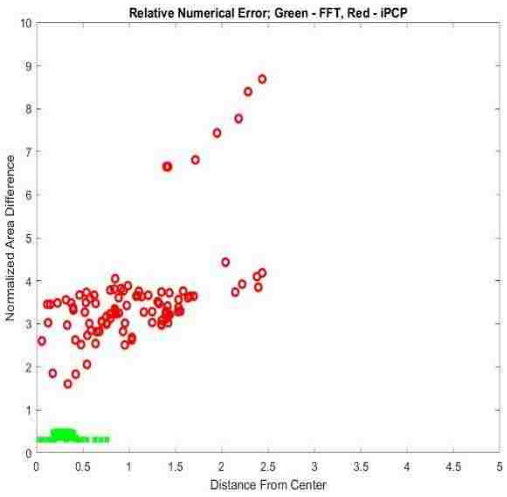

Figure 5.11b Relative Error Plot, Hex Copter Clip 2

The second hexcopter sequence is seen in Figure 5.11a and b. Here, the white SUV is driving away while the drone is loitering overhead. The challenges to both the FD method and the iPCP method are the trees to the right and the parking lot to the left. The segment from the file is from 2 minutes, 6 seconds to 2 minutes, 10 seconds. The segment is 111 frames long (96 frames for the FDM, giving 15 frames to get the iPCP started), with a window of 448x128 pixels. The unusual window was chosen to reduce the interference from the trees and parking lot to give both methods the best chance of properly functioning properly. At this juncture, the point is to compare the tracking ability.

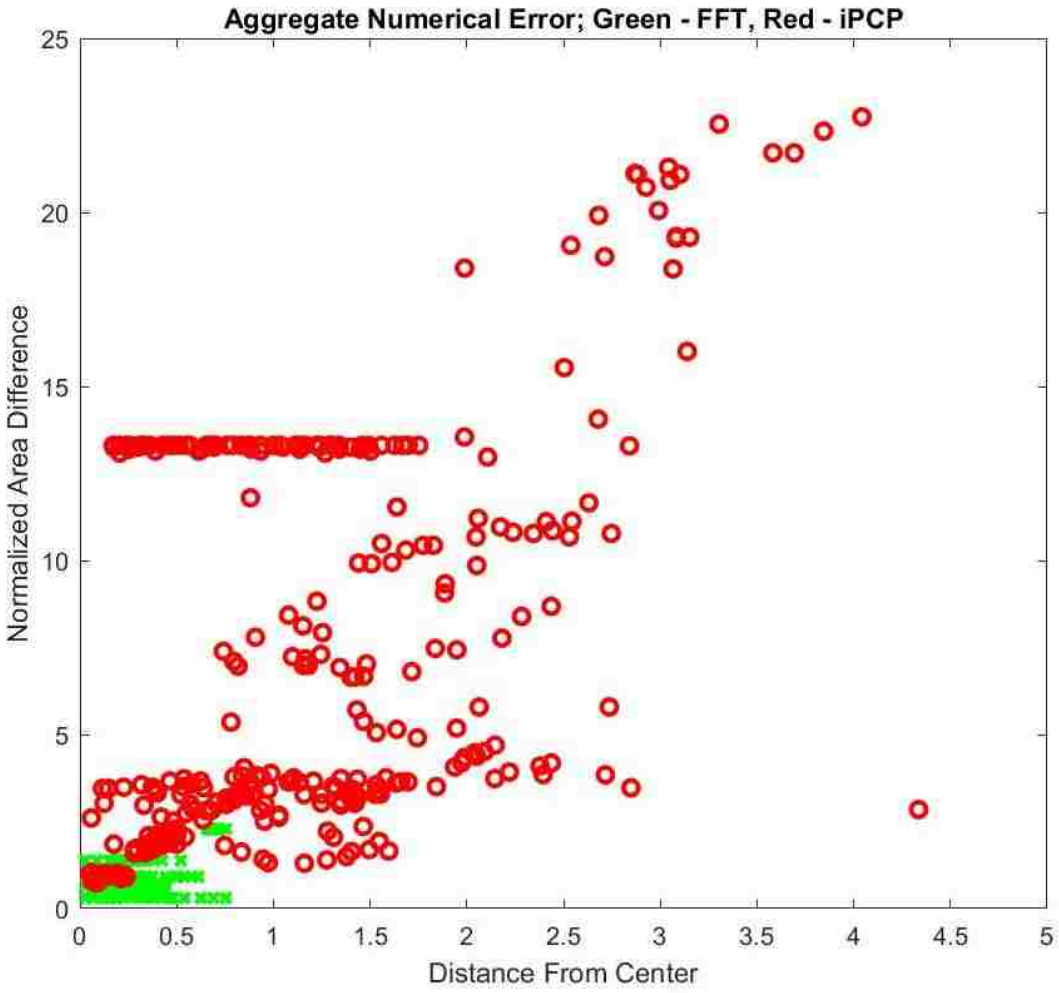

Figure 5.12 Aggregate Error - All Data

Putting it all together, the results are charted on a common graph in figure 5.12. The green representing the FD method is concentrated around the (0, 1) point of "perfection." Meanwhile, the red iPCP data is scattered all over. There are some iPCP successes, such as "Car on Road #1," but the iPCP was consistently outperformed by the FD method.

The numerical results are shown below in Table 3, where the aggregate results confirm the results from the graph. With 336 frames of video, the average normalized miss distance is 0.2593 with a variance of 0.0224, while the iPCP yields a normalized miss

distance of 1.1329 and a variance of 0.7480. Likewise, the normalized size estimate difference is 0.7340 with a variance of 0.2042 for FDM versus 6.8108 and variance of 32.6233. This means that the FDM method was doing approximately 4 times better at miss distance, versus the iPCP, and approximately 9 time better than iPCP in size estimate.

Table Three. Aggregate Numerical Errors – Comparison Tests

EVENT		**MEAN MISS DISTANCE**	**MISS VARIANCE**	**MEAN SIZE DIFERENCE**	**SIZE VARIANCE**	**NUM. FRMS.**
CAR ON ROAD #1	FD METHOD	0.1673	0.0138	0.6880	0.0014	48
	iPCP METHOD	0.4525	0.0735	2.2048	5.8076	
CAR ON ROAD #2	FD METHOD	0.2455	0.0162	0.5120	0.0590	64
	iPCP METHOD	0.8197	0.5319	2.6163	5.1548	
CAR ON ROAD #3	FD METHOD	0.2183	0.0097	0.9070	0.0728	64
	iPCP METHOD	2.3343	0.6642	12.8362	32.4922	
HEXCOPTER CLIP 1	FD METHOD	0.3174	0.0498	1.2919	0.3958	64
	iPCP METHOD	0.8668	0.2129	13.2751	0.0037	
HEXCOPTER CLIP 2	FD METHOD	0.3032	0.0123	0.4176	0.0042	96
	iPCP METHOD	1.0584	0.3413	3.5836	1.4926	
AGGREGATE RESULTS	**FD METHOD**	**0.2593**	**0.0224**	**0.7340**	**0.2042**	**336**
	iPCP METHOD	**1.1329**	**0.7480**	**6.8108**	**32.6233**	

Chapter 6

Summary and Possible Future Research

6.1 Summary

For videos characterized by moving backgrounds and multiple moving objects, the proposed frequency-domain methods delivered improved performance over time domain methods. The algorithm delivered especially well with clean, unambiguous examples where clutter was not an issue and the focus was good, but performed in a reasonable manner under most other circumstances. Where the methods did not perform adequately, it was clear that circumstances beyond the scope of the methods intervened to reduce the effectiveness of the analysis. It must be pointed out these data sets, especially the VIVID examples, are difficult datasets. DARPA did not design this set to test motion detection and quantification techniques, but rather to test activity detection algorithms. Even so, the results were convincing. Furthermore, it is clear from the results of the comparison in Chapter 5 that the FD methods are clearly superior in segmenting and following the object of interest as opposed to the iPCP method that was not designed to deal with background motion.

The success with drone hexcopter imagery points to the fact that technology has surpassed the VIVID 640x480 cameras. The current and future generations of drone videos will have stability augmentation, semiautonomous control, and high resolution, autofocusing cameras which will and do provide video without the platform issues. However, the backgrounds will still move, and there will often be multiple moving objects. This makes the application of these methods even more effective, as demonstrated in sections 5.10 and 5.11.

6.2 Future research

The FDM described in this dissertation are related to current UNM research on 2-D AM-FM models and methods described in [25-35]. The lack of a large dataset calls out for more development with a more expansive dataset for this algorithm. The proposed methodologies will have to be verified on large video databases before they become more widely used. As an example, at UNM, the AOLME project aims to extend video analysis methods in 1000 hours of videos (e.g., see earlier methods in [36-41]).

In the modern era of Big Data (e.g., [20]), it is important to develop methods that allow for very fast processing of large scale video databases. Such methods are described in [21, 22] based on the Discrete Periodic Radon Transform while FPGA implementations of general filter banks where developed in [22, 23, 24]. The FDM presented in this dissertation benefits from hardware assists, especially with parallel processing. It no doubt can benefit from an infusion of even higher performance hardware to increase its utility. It is conceivable that an external hardware implemented accelerated version (perhaps using an FPGA or a graphics processor) might allow near real time processing that keeps up with 30 frame/second video, opening up a wider array of possible uses of the algorithm and associated methods.

A paper based on this work will be submitted to EURASIP Journal on Image and Video Processing.

APPENDICES

Appendix A

High Fidelity Motion Simulator

It was quickly apparent that a realistic motion simulator was needed to validate the algorithms as they were developed, as simple moving blocks were not completely satisfactory. A real vehicle on a real landscape was desired. A photograph of the Lower Geyser Basin on the west side of Yellowstone National Park had been taken on a family vacation a few years prior to this effort by the author's wife. It had all the desired characteristics being sought, so a copy was cropped to create a long strip of landscape. The image of an Army M-35 6x6 truck was also identified for use in this effort. The background image was converted to gray scale, as was the truck. The truck was then "cut out" of its picture, so that the vehicle was the only item in it. A program was written to move a window along in the background at a specified rate from left to right, which meant that the background appears to move right to left. The truck image is placed in the window also moving as specified generally right to left (it can be moved along not just left to right but diagonally). The net result is as if the observer was riding in a vehicle moving left to right looking out the left hand window with the landscape moving by right to left, as one would expect it to. The truck is generally going faster than the observer's "vehicle," so it passes by going left to right as if it were passing the observer. Multiple motions would then be present and the algorithm tested to see if it correctly captures them all. Figures A.1 through A.4 show the actual images used. The code enabling the simulation is given in Appendix B.

Figure A.1 Lower Geyser Basin, Yellowstone National Park (photo: Virginia Stone)

Figure A.2 Yellowstone scene Cropped and converted to gray scale

Figure A.3 M-35 6x6 truck

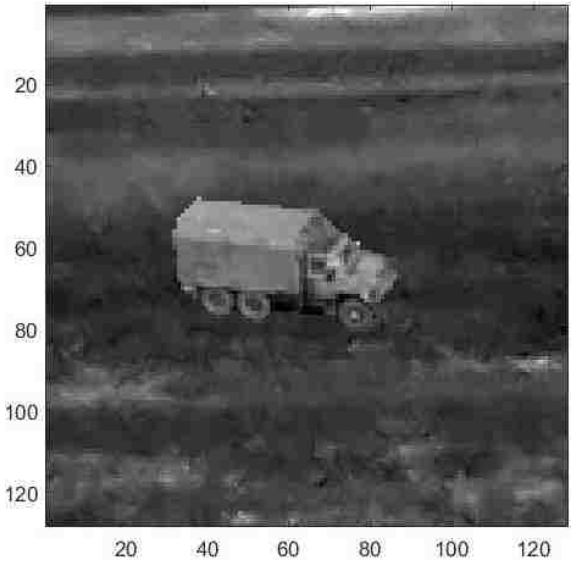

Figure A.4 Resulting composite scene presented by the simulation

Appendix B

Computer Programs Created for the Work

B.1 Motion Mapping Function, Parallel and Accelerated Version

```
function [Answer] = fastmappmex(dft, range, resolution) %#codegen
% This routine maps the motion for the analysis of motion in video
% sequences.  It features variable range and resolution, which will be
% determined by the parameters provided by the user.  The actual compiled
% version to be invoked is fastmappmex_mex.
% Inputs
%       dft – magnitude spectrum of the video sequence
%       range – span of the velocity search in –span to +span in x and y
%       resolution – used with Z transform for sub pixel/frame motion
% Output
%       Answer – 2-D array holding the motion map
%
% © V. M. Stone 2018
%
% assert the input variables; required to compile this routine
assert (isa (dft, 'double'));
assert (all (size(dft)<=[1024 1024 160]));
assert (isa (range, 'double'));
assert (isa (resolution, 'double'));
% The scaling of the result is determined here; this stays the same as
% before...
extent = 2*(range/resolution)+1;
Answer = zeros (extent, extent);  %this is where the answer goes
midp = ceil (extent/2);
% Set the threshold.
thresh = 0.1;
% Get the size of the incoming DFT.
[r, c, p] = size (dft);
% We use the temporal positive half; we'll chuck the negative half, of the
% spectrum.  I suppose we could doctor the code to just ignore it, but then
% gets insane.
% the indexing gets insane.
%F irst, figure out which plane is the "0Hz" plane
zhp = floor(p/2)+1;
% Isolate the positive half of the spectral volumes
spec = dft(:, :, zhp:p);
```

```
% Next, we need the zero points of the planes.
zx = floor(c/2)+1;
zy = floor(r/2)+1;
% We have isolated the half we want.  Eventually, we will ignore the temporal
% 0 plane, as well as the region around spatial zeros.
% While we're at it, let's redefine r, c, and p.  BE %CAREFUL!  r and c remain
% the same, but p is 1/2 of what it was...
[r, c, p] = size(spec);
%
% Build the scaling vectors, one for each axis.  They will have the same
% indexing as the points in the probe volume.  We need only 1/2 the values
% for z.
X = linspace(-pi,pi,c+1);
Y = linspace(pi,-pi,r+1);
Z = linspace(0,pi,p+1);
% We now build probe volumes for each trial, where the test plane is indicated
% by 1's.  This is done by exploiting the property that the dot product of a
% vector in the plane with the plane's normal is zero.  Because a vector
% from the origin to any point in the space is a vector in any plane that
% passes through that point, we can thus take the dot product of that vector
% and the velocity vector being examined and see what's going on.  We'll search
% the space point by point and mark the space with 1's where the product is
% sufficiently low, and 0 where it's not.  This should yield a plane defined
% in the probe space that can be anded with the 3d-fft to eliminate values
% not on that plane and allow for the testing process to succeed.
%
% figure out the size of the blanking factor – this is necessary if the spatial frequency
% space is too big
if r <= 16
   bf = 0;
elseif r <= 64
   bf = 1;
else
   bf = 2;
end
% Preallocate the transfer variable used in the probe process.
tran = zeros(r, c, p);
% run timer
%t1=tic;
% It is worth mentioning that when doing sub-pixel work, i and j are
% fractional.  They have to be translated for the algorithm...
parfor i=1:extent
%    display(i);
   for j=1:extent   %j and i map into the Vx's and Vy's, respectively, being searched for
```

```
      Vx = (j-midp)*resolution;
      Vy = (i-midp)*resolution;
      % Now, let's build a plane of 1's normal to Vx, Vy, and 1.
      % preallocate/reset the accumulator space
      zero_vals = zeros(r, c, p);
      % Do the dot products by brute force, but ignore the velocity pair (0,0).
      if Vx==0 && Vy==0
         Answer(i, j) = 0;
      else
         for m=2:p %remember, we're skipping the 0 hz plane
            for l=1:r
               for k=1:c
                  if ~(abs(zy-l)<=bf) || ~(abs(zx-k)<=bf)  %if too close to 0,0 then skip this
                     %Now, what happens with the "k" variable is that X is
                     %column, Y is row in non-MatLab thinking, so I have to
                     %exchange the column and row indices to make it work right.
                     zero_vals(l, k, m) = (abs(dot([X(k) Y(l) Z(m)],[Vx Vy 1])))<thresh;
                  end
               end
            end
         end
         %"and" it with the dft to eliminate the irrelevant stuff and
         %sum it up to get the intensity on that plane.
         tran = abs(zero_vals.*spec);
         Answer(i, j)= sum(tran(:));
      end
   end
end
% ok, how much time did it take?
%toc(t1)
```

B.2 Peak Detecting Function, Parallel Version

```
function [vel]=detectpeaksp(mapin,scale)
%
% The object of this code is to detect the peaks in an energy map
% and return the velocities indicated by the peaks.  It is possible that
% there are more than one peak.  We will ignore the edges of the map,
% as the data we seek supposed to be closer to the center
% Assume the map is symmetrical, at least in its layout.
%
%Inputs
%        mapin-      the input map
%        scale-      scale factor of incoming map (i.e., 1, 1/2, 1/4, etc. pix resolution)
% Outputs
%        vel-        velocities indicated by the peaks, and relative amplitude
%
% © 2018   V. M. Stone
%
% Create a structure to hold the captured peaks
[R,C]= size(mapin);
answer= zeros(R,C);
% Normalize the input map; this keeps us sane
mapin= mapin/max(mapin(:));
% Since it's normalized, the 90% point is above 0.9
thresh = 0.9;
% Let's walk through the map; ignore the edges; parfor invokes parallel processing
parfor i=2:R-1
   for j=2:C-1
      % This mess checks the 8-neighbors by brute force (my favorite way)
      if ((mapin(i, j) > mapin(i-1,j-1)) && ...
         (mapin(i, j) > mapin (i-1,j)) && ...
         (mapin(i, j) > mapin (i-1,j+1)) && ...
         (mapin(i, j) > mapin (i,j-1)) && ...
         (mapin(i, j) > mapin (i,j+1)) && ...
         (mapin(i, j) > mapin (i+1,j-1)) && ...
         (mapin(i, j) > mapin (i+1,j)) && ...
         (mapin(i, j) > mapin (i+1,j+1)))
         answer(i, j)= mapin(i, j);
         if (answer(i, j) < thresh)
            answer(i, j)= 0;
         end
      end
   end
```

```
end
% Let's determine the scaling of the map; assume it's symmetrical
B= floor(C/2);
B= B*scale; % This adjusts for subpixel/frame resolution, if being used
% Now we set up the actual "ruler" to relate the index to the actual value
ruler= linspace(-B,B,C);
% Let's set up to identify the velocities
numpeaks= 0;
% How about a 3 by n array to hold the velocities and associate intensity?  Now for a
% walk through the 2-d array and find the detected peaks...
for i=2:R-1
   for j=2:C-1
      if answer(i, j) ~= 0
         numpeaks= numpeaks+1;
         vel(1,numpeaks)= ruler(j); %x is in column space
         vel(2,numpeaks)= ruler(i); %y is in row space
         vel(3,numpeaks)= answer(i, j);
      end
   end
end
vel = (sortrows(vel',-3))';
% We're done
```

B.3 Chirp-Z 2-D Plus

B.3.1 Forward chirp-Z transform

```
function [B]=chirpz2dplus(A, Res)
%Here, we want to increase the resolution of the spatial frequency part of
%the 3d transform.  We keep the temporal axis the same (i.e., use a
%simple fft instead of the czt).  We'll keep the spacing even just to
%preserve our sanity.
%
%Input
%       A – video segment
%       Res – resolution factor ( 1/2, 1/4, 1/8 etc.)
%Output
%       B – complex spectrum, unshifted, as it is with the FFT
%
% © V. M. Stone    2018
%
%turn resolution into size multiplier
M=1/Res;
%figure out the size of the input and define the output stack
[R,C,P]=size(A);
B=zeros(R*M, C*M,P); %we'll start with an appropriate workspace
%first scratch space
B1 = zeros(R, C*M,P);
%second scratch space
B2 = zeros(R*M,C*M,P);  %we need working scratch for the intermediate results
%we'll use the output variable as the third scratch space
%evaluate the coefficients for the czt
a=1;   %even spacing, supposedly
w = exp(-j*2*pi/(M*R));  %assumptions: R=C (square array) so we can use this
%do the columns
for k=1:P  %We have to cycle through all the planes...
   for i=1:R %and then we cycle through each row in each plane, storing in B1
      B1(i,:,k)= czt (A(i,:,k),R*M, w, a);
   end
end
%then we next do the columns of the first result, putting the second result in B2
for k=1:P  %again, we have to cycle through all the planes...
   for i=1:C*M %and then we cycle through each column in each plane, storing in B2
      B2(:,i, k)= czt (B1(:,i, k),C*M, w, a);
   end
end
%and on to the vertical axis of the second result (the time axis) into B3.
```

```
%We have to index each point on, say, the bottom spatial plane and do each
%vertical column, storing it in B.
for k =1:C*M
   for I =1:R*M
      B(i, k,:) = fft (B2(i, k,:));
   end
end
%done
```

B.3.2 Inverse Z transform

```
function [B] = invchirpz2dplus (A, Res)
% This will compute the inverse of the chirp-Z 2-D plus transform.  As a
% reminder, the temporal axis is 1 to 1 same (i.e., use ift instead of the iczt).
%
% Input
%        A – a chirpz2dplus spectrum
%        Res – the resolution (1/2, 1/4, 1/8, etc.)
% Output
%        B – the spatiotemporal video stack
%
% ©   V. M. Stone    2018
%
% turn the resolution into size multiplier
M = 1/Res;
%figure out the size of the input and define the output stack
[R, C, P] = size(A);
B = zeros(R*M, C*M, P); %we'll start with an appropriate workspace; we're not
%expanding or contracting the temporal axis
B1 = zeros(R, C*M,P);
%evaluate the coefficients for the czt
a = 1;   %calls for even spacing
w = exp(j*2*pi/(M*R)); %assumptions: R=C (square array) so we can use this; this also
                       %makes it do the inverse transform
%next, do the columns
for k = 1:P  %We have to cycle through all the planes...
   for I = 1:R %and then we cycle through each row in the plane, storing in B1
      B1(i, :, k) = czt(A(i, :, k),R*M, w, a);
   end
end
B2 = zeros(R*M, C*M, P);  %we need working scratch for the intermediate results
%then we next do the columns from the first result, putting the second result in B2
for k=1:P  %again, we have to cycle through all the planes...
   for i=1:C*M %and then we cycle through each column in the plane, storing in B2
```

```
      B2(:, i, k)= czt(B1(:, i, k),C*M, w, a);
   end
end
clear B1;
%and on to the vertical axis of the second result (the time axis)
%We have to index each point on, say, the bottom spatial plane and do each
%vertical column
for k=1:C*M
   for i=1:R*M
      B(i, k, :)=ifft(B2(i, k, :));
   end
end
%fix the scaling
 [r, c, p]=size(B);
B=B/(r*c*p);
% done
```

B.4 Image Extraction

```
function [obj]=extract_object(vidin, xvel, yvel)
% This little number filters out the stuff NOT on the plane defined by xvel
% and yvel.  This essentially extracts the object making the plane.
% Notice: this version does not do a chirp-Z.  Modify it with chirpZ2dplus and
% invchirpZ2dplus if necessary. It will also need the scale factor passed in.
%
% Input
%        vidin – source video
%        xvel, yvel – velocity of object to be extracted
% Output
%        obj – extracted object
%
% ©   V. M. Stone   2018
%
% first order of business - get the fft, and keep it complex
fftin = fftshift(fftn(vidin));
[r, c, p] = size(fftin);
thresh = 0.2;
% we now build a space where the plane is indicated by 1's
% we'll do this using the property that the dot product of a vector in the
% plane with the plane's normal is zero.  We'll search the space point by
% point and mark the space with 1's where the product is sufficiently low,
% and 0 where it's not.  This should yield a plane defined in the space that
% can be essentially anded with the 3d-fft to facilitate the filtering
% process.
% build a plane of 1's normal to Vx, Vy, and 1.
X = linspace(-pi,pi,c+1);
Y = linspace(pi,-pi,r+1);
Z = linspace(-pi,pi,p+1);
zero_vals=zeros(r, c, p);
%Do the dot products by brute force,
for k=1:p
   for j=1:r
      for i=1:c
         %now, what happens with the "j" variable is I flip the thing
         %upside-down, as positive, here, is from top to bottom...
         if (abs(dot([X(i) Y(j) Z(k)],[xvel yvel 1])))<thresh
            zero_vals(j, i, k) = 1;
         end
      end
   end
```

```
end
%apply mask
fftout = (zero_vals.*fftin);
%get tne image stack back
vidout = real(ifftn((ifftshift(fftout))));
%fish out the image
obj = vidout(:,:,(p/2)+1);
```

B.5 Object Tracking

B.5.1 Main Tracking Routine, as Modified for Comparison Test

```
function [outimg,boxes]=follow_object_report_modified(inimg,objimg,xvel,yvel)
% This puts it all together.  We are going to find the image of the moving object,
% figure out where the box goes in the frame, and put it on the output image.
%
% Inputs
%       liming-      the original input video sequence
%       objimg-      extracted object image, normalized and histogram adjusted
%       xvel, yvel-  the velocity vectors, as detected by the peak detector
% Outputs
%       outimg-      the original video sequence with the boxes added
%       boxes-       box array, one column with upper left-hand corner's x (column),
%                    y (row), width and height, all in plot coordinates. For use by the
%                    scoring for the comparison
%
% ©   V. M. Stone  2018
%
% copy the incoming video segment into the output variable
outimg= inimg;
% get the size of the input segment
[R,C,P]= size(inimg);
% find the object in the frame; make a bounding box
BB = get_object_bounding_box(objimg);
%bias the first bounding box position with the velocity vector
BB(1) = round(BB(1)-(P/2)*xvel);
BB(2) = round(BB(2)+(P/2)*yvel);  %remember, yvel is up while y is down
%s and set up the box array
boxes = zeros(4, P);
% apply the boxes
for p=1:P
   %
   boxes(:,p) = BB;
   % Add the box; add_standard_box handles clipping for us
   outimg(:,:,p) = add_standard_box(inimg(:,:,p),BB);
   %move the box along
   BB(1) = round(BB(1)+xvel);
   BB(2) = round(BB(2)-yvel);  %remember, yvel is up while y is down...
end
% we're done
```

B.5.2 Create the Object Bounding Box

```
function [box] = get_object_bounding_box(obj)
% We're going to scrub the reigonprops array to figure out if there's any
% outliers to erase prior to constructing a convex hull around it
%
% Input         obj - a "cleaned up" image (normalized and adjusted)
% Output        box – bounding box (standard format)
%
% © V. M. Stone    2018
%
% take a cut at the binary image
B = make_bin_img(obj, 0.99);
% label the image
L = bwlabel(B);
% get the region properties
P = regionprops(L);
% how many regions?
NB = length(P);  %Number of Blobs
%
% There are generally three possibilities: it's a solid object (one
% blob), it’s predominately a single large blob with lot of clutter around
% it, or a really fragmented objects, with a few more or less
% medium blobs with irrelevant clutter nearby.  Let's see what we have.
%
if NB == 1 %There's only one blob; just use the bounding box.
   box = round(P(1).BoundingBox);
else %There's more than one blob. Generally, the largest blob is part of the
   % object of interest.  We'll start there and work down.
   % Sort the list by size of blob
   P = sort_blobs_desc(P);
   % Figure out the distance to be used as the distance metric. Let's use
   % the size of the bounding box, for lack of anything else
   box = round(P(1).BoundingBox);
   if P(1).BoundingBox(3) > P(1).BoundingBox(4)
      SD = P(1).BoundingBox(3);
   else
      SD = P(1).BoundingBox(4);
   end
   % We could sort by distance and then cut off the list at some distance,
   % or we could just look at the other blobs and keep the close ones and
   % dump the far away ones.  Let's use the second method, as it's easier.
   for i = 2:NB
      %Compute the distance to each of the other blobs
```

```
      D = nearest(sqrt((P(1).Centroid(1)-P(i).Centroid(1))^2 + ...
         (P(1).Centroid(2)-P(i).Centroid(2))^2));
      %If it's too far away, blob it out.
      if D > 3*SD %Dump the blob - it's too far away.  The multiplier is
         % heuristically determined (i.e., trial and error).
         cbox = round(P(i).BoundingBox);
         %black out the contents of the bounding box in the label image
         B(cbox(2):cbox(2)+cbox(4),cbox(1):cbox(1)+cbox(3)) = 0;
      end
   end
   % Throw a convex hull around what's left
   L = bwconvhull(B);
   P = regionprops(L); %there will be only one region- the convex hull
   % Collect the bounding box
   box = round(P(1).BoundingBox);  %done with this case
end
% Relax the box a bit
box(1) = box(1) - 2;  %expand the box by 4 pixels
box(2) = box(2) - 2;
box(3) = box(3) + 4;
box(4) = box(4) + 4;
% We're done
```

B.5.3 Add a Standard Box to a Video Frame

```
function [outframe] = add_standard_box (inframe, boxdef)
% Add a box to the input frame and put it out as the output frame.
%
% Input
%        boxdef - contains a MatLab standard box definition
%        inframe – frame to which the box will be added
% Output
%        outframe – the modified image
%
% © V. M. Stone   2018
%
%Copy the input into the output
outframe = inframe;
[R, C] = size(outframe);
%Unbundle the mins and maxs
ulr = round(boxdef(2)); %upper left row
if ulr < 1 %clipping at the edge of the frame, if necessary
   ulr = 1;
end
```

```
ulc = round(boxdef(1)); %upper left column
if ulc < 1
   ulc = 1;
end
lrr = ulr+round(boxdef(4)); %lower right row
if lrr > R
   lrr = R;
end
lrc = ulc+round(boxdef(3)); %lower right column
if lrc > C
   lrc = C;
end
%add in the box
maxpix = max(inframe(:));
if maxpix == 0
   maxpix = 1;
end
%do the sides
for i=ulr:lrr
   outframe(i, ulc) = maxpix;
   outframe(i, lrc) = maxpix;
end
%now the top and bottom
for i=ulc:lrc
   outframe(ulr, i) = maxpix;
   outframe(lrr, i) = maxpix;
end
%done
```

B.5.4 Alternate Convex Hull Applying Tracking Routine

```
function [outimg]=follow_object_hull(inimg, objimg, xvel ,yvel)
% We are going to find the image in the frame, compute a convex hull, and
% put it into the output image.
%
% Inputs
%       inimg-      the original input video sequence
%       objimg-     middle frame of the reconstructed object image
%       xvel, yvel- the velocity vectors
% Output
%       outimg-     copy of the original video sequence with the hull added
%
% © V. M. Stone   2018
%
```

```
% Normalize the image to 0-1 in intensity values.  This preserves our
% sanity and makes the other stuff work 100% of the time.
normimg= objimg/max(objimg(:));
% Adjust the grey scale to enhance contrast.
normimg= imadjust(normimg);
%make a binary image
binimg = make_bin_img(normimg, 0.95);
%trim the edges
[R,C]=size(binimg);
binimg(1:10, :) = 0;
binimg(R-10:R, :) = 0;
binimg(: ,1:10) = 0;
binimg(: ,C-10:C) = 0;
% Throw a convex hull around the binary image
hull= bwconvhull(binimg);
% get the hull properties
P = regionprops(hull, 'ConvexHull');
% Trace the hull; the point on the hull will be i, j.
j = round(P.ConvexHull(1,1));
i = round(P.ConvexHull(1, 2));
bound=bwtraceboundary(hull,[i  j], 'E');
% Put the hull on the output image, moving it as necessary.
% First, a little bookkeeping.
outimg= inimg;
[RI,CI,P]= size(inimg); %All I really need is P, the number of planes.
[R,~]= size(bound); %I do need the number of rows in bound; C is 2.
% Now, we set the bias of the boundary to put it into the output stack.
rowbias = floor(-(P/2)*(yvel));
colbias = floor(-(P/2)*(xvel));
% Set the intensity of the outline
bright= max(max(max(outimg)));
% Write it in.
for p=1:P
   %Do the current plane.
   for i=1:R
      r = floor(bound(i,1)+rowbias);
      c = floor(bound(i,2)+colbias);
      %clip, if necessary
      if (r < 1)
         r = 1;
      elseif (r > RI)
         r = RI;
      end
      if (c < 1)
```

```
            c = 1;
        elseif (c > Cl)
            c = Cl;
        end
        outimg(r, c, p)= bright;
    end
    %Update the bias factor.
    colbias= colbias+xvel;
    rowbias= rowbias+yvel;
end
% and we're done, I think
```

B.6 Code for the Comparison

B.6.1 Automatically Detect Moving Object, as Modified

```
function [out_vid, boxes] = auto_detect_moving_object_modified(in_vid, range)
% The operator is asked to manually box a moving object through a video clip, then
% this software attempts the same task.  The result will be compared by a scoring
% program.
%
% Inputs
%       in_vid – the input video
%       range – the expected velocity range
% Outputs
%       out_vid – boxed version of the input video
%       boxes – an array of specifications for each of the boxes
%
% ©   V. M. Stone    2018
%
% Get the frame size
[R, C, P] = size(in_vid);
% Preallocate space.
out_vid = in_vid;
boxes = zeros(4, P);
%
% Define block size and stride (i.e. overlap).
block = 16; %number of frames for each iteration
stride = 16; %stride is how far the analysis jumps ahead for each iteration
% This nibbles its way through the input video, finding and marking the
% object on the way
% First thing is to set the starting block
curstart = 1;  %let's start with the first segment
objstackptr = 1;
while (curstart+block-1)<=P
   % Cut out the sub-stack
   stack = in_vid(:,:,curstart:(curstart+block-1));
   % Make magnitude spectrum.
   rspec = abs(fftshift(fftn(stack)));
   % Build an energy map [Note: this routine needs the parallel processing
   % stuff turned on].
   map = fastmappmex_mex(rspec,range,1);  %this is the compiled version
   % Detect the peaks; assume the largest peak is the object. This ignores
   % the problem of two or more objects for now.
   peaks = detectpeaksp(map,1);  %peaks are sorted by intensity
```

```
    % Assume that the dominant peak is probably the object of interest
    xvel = peaks(1,1);
    yvel = peaks(2,1);
    %Do any necessary heuristics
    [xvel, yvel] = heuristics(map, xvel, yvel);
    %Extract the object.
    obj = extract_object(in_vid(:, :, curstart:(curstart+block-1)), xvel, yvel);
    % Normalize the extracted image to 0-1 in intensity values.  This preserves our
    % sanity and makes the other stuff work 100% of the time.
    obj= obj/max(obj(:));
    % Adjust the grey scale to enhance contrast.
    obj= imadjust(obj);
    % Follow object box; save the box locations frame-by-frame for comparisons.
    [outstack, boxstats]=follow_object_report_modified(stack, obj, xvel, yvel);
    % add outstack (the output video frame) into out_vid
    out_vid(:,:,curstart:(curstart+block-1)) = outstack;
    % add boxstats to boxes
    boxes(:,curstart:(curstart+block-1)) = boxstats;
    % update pointers
    curstart = curstart+stride;
end
```

B.6.2 Heuristics Routine

```
function [xvel, yvel] = heuristics( map, xvel, yvel)
% Heuristics examines the map for peaks adjacent to the main peak in the map.
% If it finds one, it will increment the velocity estimate as appropriate,
% as the integerized map tends to truncate the estimate and the subsequent
% processing misses the mark. This is more pronounced in real sequences than
% in synthetic ones.
%
% Input
%       map – incoming motion map
%       xvel, yvel – incoming velocities
% Outputs
%       xvel, yvel – adjusted velocities
%
% ©   V. M. Stone    2018
%
% Deduce the scaling of the map so we can do the heuristic.
% As of now, the map is square. It is also an integer system. This heuristic
% gets blown all to hell if we try this with fractional resolution.
[r, c] = size(map);
mapcen = ceil(r/2);
```

```
mapmax = r-mapcen;
mapscale = linspace(-mapmax, mapmax, r);
% Now that I know the map scaling, I can back out the neighbors to the max
% peak. Compute the index of the peak.
i = find(mapscale == yvel);  %first the row
j = find(mapscale == xvel);  %then the column
% Let's try a general heuristic
% Check the 8 neighbors. Is any of them a secondary peak?  It's always plus
% one in the current direction, or plus one laterally.
% first, let's normalize this thing, just in case…
map = map/max(map(:));
if (xvel > 0) %OK, lets try to the right. Remember, i is row (y), j is column (x)
   if (map(i-1,j+1) > 0.9)  %look up diagonally right first; up is NEGATIVE, remember?
      yvel = yvel-1;  %this wouldn't work if we hadn't normalized the map
      xvel = xvel+1;
   elseif (map(i+1,j+1) > 0.9) %look down diagonally right
      yvel = yvel+1;
      xvel = xvel+1;
   elseif (map(i-1,j) > 0.9)  %look up;
      yvel = yvel-1;
   elseif (map(i,j+1) > 0.9) %look to the right
      xvel = xvel+1;
   elseif (map(i+1,j) > 0.9)  %look down
      yvel = yvel+1;
   end
elseif (xvel <= 0) %Try to the left. Remember, i is row (y), j is column (x)
   if (map(i-1,j-1) > 0.9)  %look up diagonally left
      yvel = yvel-1;
      xvel = xvel-1;
   elseif (map(i+1,j-1) > 0.9) %look down diagonally left
      yvel = yvel+1;
      xvel = xvel-1;
   elseif (map(i-1,j) > 0.9)  %look up;
      yvel = yvel-1;  %this wouldn't work if we hadn't normalized the map
   elseif (map(i,j-1) > 0.9) %look to the left
      xvel = xvel-1;
   elseif (map(i+1,j) > 0.9)  %look down
      yvel = yvel+1;
   end
end
% this usually leaves the peak unchanged
```

B.6.3 Scoring Routine

```
function [score] = numerical_score(refboxes, trialboxes)
% This compares the ground truth boxes (refboxes) with the machine
% generated boxes (trialboxes) and computes deltas for later display.
% The stats come out as the vector score.
%
% Inputs
%       refboxes - Box vectors from the manual scoring program.  Format is
%                       upper lhc x, upper lhc y, width, heigth (standard format)
%       trialboxes - Box vector from method. Format is the same.
% Output
%       score – the score: miss distance, size ratio
%
% ©   V. M. Stone     2018
%
% Compare the length of the box arrays.  If they aren't equal, there's a
% problem.
[~,reflen] = size(refboxes);
[~,triallen] = size(trialboxes);
if (reflen ~= triallen)
   display('Box arrays are not the same length!');
   return;
end
% Make an empty score array
score = zeros(2, reflen);
% Compute the normalization factors.  Area will be normalized to the area
% of the reference box, which will be computed here once (it's always the
% same size) and the "miss distance" normalized to 1/2 the diagonal of the
% reference box, again computer here only once.
% First, the area of the reference box
refarea = refboxes(3, 1)*refboxes(4, 1);
reflength = norm([refboxes(3,1),refboxes(4,1)])/2;
% Loop through the boxes and compile the statistics.
for i = 1:reflen   %Since they're the same length, either one will do
   % Compute trial box stats
   trialarea = trialboxes(3, i)*trialboxes(4, i);
   % Normalize the trial area for direct comparison (of course, normalized
   % refarea is 1).  If the trial area is > 1, it's bigger.
   % Compute normalized trial area
   normarea = trialarea/refarea;
   % Compute normalized distance metric. First, we get the centers of the boxes.
   refx = refboxes(1, i)+(refboxes(3, i)/2);
   refy = refboxes(2, i)+(refboxes(4, i)/2);
```

```
    trix = trialboxes(1, i)+(trialboxes(3, i)/2);
    triy = trialboxes(2, i)+(trialboxes(4, i)/2);
    % Now, we compute the distance and normalize it to 1/2 the diagonal of the ref
    % box.
%    distance = abs(sqrt((trix-refx)^2+(triy-refy)^2))/(refboxes(3,i)/2);
    normdistance = norm([(trix-refx),(triy-refy)])/reflength;
    %add to the vector
    score(1, i) = normarea;
    score(2, i) = normdistance;
end
% We should be done.
```

References

[1] J. K. Aggarwal and N. Nadhakumar, “On the Computation of Motion from Sequences of Images – A Review,” *Proc. Of the IEEE*, vol. 76, no. 8, pp. 917-935, August 1988.

[2] J. Konrad, “Motion Detection and Estimation,” in *The Essential Guide to Video Processing*, 2nd ed., A.C. Bovik, Ed., New York: Elsevier, 2009, pp. 31-68.

[3] Y. Wang, J. Ostermann, and Y. Zhang, *Video Processing and Communications*. New Jersey: Prentice-Hall, 2002, ch. 6, 7.

[4] A. Kojima, N. Sakurai, and J. Kishigami, “Motion Detection Using 3D-FFT Spectrum,” in *Proc. 1993 IEEE ICASSP*, vol. 5, pp. 213-216.

[5] A. Briassouli and N. Ahuja, “Integration of Frequency and Space for Multiple Motion Estimation and Shape-Independent Object Segmentation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 5, pp. 657-669, May 2008.

[6] D. Alexiadis and G. Sergiadis, “Estimation of Motions in Color Image Sequences Using Hypercomplex Fourier Transforms,” *IEEE Transactions on Image Processing*, vol. 18, no. 1, pp. 168-178, January 2009.

[7] B. Horn and B. Schunck, "Determining Optical Flow," Artificial Intelligence, vol. 17, pp. 185-204, 1981.

[8] T. Bouwmans and E. H. Zahzah, “Robust PCA via Principal Component Pursuit: A review for comparative evaluation in video surveillance,” *Computer Vision and Image Understanding*, vol. 122, pp 22-34, 2014.

[9] P. Rodriguez, B. Wolhberg, "Incremental Principal Component Pursuit for Video Background Modeling", submitted *IEEE Signal Processing Letters*, 2014.

[10] P. Rodriguez, B. Wolhberg, "Fast Principal Component Pursuit via Alternating Minimization", *Proceedings of the 2013 IEEE International Conference on Image Processing* (ICIP'13), Melbourne, Australia, pp. 69-73, Sep. 2013.

[11] A. Kojima, N. Sakurai, and J. Kishigami, “Motion Detection Using 3D-FFT Spectrum,” in *Proc. 1993 IEEE ICASSP*, vol. 5, pp. 213-216.

[12] A. Briassouli and N. Ahuja, “Integration of Frequency and Space for Multiple Motion Estimation and Shape-Independent Object Segmentation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 5, pp. 657-669, May 2008.

[13] Wang, Y., J. Ostermann, and Y Zhang. *Video Processing and Communications*. Upper Saddle River, New Jersey; Prentice Hall, 2002, ch. 3.

[14] Dudgeon, D. E., and R. M. Mersereau. *Multidimensional Digital Signal Processing.* Englewood Cliffs, New Jersey; Prentice Hall, 1984.

[15] Leo I. Bluestein, "A linear filtering approach to the computation of the discrete Fourier transform," *Northeast Electronics Research and Engineering Meeting Record*, vol. 10, 218-219 (1968).

[16] Lawrence R. Rabiner, Ronald W. Schafer, and Charles M. Rader, "The chirp z transform algorithm and its application," *Bell Syst. Tech. J.* **48**, 1249-1292 (1969). Also published in: Rabiner, Shafer, and Rader, "The chirp z-transform algorithm," *IEEE Transactions on Audio Electroacoustics,* vol. 17, no. 2, 86–92 (1969). K.

[17] K. Seshadrinathan, R. Soundararajan, A. C. Bovik and L. K. Cormack, "Study of Subjective and Objective Quality Assessment of Video", *IEEE Transactions on Image Processing* , vol.19, no.6, pp.1427-1441, June 2010.

[18] K. Seshadrinathan, R. Soundararajan, A. C. Bovik and L. K. Cormack, "A Subjective Study to Evaluate Video Quality Assessment Algorithms", *SPIE Proceedings Human Vision and Electronic Imaging*, Jan. 2010.

[19] R. Collins, X. Zhou, and S. K. Teh, "An Open Source Tracking Testbed and Evaluation Web Site," *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS 2005)*, January, 2005.

Journal papers (published, accepted, or to appear)

[20] Pattichis, C.S., "Radiogenomics for Precision Medicine With A Big Data Analytics Perspective," *IEEE Journal of Biomedical and Health Informatics,* accepted.

[21] Carranza, C., Llamocca, D., and Pattichis, M.S., "Fast 2D Convolutions and Cross-Correlations Using Scalable Architectures," *IEEE Transactions on Image Processing*, vol. 26, no. 5, pp. 2230-2245, May 2017.

[22] Carranza, C., Llamocca, D., and Pattichis, M.S., "Fast and Scalable Computation of the Forward and Inverse Discrete Periodic Radon Transform," *IEEE Transactions on Image Processing*, vol. 25, no. 1, pp. 119-133, Jan. 2016.

[23] Llamocca, D. and Pattichis, M.S., "Dynamic Energy, Performance, and Accuracy Optimization and Management for Separable 2-D Filtering for Digital Video Processing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 4, article 30, 30 pages, Jan. 2015.

[24] Llamocca, D., Pattichis, M., "A Self-Reconfigurable Platform for the Implementation of 2D Filterbanks with Real and Complex-valued Inputs, Outputs, and Filter Coefficients", *VLSI Design*, vol. 2014 (2014), Article ID 651943, 24 pages, http://www.hindawi.com/journals/vlsi/2014/651943/.

[25] Christodoulou, C.I., Kaplanis, P.A., Murray, V., Pattichis, M.S., Pattichis, C.S., and Kyriakides, T., "Multi-Scale AM-FM Analysis for the Classification of Surface Electromyographic Signals," *Journal of Biomedical Signal Processing and Control*, vol. 7, no. 3, pp. 265-269, 2012.

[26] Murray, V., Pattichis, M.S., Barriga, E.S., and Soliz, P., "Recent Multiscale AM-FM Methods in Emerging Applications in Medical Imaging," *EURASIP Journal on Advances in Signal Processing (Springer)*, vol. 2012, no. 1, 23 pages, 2012.

[27] Ramachandran, J., Pattichis, M.S., Scuderi, L.A., and Baba, J.S., "Tree Image Growth Analysis Using Instantaneous Phase Modulation," *EURASIP Journal on Advances in Signal Processing*, Special Issue: Recent Advances in Theory and Methods for Non-stationary Signal Analysis, vol. 2011, Article ID 518602, 22 pages, 2011.

[28] Loizou, C.P., Murray, V., Pattichis, M.S., Pantziaris, M., and Pattichis, C.S., "Multiscale Amplitude-Modulation Frequency-Modulation (AM-FM) Analysis of Ultrasound Images of the Intima and Media Layers of the Carotid Artery," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 2, pp. 178-188, 2011, PMID 20889436.

[29] Loizou, C.P., Murray, V., Pattichis, M.S., Seimenis, I., Pantziaris, M., and Pattichis, C.S., "Multiscale Amplitude-Modulation Frequency-Modulation (AM-FM) Texture Analysis of Multiple Sclerosis in Brain MRI Images," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 1, pp. 119-129, Jan. 2011, PMID 21062681.

[30] Murray, V., Rodriguez, P. and Pattichis, M.S., "Multi-scale AM-FM Demodulation and Reconstruction Methods with Improved Accuracy," *IEEE Transactions on Image Processing*, vol 19, no. 2, pp. 1138-1152, May 2010, PMID: 20071260.

[31] Agurto, C., Murray, V., Barriga, E., Murillo, S., Pattichis, M.S., Davis, H., Russell, S.R., Abramoff, M.D., and Soliz, P., "Multiscale AM-FM Methods for Diabetic Retinopathy Lesion Detection," *IEEE Transactions on Medical Imaging*, vol. 29, no. 2, pp. 502-512, February 2010, PMID: 20129850.

[32] Pattichis, M.S. and Bovik, A.C., "Analyzing image Structure by multidimensional frequency Modulation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 753-766, May 2007, PMID: 17356197.

[33] Pattichis, M.S., Panayi, G., Bovik, A.C., and Shun-Pin, H., "Fingerprint Classification Using an AM-FM Model," *IEEE Transactions on Image Processing*, vol. 10, no. 6, pp. 951-954, June 2001.

[34] Pattichis, M.S., Bovik, A.C., Havlicek, J.W., and Sidiropoulos, N.D., "Multidimensional Orthogonal FM Transforms," *IEEE Transactions on Image Processing*, vol. 10, no. 3, pp. 448-464, March 2001, PMID: 18249634.

[35] Pattichis, M.S., Pattichis, C.S., Avraam, M., Bovik, A.C., and Kyriakou, K. "AM-FM Texture Segmentation in Electron Microscopic Muscle Imaging," *IEEE Transactions on Medical Imaging,* vol. 19, no. 12, pp. 1253-1258, December 2000, PMID: 112

Conference papers

[36] Jacoby, A., Pattichis, M., Celedón-Pattichis, S., and LópezLeiva, C., "Context-sensitive Human Activity Classification in Collaborative Learning Environments," *IEEE Southwest Symposium on Image Analysis and Interpretation*, pp. 141-144, 2018.

[37] Shi, W., Pattichis, M., Celedón-Pattichis, S., and LópezLeiva, C., "Robust Head Detection in Collaborative Learning Environments using AM-FM Representations," *IEEE Southwest Symposium on Image Analysis and Interpretation*, pp. 65-68, 2018.

[38] Stubbs, S., Pattichis, M., and Birch, G., "Interactive Image and Video Classification using Compressively Sensed Images," to appear, *2017 Asilomar Conference on Signals, Systems, and Computers*.

[39] Panayides, A., Pattichis, C., and Pattichis, M., "The Potential of Big Data Medical Video Analytics in Healthcare," to appear, 2017 IEEE International Conference on Biomedical and Health Informatics, 2017.

[40] Pattichis, M. S., Celedón-Pattichis, S., & LópezLeiva, C. A., "Teaching image and video processing using middle school mathematics and the Raspberry Pi," in Special Session on Advances in Signal Processing Education, IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), New Orleans, Louisiana, pp. 6349-6353, 2017.

[41] Eilar, C., Jatla, V., Pattichis, M. S., Celedón-Pattichis, S., & LópezLeiva, C. A., "Distributed Video Analysis for the Advancing Out of School Learning in Mathematics and Engineering Project," *2016 Asilomar Conference on Signals, Systems, and Computers*, pp. 571-575, 2016.

[42] Sarkar, Indranil, and Adly T. Fam. "The interlaced chirp Z transform." *Signal Processing* 86.9 (2006): 2221-2232.