Tracking of Various Targets in the Infrared and Issues Encountered

by

Kyle T. Ausfeld

B.S. University of Rochester, 2010

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Chester F. Carlson Center for Imaging Science
College of Science
Rochester Institute of Technology

August 20, 2012

Signature of the Author _____

Accepted by _____
Dr. John Kerekes, M.S. Degree Program Date

CHESTER F. CARLSON CENTER FOR IMAGING SCIENCE
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Kyle T. Ausfeld
has been examined and approved by the
thesis committee as satisfactory for the
thesis required for the
M.S. degree in Imaging Science

Dr. Zoran Ninkov, Thesis Advisor

Dr. Carl Salvaggio

Dr. J. Daniel Newman

Date

Tracking of Various Targets in the Infrared and Issues Encountered

by

Kyle T. Ausfeld

Submitted to the
Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Master of Science Degree
at the Rochester Institute of Technology

Abstract

Computer aided object tracking is a subject of increasing interest. The applications for these tracking algorithms are widespread; from homeland security and surveillance to the study of animal behavior. In past years, the visible part of the electromagnetic spectrum has been the dominant regime in which algorithms have been developed. This is in part due to a greater amount of available data while many of these algorithms were being developed. Algorithms such as the Mean Shift algorithm have become a standard in which other tracking algorithms are tested against. However, infrared video data presents some challenges that are not as pronounced or existent in visible video data. Many infrared detectors are much more easily saturated than visible detectors, which can cause a loss of both spatial and temporal information about an object of interest. Although there are extra challenges associated with infrared data, the advantages make the development of tracking algorithms for the infrared an important task. Persistence surveillance of targets of interest may be performed with the use of one infrared imaging system due to day and night imaging capabilities. This work demonstrates the utility of a polynomial fitting adaptive Kaman filter with the measurement made by the Mean Shift algorithm for tracking non-linear object motion in the infrared. Additionally, the Pearson product-moment correlation coefficient is shown to be superior to the Bhattacharyya correlation coefficient when working with low spatial resolution and noisy images.

Acknowledgements

I would like to thank the New York State Foundation for Science, Technology, and Innovation (NYSTAR) through the Center for Emerging and Innovative Sciences (CEIS), ITT Exelis Geospatial Systems and NASA (grant NAS5-2105) for supporting this work. Extra thanks goes out to Zoran Ninkov, Dan Newman, Carl Salvaggio, Paul Lee, David Rhodes, Judy Pipher, Kenny Fourspring, and Ross Robinson for thoughtful discussions, direction giving, and help during data collects, as well as help with some post-processing of the data.

I would like to dedicate this work to my family for helping and supporting me though school, as well as to my girlfriend, for her encouragement and patience.

Contents

List of Figures

Chapter 1

Introduction

The tracking of moving objects within their field of view is something that humans and animals have naturally done in order to survive. It allows us to locate food and detect danger. In recent times, the development of technology have allowed us to automate these processes, allowing a more hands-off approach to detecting objects of interest or danger. Imaging systems have become increasingly more advanced, allowing better spatial, temporal, and spectral resolution. Similarly, computer hardware and software have advanced to the point where we are able to make use of them to detect and track objects for us in increasingly less time. This chapter is broken up into a background on general object tracking (Section 1.1) and an introduction as to why the infrared spectral region is useful for object tracking (Section 1.2).

1.1 Object Tracking

The most general tracking objective is to specify the location of an object of interest. In general, object tracking refers to the use of an image sequence to detect and follow where an object moves within a scene. Further analysis of this information gives insight into how the object is moving and can be used for anomaly detection [1], gait detection [2], animal behavior [3], traffic flow [4], or the detection of movements within a restricted area. The ability to detect and quantify these changes without human input is the desired outcome, saving both time and resources.

There are a number of different approaches to the development of tracking algorithms. A thorough discussion of the categorization of tracking objects may be found in [5]. Two major approaches to the development of tracking algorithms are discussed below; those that focus on target representation (Section 1.1.1) and those that use filtering and statistics for tracking (Section 1.1.2).

1.1.1 Target Representation Approach

Algorithms that fall under the representation category create a map containing information on where the object is. The way the object may be represented varies greatly, from a probability map based on distance and "color" information (Mean Shift algorithm), to edge representation tracking [5]. The Mean Shift algorithm will be discussed in greater detail in Section 2.2.

Edge representation tracking is performed using an edge detection algorithm (e.g., Sobel or Canny), followed by the selection of a shape of interest. Using the edge information, the next frame to be used simply applies a search algorithm for an object who's edges have the best correlation with the model from the first frame. There are drawbacks to this technique. Some applications have objects that do not have "hard" enough edges to be detected accurately, while other applications may have too many edges to make this an efficient method.

A simple representation that may be used in infrared applications is a simple "hot spot" tracking algorithm. This type of representation only works when the object of interest is imaged in the thermal infrared, and is significantly hotter than its surroundings. The object appears to be a group of bright pixels, and is represented by these bright pixels. Tracking is performed by detecting this bright set of pixels in subsequent frames. Although this is a crude method, it can be performed quickly and accurately if a scenario is sufficiently simple.

1.1.2 Filtering and Statistics Approach

Representing objects with filters and statistics has been implemented in various fields, from tracking to mean estimation of noisy data and noise identification [6, 7]. These types of algorithms use a recursive filter to predict motion of a particle (statistically, e.g., a Kalman filter, discussed in more detail in Section 2.3) or use a filter associate various data points with an object, giving a way to search for the object.

The data association method works by forming a statistical model of an object in the initiating frame. The following frames associate the new data with the model created either in the initiating frame, or the last frame in which the model was updated. If only a single object is of interest, a measurement is made to find the data defined by the model in the new frame. When multiple objects are of interest, a simple approach looks at previous locations to perform a nearest neighbor search [5]. More robust approaches exists, but discussion of such methods will be deferred to [5].

1.2 The Infrared and Object Tracking

The infrared part of the electromagnetic spectrum (0.74μm to 300μm) is rapidly gaining interest for object tracking due to the increase in availability at decreasing costs. There

have been many developments in both the format and size of infrared detector arrays, providing higher spatial resolution. Using a new part of the spectrum allows tracking algorithms to work with information not available in the visible part of the spectrum (380 nm to 740 nm).

Infrared detectors are able to perform surveillance around-the-clock. This is due to the self-emission of objects in the infrared as described by black body radiation (mid-wave to long-wave infrared), or the reflection of light at non-visible wavelengths (near-infrared to mid-wave infrared). The emissivity of an object describes how close to a black body the object radiates at. Therefore, objects of similar temperatures with different emissivities or vice-versa become clear to differentiate. This is especially useful for detecting and tracking camouflaged objects. In the visible part of the spectrum, the camouflage makes an object of interest blend into the background. In the infrared, the warmth of the disguised object or the change in emissivity from object to background would give a different signature, making the task of object detection and object tracking possible. Another situation could be that of two "identical" objects undergoing an occlusion during tracking. After the objects of interest appear from behind the occluding object, the distinction of these "identical" objects in the visible spectrum can be rather difficult. However, if one object was warmer than the other (due to running longer or sat in the sun while the other was in shade), these objects would appear different both before and after the occlusion when viewed at in the infrared. This extra information obtained in the infrared allow tracking algorithms to succeed where they would otherwise struggle.

There are some disadvantages to using only the infrared part of the spectrum for object tracking. Most infrared detector arrays are only capable of collecting spatial data in only one spectral band. On the contrary, many detectors in the visible part of the spectrum are able to collect spectral data as well as spatial data via color filter arrays. Multi-band infrared detector arrays have been built and are in use[8], however, there is a lack of sufficient data available to the public to develop tracking algorithms. Another solution is the use of both a multi-band visible camera system combined with an infrared camera system, giving even more data and flexibility [9].

Another difficulty with viewing objects in the infrared is due to the strong dependence between orientation and appearance. Part of this dependence comes from object profile change with orientation angle, however this effects both the visible and the infrared parts of the spectrum. In the visible spectrum, the color of an object is being detected. The color of objects (such as cars or airplanes) tend to be uniform, so only the object silhouette changes with view angle, but the color component stays the same. In the emissive part of the infrared (mid-wave to long-wave), the light that is being detected can be viewed as an object's temperature. Along with the silhouette changes with view angle, an objects temperature is also likely change due to engine heating or some other heat source. This is shown in figure 1.1. This set of images from the Sensiac ATR Algorithm Development Image Database of a BMP2 armored personnel carrier in the mid-wave infrared at night

[10]. This figure demonstrates how heat from an engine's exhaust may become clear in the infrared from one view angle, but undetectable from another. Also note that the object profile from both view angles is very similar.

Additionally, changes in view angle in the infrared can reveal "hot spots" such as the one shown in Figure 1.1. These spots are from a rapid increase in photon flux, which can be caused by a heat source (mid-wave to long-wave infrared) or by a solar glint (near to mid-wave infrared). These rapid increases in photon flux can cause detectors to saturate if high enough. Due to "bleeding" in detectors when they saturate, information about the saturating object is washed out and can even effect near-by objects. An example of the saturation due to solar glint is shown in Figure 1.2, taken in the mid-wave infrared (visible image shown in Figure 1.3). This shows a 300S aerobatic airplane at the 2011 ESL Rochester International Air Show perform an aerobatic roll while diving. The change in orientation of the wing causes a glint to be directed at the detector, causing a saturation. The only information obtainable during a saturation is the general position of the object, but even that can be difficult to determine with a more severe saturation.

An example of infrared detector saturation due to engine heat is shown in Figure 1.4, showing the US Navy Aerobatic Team's Blue Angels during the 2011 ESL Rochester International Air Show. The image demonstrates a saturation due to engine exhaust as the Blue Angels perform an aerobatic stunt. The other jets to the bottom right show how large a saturation can become, and how close objects can blend together during a saturation. Some detectors exhibit an effect during saturation which appears as a "ghost" near the saturating object. This shows up in Figure 1.4 surrounding the saturating Blue Angel. Both Figure 1.2 and 1.4 were taken with a nitrogen cooled Kodak KIR-310 PtSi 640x480 pixel array with a germanium lens. The focal length is 200 mm with f/2, and a 3.2-4.1 μm filter. The data was collected from the Rochester Institute of Technology campus, about 4 km from the Rochester International Airport (where the air show is held) at a frame rate of 24 frames per second.

There are difficulties with developing object tracking algorithms in the infrared part of the spectrum. However, the advantages infrared data provides over data collected in the visible spectrum makes the development of infrared tracking algorithms an important task. Most current infrared tracking algorithms work by taking algorithms developed for use in the visible spectrum, and modify them to work in the infrared. This is generally very easy to do; usually just a matter of using a greyscale color system instead of RGB or HSV. This is the approach used in this work, and the algorithms used with be discussed in Chapter 2. After the algorithms are introduced and discussed, some applications of both tracking and infrared tracking are introduced and shown in Chapter **??**. The last chapter (3) will discuss the results of the newly implemented algorithm (introduced in Section 2.5) along with results and implications from the discussion on correlation coefficient selection (Section 2.4).

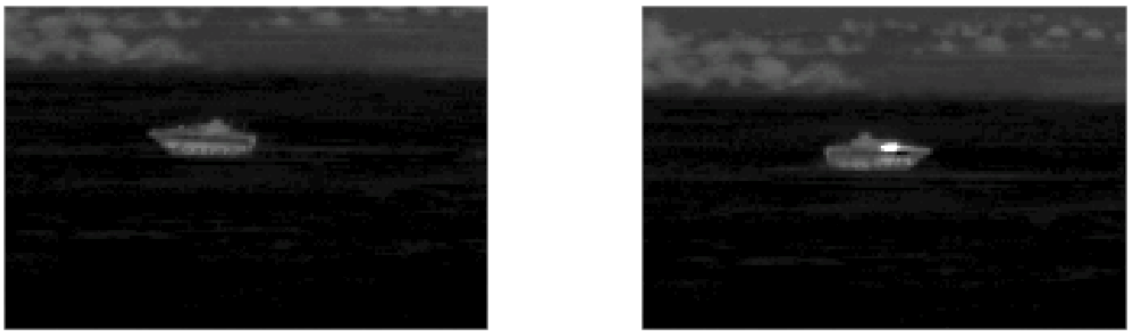


Figure 1.1: A set of infrared images of a BMP2 armored personnel carrier at night. Note the clearly visible hot spot, due to engine heat, only visible on one side of the vehicle.

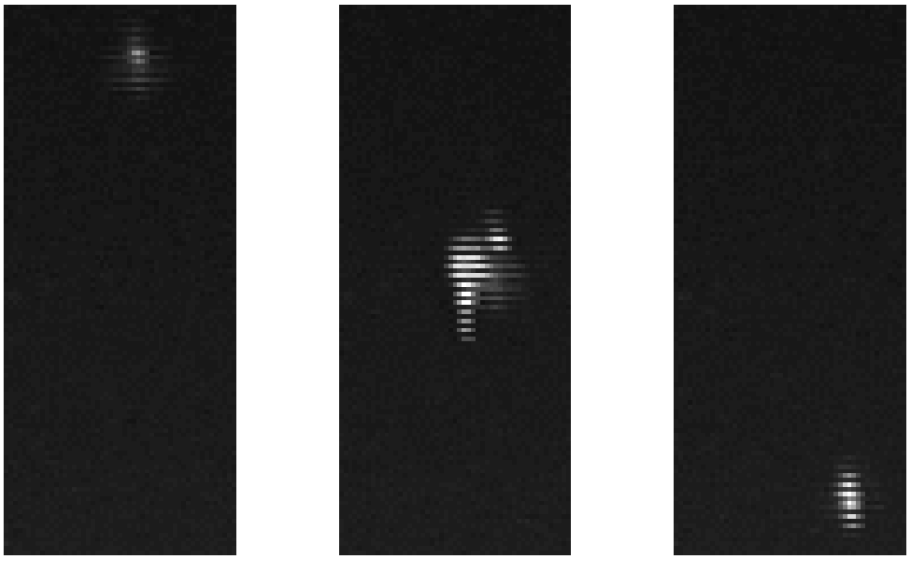


Figure 1.2: Extra 300S aerobatic airplane at the 2011 ESL Rochester International Air Show, shown in the infrared. From left to right, frame numbers 35, 45, and 55, with a frame rate of 24 frames per second. Note the large change in object appearance in under one second, including a saturation due to glint from the airplane wing under a slight roll manoeuver.

Figure 1.3: Extra 300S aerobatic airplane flown at the 2011 ESL Rochester International Air Show, shown in the visible. Taken from the ESL Rochester International Air Show website[11].

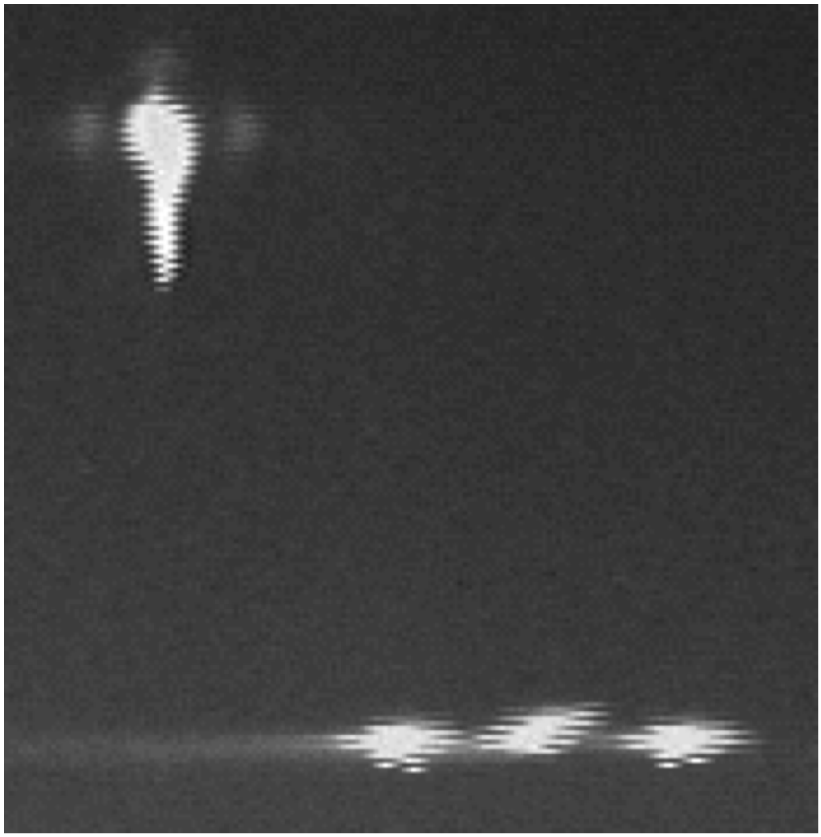


Figure 1.4: Infrared sequence of the US Navy Aerobatic Team's Blue Angels during the 2011 ESL Rochester International Air Show. Note how many pixels a saturating jet takes up compared to the other jets. Also note the "ghosting" caused by the detector array during saturation.

Bibliography

[1] Fan Jiang; Ying Wu; Katsaggelos, A.K.; "Detecting contextual anomalies of crowd motion in surveillance video," *16th IEEE International conference on Image Processing (ICIP)*, pp.1117-1120, 7-10 Nov. 2009.

[2] Lee, L.; Grimson, W.E.L.; "Gait analysis for recognition and classification," *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, pp.148-155, 20-21 May 2002.

[3] Suryan, R. M. and Harvey, J. T. (1998); "Tracking harbor seals (phoca vitulina richardsi) to determine dive behavior, foraging activity, and haul-out site use." *Marine Mammal Science*, 14: 361372. doi: 10.1111/j.1748-7692.1998.tb00728.x.

[4] Young-Kee Jung; Yo-Sung Ho; "Traffic parameter extraction using video-based vehicle tracking," *1999 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems, Proceedings*, vol., no., pp.764-769, 1999.

[5] Yilmaz, A., Javed, O., and Shah, M.; "Object tracking: A survey." *ACM Comput. Surv.* 38, 4, Article 13 (Dec. 2006), 45 pages. DOI = 10.1145/1177352.1177355 http://doi.acm.org/10.1145/1177352.1177355.

[6] Welch, Greg; Bishop, Gary; "An Introduction to the Kalman Filter," *University of North Carolina at Chapel Hill*, pp. 1-16, 2006.

[7] Oussalah, M.; De Schutter, J., "Adaptive Kalman Filter for Noise Identification," *Proceedings of the International Conference on Noise and Vibration Engineering*, pp. 1225-1232, 2000.

[8] Parish, G.; Musca, C.A.; Siliquini, J.F.; Antoszewki, J.; Dell, J.M.; Nener, B.D.; Faraone, L.; Gouws, G.J.; "A monolithic dual-band HgCdTe infrared detector structure," *Electron Device Letters*, IEEE , Vol. 18, no. 7, pp. 352-354, Jul 1997.

[9] Goubet, Emmanuel; Katz, Joseph; Porikli, Fatih, "Pedestrian Tracking Using Thermal Infrared Imaging," *Infrared Technology and Applications XXXII*, pp. 62062C-1-12, 2006.

[10] ATR Algorithm Development Image Database, *Military Sensing Information Analysis Center*, 25 July 2008.

[11] http://www.rochesterairshow.com/performers.php, 25 Jan. 2012.

Chapter 2

Algorithms

This chapter will discuss the idea behind some algorithms used in tracking, as well as how they are implemented. Before object tracking may commence, an object must first be detected. There are many object detection algorithms, some very simple, other complex. Section 2.1 will introduce just a couple simple methods. The tracking in the remainder of this work will start with a manual detection in order to test just the tracking algorithms themselves. The tracking algorithms used include the Mean Shift algorithm (MS, Section 2.2) and Kalman filters (KF, Section 2.3). Another important consideration when using tracking algorithms is how the correlation coefficient choice can effect algorithm performance (Section 2.4). The proposed method is a combination of a KF with a polynomial position fitting (PFAKF), discussed in Section 2.5.

2.1 Brief Introduction to Object Detection

Object detection is in itself a subject of study. There are many different algorithms to perform this task, running the gamut from Bayesian network detectors [1], to learning algorithms using single or multiple classes [2, 3]. For this work, a quick introduction into a couple simple methods will be sufficient. The methods discussed below will all work on data with a constant background (e.g., video overlooking a parking lot) and with moving targets (e.g., cars moving around said parking lot) to varying degree; based on image quality. All detection figures will be differed to Appendix A.

2.1.1 Frame Differencing

Frame differencing is the most basic way to detect a moving object in a stationary scene. Early uses of frame differencing were in the 1970's with the advent of television [4, 5]. There are a few ways in which frame differencing can be used, but all methods employ the subtraction of one frame from another.

Single Frame Difference

The standard implementation is just a single frame difference and is very quick to implement. The image remaining generally consists of "blobs" representing an object moving through the scene. Mathematically, it is defined as

$$D_i = F_i - F_{i-1}, \tag{2.1}$$

where D is the differenced image, F is the set of raw frames, and $i, i-1$ represent the current frame and the previous frame, respectively. This can also be implemented using a known background image with no objects in it to subtract off the background. The only difference would be to substitute $F_{i-1} \rightarrow B$ in equation (2.1) where B is the background image.

One of the biggest difficulties with this method arises when working with noisy images, or if the background is not completely stable. This occurs when there is lots of foliage in the scene and the wind is substantial. An example of single frame differencing is shown in Figure A.2. The image was taken in the infrared and is taken from the Sensiac database [6]. It is relatively noise free, but notice how the entire image is speckled with noise, potentially causing confusion.

Double Difference

A more robust technique of frame differencing is called the double difference, and first appeared in 2000, by Collins et al. [7]. The idea behind this technique is to limit the effects of noise on frame differencing. The technique involves two steps, the first of which is creating a pair of differenced frames,

$$D_i = F_i - F_{i-1} \text{ and } D_{i-1} = F_{i-1} - F_{i-2}. \tag{2.2}$$

The next step is a comparison step, pixel-by-pixel, with a threshold in order mitigate effects of noise in the image sequence. This looks like

$$DD(m,n) = \begin{cases} 1 & \text{if } D_i(m,n) \ \& \ D_{i-1}(m,n) > \text{Threshold} \\ 0 & \text{else} \end{cases}.$$

DD is then a binary image of moving objects. The threshold is chosen such that the noise in the image sequence is attenuated. This passes only the moving objects, making target detection more robust.

An example of the improvement provided by the double difference can be seen in Figure A.3. This is the same sequence used above in Figure A.2. Notice how the noise in the sky that could be confused as an object has been dramatically reduced, leaving just the object itself.

Statistical Difference

The previous methods may be sufficient for high quality data with a good signal-to-noise ratio, they fail drastically when noisy data is presented. Another frame difference method is to use a statistical quantity to subtract off the current frame. This technique can prove to be more robust on a wider range of image quality, but with a loss of computational speed. To properly build up a good set of data in which to calculate statistics requires a decent number of frames to initiate the detection. This is not an issue if using video with close to 30 frames per second if there are minutes of this data to analyze. If this is not the case, other methods would be preferable.

The statistical difference frames are all created in the same way, but can use different statistical properties to calculate them. The statistics are calculated at each pixel in the previous N frames at each pixel. This frame is then subtracted from the current frame, similar to equation (2.1). In very noisy data, a standard approach is to use the median value at each pixel. However, using the mode of the pixels has also proven to be useful for detection. Figure A.5 shows the mode difference image of Figure A.4. For comparison, Figure A.6 shows the median difference image, while Figure A.7 shows the mean. Notice how the mean is the worse of the statistical difference methods,, but still could be used. However, it does seem to show "trails" of where the moving objects were, giving potential false positives.

For comparison to the earlier techniques, Figures A.8 and A.9 show the single frame difference and double difference methods on the same image. Note that significantly more noise gets through, making the differenced images completely useless for object detection. The large amount of noise in this data is likely due to the imager not being cooled. This data was taken using a microbolometer, at about 10 μm from the roof of RIT overlooking a parking lot. The "1" at the bottom of Figure A.4 is a watermark that could not be removed from the camera, and is the filter number currently being used.

2.2 Mean Shift

The Mean Shift algorithm (MS) was first used as a gradient decent technique used in pattern recognition [8]. It was more recently used for feature space image segmentation [9], followed by object tracking [10]. It is also known as kernel based object tracking, due to the use of two kernels that are applied to the target before the centroiding part of the algorithm.

This algorithm is generally used in tracking on visible data, and can be implemented on either color or monochromatic image sequences. Although this algorithm only works using a single "color" band, color images are generally converted to HVS (hue, value, saturation) space, then the analysis is performed on the hue component, although this is not necessary. Due to this structure, it is very simple to implement on infrared data, using

the intensity as one would with monochromatic images. However, there are some minor differences one should be aware of before simply applying this algorithm to any image sequence.

When using the infrared, some difficulties arise in the fact that the target and background are not always so distinct. This can occur if taking video in the thermal infrared on a hot day of vehicles. By contrast similar video taken during the winter months would prove simple to track on, as the targets are very distinct. This difficulty arises because MS can be distracted by near-by objects of similar appearance to the object of interest. Although the kernel based on the "color" of an object can limit this, as well as the distance kernel, some data is simply more difficult for MS to track.

The algorithm can be viewed as a procedure performed at every frame, and calculates the center of an object. The object the centroid is performed on is not just a sub-image containing the object of interest, but a probability density function of that object. Here is the general form of the algorithm, performed at each frame.

1. Given an initial (or previous) location and object size, create sub-image containing the original object's location, larger than the object's size.

2. Create a kernel based on object size to weight the center pixels more heavily.

3. Create a kernel based on the pixel intensity range for the object in question.

4. Apply both kernels to the sub-image, creating a probability density function of the likelihood a pixel belongs to the object of interest.

5. Sum all pixels or perform a moment calculation on the distribution, finding the center location.

6. Repeat this, using the location found last until convergence is met, or a maximum iteration count is exceeded.

MS uses the initiated sub-image and location to create a target model which is then used to extract the pixel intensity range during step 3. This target model is denoted by q, while the target candidate is denoted p. For more details on how I implemented the MS in MATLAB, see Appendix B.1.

The kernel used for spatial "windowing" will be denoted k_d while the kernel for pixel intensity will be denoted k_h. To calculate k_d, we must first choose a kernel function which we shall use to weight the distance from the center of the sub-image. The kernel must be an isotropic, convex, and monotonically decreasing function [10]. This makes sure the likelihood of belonging to the object of interest is forced to the center of the sub-image. There are two commonly used functions, the Epanechnikov and Gaussian kernel profiles. This work makes use of the Epanechnikov for reasons explained further down, and is

recommended for use. For this work, the spatial kernel is defined as:

$$k_d(i,j) = \frac{2}{\pi}\left(1 - \sqrt{\frac{(i - i_{center})^2 + (j - j_{center})^2}{d^2}}\right). \tag{2.3}$$

i and j represent the pixel locations within the sub-image, and i_{center} and j_{center} define the center of this window. d is an adjustable parameter used to adjust the speed in which the kernel drops off, and depends on target size.

The calculation of k_h may be performed in a couple different ways, depending on application. If the object of interest is pretty uniform in intensity, then the kernel may use a simple triangle function centered at the mean intensity of the object.

$$k_h(u) = 1 - \frac{\|u - u_{peak}\|}{\mid h \mid} \tag{2.4}$$

This is was the equation used for this work, with h representing the number of histogram bins on either side of the central bin, u_{peak}. Outside of this region, k_h is defined to be zero. The central peak may be chosen either by hand, or may be selected by a set procedure. Selection by hand may be more reliable if the histogram is not easily segmented into target and background, making a simple selection procedure inaccurate. This is discussed in further detail in Appendix B.1.2.

The after the two kernels are calculated, we are ready to calculated both p and q. q is calculated using the model, which is the first frame of tracking, I_1. It may be calculated only once, as it should not change frame-to-frame*. This gives:

$$q(i,j) = \sum_{u}^{m \text{ bins}} I_1(i,j,u') \cdot k_d(i,j) \cdot k_h(u) \cdot \delta\left(u, u'\right). \tag{2.5}$$

u' represents the bin that pixel (i,j) belongs to, and $\delta\left(u,u'\right)$ is the Kronecker delta function, defined as

$$\delta\left(u,u'\right) = \begin{cases} 1 & \text{iff } u = u' \\ 0 & \text{iff } u \neq u' \end{cases}. \tag{2.6}$$

The calculation of p is similar to that of q, with the change of $I_1 \rightarrow I_n$, where n is the current frame number.

$$p(i,j) = \sum_{u}^{m \text{ bins}} I_n(i,j,u') \cdot k_d(i,j) \cdot k_h(u) \cdot \delta\left(u, u'\right) \tag{2.7}$$

*As a word of warning, if the object of interest starts to move off the side of the frame, the size of p may become smaller than q. In this case, tracking would either be over for the object, or q could be recalculated with a size equal to p, allowing for a couple extra frames of tracking.

The Epanechnikov profile is advantageous when calculating the new center location,

$$l_r = \frac{\Sigma_{i,j} i \cdot p \cdot \ (-k'_d)}{\Sigma_{i,j} p \cdot \ \left(-k'_d\right)}, \tag{2.8}$$

where k'_d is the derivative of the kernel k_d. There would also be a similar function, namely l_c, which would be the new column location, while l_r is the new row location. Also note that if k_d has a Epanechnikov profile, its derivative is simply a constant, and may be ignored. Note this is the same as taking a weighted average, which means the new center location can simply be calculated via moments. Ignoring k'_d, we can calculate the moment as

$$m_{00} = \sum_{i,j} p(i,j),\ m_{10} = \sum_{i,j} i \cdot p(i,j),\ m_{01} = \sum_{i,j} j \cdot p(i,j). \tag{2.9}$$

Here, m_{00} is effectively the "area" of p, while m_{10} and m_{01} are the first order moments. This leads us, when using an Epanechnikov kernel, to the relation

$$l_r = \frac{m_{10}}{m_{00}} \text{ and } l_c = \frac{m_{01}}{m_{00}}. \tag{2.10}$$

The use of higher order moments may be used to get orientation information, including a rotation. This allows for an adaptive MS algorithm to be developed that can adapt the window size to the object on the fly, namely the CAMSHIFT algorithm. Discussion of this algorithm is deferred to Bradski 1998 [11].

This is essentially the whole MS algorithm. The calculation (or recalculation) of p and q is repeated at each newly found central location, (l_r, l_c), until the central location changes less than a certain preset amount; generally 1 or $\sqrt{2}$. This distance is calculated by finding the Euclidean distance between the newest location and the last known, via:

$$d = \sqrt{(l_r(new) - l_r(last))^2 + (l_c(new) - l_c(last))^2}. \tag{2.11}$$

This kind of iteration is perfect for using a while loop, which will be shown in Appendix B.1.3. Applications of this algorithm will be given in Section 3.

2.3 Kalman Filters and Variants

There are various Kalman filters (KF), each with a slightly different implementation, however, the basic frame of the KF is roughly the same. The power of the KF is from how it can be recursively calculated, making it both efficient and powerful [12]. The general scheme is:

1. Initialize the filter by giving it a starting state and process/state covariance estimation.

2. Step state forward a time step using the state model, and update the process error estimation.
3. Make a measurement of the state.
4. Calculate the Kalman factor, based on the predicted state, measured state, process error estimation, and the measurement covariance.
5. Update the process error estimation.
6. Repeat steps 2-5 as long as desired.

KFs were originally introduced by Kalman in 1960 to solve the problem of predicting random signals [13]. They are able to smooth noisy signals, and are used in signal detection. More recently, adaptive Kalman filters (AKF) have been used for noise identification, or in conjunction with GPS (Global Positioning Systems) for vehicle navigation [14, 15]. There are many variations of KFs, such as the extended Kalman filter (EKF, uses a non-linear state prediction model), unscented Kalman filter (UKF, which is a modification of the EKF for highly non-linear models), and adaptive Kalman filter (AKF). This work only focuses on the standard Kalman filter (Section 2.3.1), as well as the adaptive Kalman filter (Section 2.3.2). The notation will be built up for the standard KF, then the AKF will be built on top of this.

2.3.1 Kalman Filter

All KFs can be expressed in terms of a prediction step and a measurement step. The prediction step requires a model, and is denoted A. A simple, physics based model is

$$A = \begin{bmatrix} 1 & \triangle t & 0.5\left(\triangle t\right)^2 \\ 0 & 1 & \triangle t \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.12}$$

This will simply transform a state matrix, $\mathbf{X}$, forward using the standard kinematic mechanics equations. We then add random Gaussian noise to this (ν, μ), with a mean of zero, to give us a predicted state,

$$\hat{\mathbf{X}}_n = A\mathbf{X}_{n-1} + \nu\left(n-1\right). \tag{2.13}$$

Before the measurement step may be performed, we have to update the process (prediction/state) error estimation. This is performed by the process covariance, Q, and the state model:

$$\hat{\mathbf{P}}_n = A\mathbf{P}_{n-1}A^{\mathrm{T}} + Q. \tag{2.14}$$

Now the measurement must be performed. For this work, the state, $\mathbf{X}$, is a three-vector, with components of position, velocity and acceleration. For ease, two KFs are

applied, one in the row direction, one in the column direction. The measurement consists of the objects location using the MS algorithm, while the velocity and acceleration are calculated using the last N positions. The measurement, z_n is then used to calculate the new state X_n, after the Kalman factor is calculated. The starting point of the measurement using MS is generally the last known location. However, when using MS in conjunction with a KF, the MS starts its measurement at the state estimated position, $\hat{\mathbf{X}}_n(1)$ from equation (2.13).

The Kalman factor (also known as the Kalman gain) is used as a weighting scalar* to tell whether the measurement or prediction should be trusted. It is denoted K_n and can be calculated by

$$K_n = \frac{\hat{\mathbf{P}}_n \mathbf{H}^{\mathrm{T}}}{\mathbf{H}\hat{\mathbf{P}}_n \mathbf{H}^{\mathrm{T}} + R}. \tag{2.15}$$

H is a factor used to extract only the desired information (e.g., the position), and is $H^{\mathrm{T}} = [1, 0, 0]$. R is the measurement covariance, and is a scalar for this work†.

The actual state the KF determines the system to be in is calculated by

$$\mathbf{X}_n = \hat{\mathbf{X}}_n + K_n \left(z_n - \mathbf{H}\hat{\mathbf{X}}_n \right), \tag{2.16}$$

with estimated error of

$$\mathbf{P}_n = \left(I - K_n \mathbf{H} \right) \hat{\mathbf{P}}_n. \tag{2.17}$$

The values for $\mathbf{X}_n \rightarrow \mathbf{X}_{n-1}$ in equation (2.13), and similarly for equation (2.14) on the next frame. Sample MATLAB code can be found in Appendix B.2.

2.3.2 Adaptive Kalman Filter

The standard KF requires that the noise covariance for both the measurement and the system be constant, however, this is generally not the case. The adaptive Kalman filter (AKF) adjusts these parameters, allowing for more accurate predictions and a better estimation of the Kalman factor. The form of the AKF used for this work follows the form used by Li et al. [16].

This technique requires a history of the strength of the measurement step in order to calculate the new covariance matrices. The actual strength of the measurement is discussed in detail in Section 2.4. For now, we will just denote this measurement as ρ. There is

*The Kalman factor can be a vector, but for our purposes, it represents a scalar.

†Note that for this work, Q is not a scalar while R is. This is because the estimated process state has three dimensions (position, velocity, acceleration), while the measurement only has one (position). To foreshadow the next section, note that Q and R do not have a subscript n. This is to emphasize that once initialized, their values remain constant. This is where the adaptive Kalman filter comes in, by *adapting* them at each step.

also an associated threshold, T_ρ, above which will be deemed a reliable measurement. To calculate the noise covariance for the system, Q, we first calculate factors of σ_Q.

$$\sigma_Q = \begin{cases} \rho & \rho \geq T_\rho \\ 0 & \text{Otherwise} \end{cases}. \tag{2.18}$$

From a the known past values of ρ, we can calculate the full σ_Q with

$$\sigma_Q(n-1)^2 = (1-\lambda)\sigma_Q(n-1)^2 + \lambda\sigma_Q(n-2)^2. \tag{2.19}$$

$$Q(n-1) = \begin{bmatrix} \sigma_Q(n-1)^2 & 0 & 0 \\ 0 & 0.5\sigma_Q(n-1)^2 & 0 \\ 0 & 0 & 0.2\sigma_Q(n-1)^2 \end{bmatrix} \tag{2.20}$$

Note that Q is calculated before the measurement step of the current frame, n. The factor λ is called a forgetting factor, and is defined as $0 \leq \lambda \leq 1$. Its purpose is to tell how quickly the AKF "forgets" the strength of previous measurements.

Similarly, we must also calculate the updated measurement covariance, $R = \sigma_R(n)^2$. σ_R can be calculated with

$$\sigma_R = \begin{cases} 1-\rho & \rho \geq T_\rho \\ C & \text{Otherwise} \end{cases}, \tag{2.21}$$

where C is a large constant. The point of this constant will be clear in a moment.

$$\sigma_R(n)^2 = (1-\lambda)\sigma_R(n)^2 + \lambda\sigma_R(n-1)^2 \tag{2.22}$$

To end the discussion on the AKF, it is instructive to note how these factors of Q and R effect the location output of the filter. This can be done by looking at equations (2.14) and (2.15-2.16). For equation (2.14), if the measurements are poor, Q approaches zero, and the process estimated error is not adjusted, which adjusts the Kalman factor. If the current measurement is poor, R becomes large, making the Kalman factor small, approaching zero. This is where the constant, C, is important. The higher C is, the weaker the Kalman factor's effect on the new position is (equation (2.16)). Basically, the measurement of the correlation between the measurement and the target model decides whether or not the measurement step is trusted or not. Therefore, having an accurate way of measuring correlation is critical to the accuracy of this AKF.

2.4 Correlation Coefficients

As stated above, the choice of correlation coefficient is very important to the accuracy of a KF. Essentially, this coefficient is flicking a switch to turn off the measurement step temporarily. If this is performed too late, the actual object motion can be contaminated

Figure 2.1: This is an infrared image of an MX2 aerobatic airplane. It has fixed striations due to the readout card, making the airplane appear to have more structure than actually visible.

with noise, causing inaccurate location/state predictions. This section will discuss which correlation coefficient to choose under which circumstances. Two popular metrics are discussed, the Bhattacharyya coefficient (Section 2.4.1) and the Pearson product-moment coefficient (Section 2.4.2).

2.4.1 Bhattacharyya Correlation Coefficient

The Bhattacharyya Correlation Coefficient (BC) is commonly used in conjunction with the MS algorithm [10, 16]. It calculates the similarity of two different probability distributions. The BC is commonly used because it has properties similar to a divergence metric [17]. This means the correlation makes use of changes within the distributions of interest. It is calculated by

$$\rho_{\mathrm{BC}} = \sum_{i,j} \sqrt{p_{i,j} q_{i,j}}. \tag{2.23}$$

However, the use of a divergence measure for distribution correlation isn't always best. An example is the false structure or noise due to readout pattern or fixed pattern noise. This is not uncommon in infrared images, and an example is shown in Figure 2.1, as well as in Figures 1.2 and 1.4. These striations are clearly a fixed pattern noise, and have no physical meaning. However, a divergence like measure would treat these as definite structure. This can cause a false correlation measurement, either higher or lower than the actual correlation. As stated above, an accurate measurement is what allows a KF to work properly.

2.4.2 Pearson Product-Moment Correlation Coefficient

The Pearson product-moment correlation coefficient (PPMCC) is another method for calculating the correlation between two distributions. Unlike BC, PPMCC is a linear correlation, and only looks at the overall distribution, not the internal structure. It is calculated using

$$\rho_{\text{PPMCC}} = \frac{\sum_{i,j} (p_{i,j} - \bar{p})(q_{i,j} - \bar{q})}{\left(\sum_{i,j} (p_{i,j} - \bar{p})\right)^2 \left(\sum_{i,j} (q_{i,j} - \bar{q})\right)^2}. \tag{2.24}$$

This type of metric, although more "primitive" than the BC should still be used in certain situations. Situations where the imagery is of low resolution, or if there is any fixed pattern noise that may give a false reading of correlation. A comparison of these two different metrics for different resolution imagery is shown in Chapter 3.

2.5 Polynomial Object Path Tracking

This proposed method combines the utility of MS with the prediction ability of the AKF, plus the addition of a polynomial fit (PFAKF, briefly outlined in [18]). This polynomial fit will allow the normal combination of AKF and MS to predict object motion that is non-linear. Since the AKF's linear state model seems to be robust, the use of a non-linear state model will still not predict non-linear motion. The idea behind this algorithm is to take the AKF as it is, but modify the path it predicts the object to be traveling along. This is done with the use of a polynomial fit to the history of known locations. The procedure can be viewed as this:

1. Initiate AKF as described in Section 2.3.2.
2. Calculated the distance between the last known location and linear prediction.
3. Calculate polynomial equation that has a least squares fit to a number of last known locations.
4. Find the point along the polynomial curve that is the same distance from the starting point. This makes sure the object still moves the same amount, however, it will be moving along the curved path.
5. Continue the AKF/MS as discussed above.

The actual method used for finding the polynomial fit is up to the user, any least squares method should work the same. This work used the built-in MATLAB "fit.m" function (sample code can be found in Appendix B.3).

If we choose to look at a second degree polynomial fit (quadratic), then the equation we are trying to solve for is of the form

$$y_r = p_1 x_c^2 + p_2 x_c + p_3. \tag{2.25}$$

To calculate the distance along this curve, simply integrate along the arc length

$$d_{arc} = \int_{\mathbf{X}_{n-1}}^{\hat{\mathbf{X}}_n} \left(1 + \left(\frac{dy_r}{dx_c}\right)^2\right)^{1/2} dx_c. \tag{2.26}$$

$\hat{\mathbf{X}}_n$ is found by increasing it by a given amount until $d_{arc} \simeq \|\mathbf{X}_{n-1} - \hat{\mathbf{X}}_n\|$. In other words, until the linear projected distance is roughly equal to the length of the arc along our fit function. This new column predicted position can then be plugged into equation (2.25) to find the new row predicted position. This step allows the filter to still calculate the proper position, velocity, and acceleration, while following a non-linear path, as desired.

There are a couple of potential issues with using a procedure such as this one, most of which stems from the drastic change in appearance of objects in the infrared (discussed further in Section 1.2). For objects that go untracked for longer and longer periods of time, the odds of the orientation and appearance changes increases. This change in appearance can cause a drop in the correlation between the model and candidate. Due to the importance of the correlation coefficient in the KF process, this could force the filter to fail, even though the prediction actually found the object of interest again. The object's change in appearance could be due to orientation change (profile versus head-on) or intensity changes*.

Examples of the difficulties will be shown in Section 3.2. One way to deal with this issue is to have a "timer" that keeps track of how long the measurement step has been below the threshold level. After a predetermined amount of time, drop the threshold by 60% to see if the object may be acquired. Although this method still has some difficulties with tracking, it is still more likely to find an object by using the polynomial fit, making the PFAKF more robust than the AKF on its own.

*Much of the data we collected using a Kodak KIR-310 PtSi with a 3.2-4.1 μm filter. This is between the reflective and emissive part of the infrared, so specular reflections as well as emitted radiation can cause large changes in intensity based on orientation.

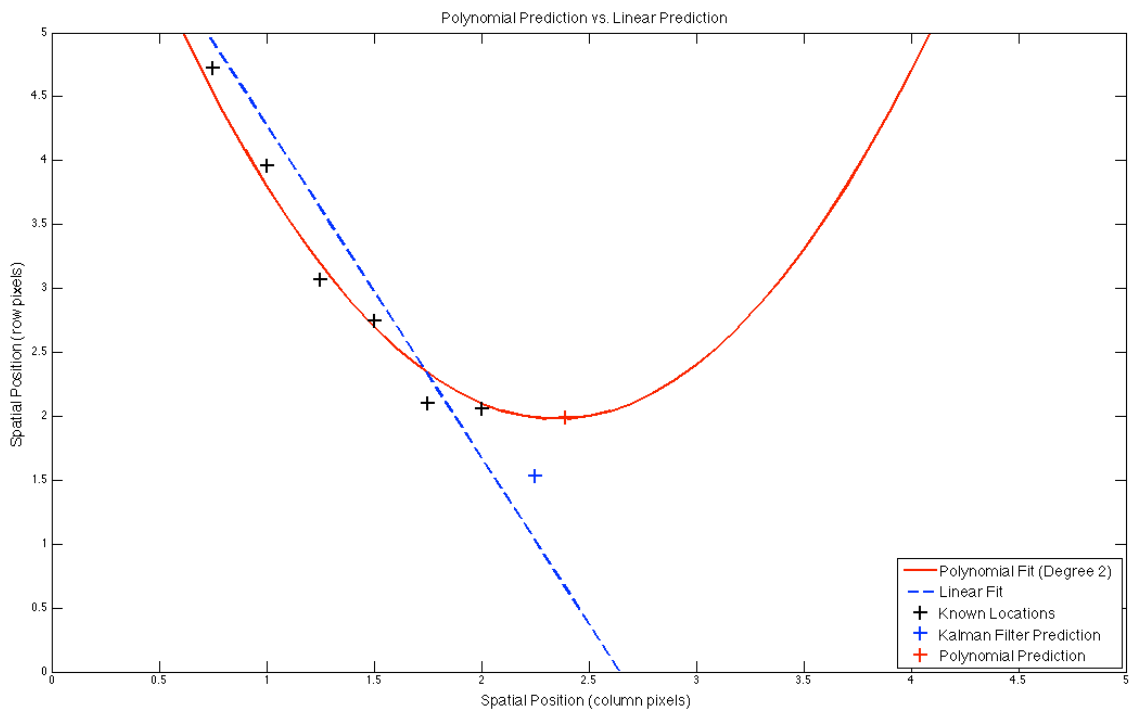


Figure 2.2: The is figure shows how the projected positions differ for an arbitrary function. The two curves were given the same data points (black "+"). The blue line represents a linear fit of the points, including the KF projection (blue "+"). The red curve represents the polynomial fit to just the known data. The polynomial projected point (red "+") was found by integrating along the curve until the distance between the last known position to the KF project point was the same as the last known point and a point on the curve.

Bibliography

[1] Sheikh, Y.; Shah, M.;, "Bayesian modeling of dynamic scenes for object detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol.27, no.11, pp.1778-1792, Nov. 2005, doi: 10.1109/TPAMI.2005.213.

[2] Moghaddam, B.; Pentland, A.;, "Probabilistic visual learning for object detection," *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pp.786-793, 20-23 Jun 1995, doi: 10.1109/ICCV.1995.466858.

[3] Torralba, A.; Murphy, K.P.; Freeman, W.T.;, "Sharing Visual Features for Multiclass and Multiview Object Detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol.29, no.5, pp.854-869, May 2007, doi: 10.1109/TPAMI.2007.1055.

[4] Connor, D.; Limb, J.;, "Properties of Frame-Difference Signals Generated by Moving Images," *Communications, IEEE Transactions on*, vol.22, no.10, pp. 1564- 1575, Oct 1974, doi: 10.1109/TCOM.1974.1092083.

[5] Jain, Ramesh; Nagel, H.-H.;, "On the Analysis of Accumulative Difference Pictures from Image Sequences of Real World Scenes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol.PAMI-1, no.2, pp.206-214, April 1979, doi: 10.1109/TPAMI.1979.4766907.

[6] ATR Algorithm Development Image Database, *Military Sensing Information Analysis Center*, 25 July 2008.

[7] Collins, Robert; Lipton, Alan; Kanade, Takeo; et al.;, "A System for Video Surveillance and Monitoring," *Tech. report CMU-RI-TR-00-12*, Robotics Institute, Carnegie Mellon University, May, 2000.

[8] Fukunaga, K.; Hostetler, L.;, "The estimation of the gradient of a density function, with applications in pattern recognition," *Information Theory, IEEE Transactions on*, vol.21, no.1, pp. 32- 40, Jan 1975, doi: 10.1109/TIT.1975.1055330.

[9] Comaniciu, Dorin; Meer, Peter;, "Mean Shift: A Robust Approach Toward Feature Space Analysis," *IEEE Transactions on Pattern Analysis and Machine Learning*, Vol. 24, Issue 5. pp. 603-619, 2002.

[10] Comaniciu, Dorin; Ramesh, Visvanathan; Meer, Peter;, "Kernel-Based Object Tracking," *IEEE Transactions on Pattern Analysis and Machine Learning*, Vol. 25, Issue 5, pp. 564-577, 2003.

[11] Bradski, Gary R., "Computer Vision Face Tracking For Use in a Perceptual User Interface," *Intel Technology Journal*, Q2, pp. 1-15, 1998.

[12] Welch, Greg; Bishop, Gary; "An Introduction to the Kalman Filter," *University of North Carolina at Chapel Hill*, pp. 1-16, 2006.

[13] Kalman, R. E.; "A New Approach to Linear Filtering and Prediction Problems," *Transaction of the ASMEJournal of Basic Engineering*, pp. 35-45, March, 1960.

[14] Oussalah, M.; De Schutter, J., "Adaptive Kalman Filter for Noise Identification," *Proceedings of the International Conference on Noise and Vibration Engineering*, pp. 1225-1232, 2000.

[15] Hu, Congwei; Chen, Wu; Chen, Yongqi Chen; Liu, Dajie;, "Adaptive Kalman Filtering for Vehicle Navigation," *Journal of Global Positioning Systems*, Vol. 2, No. 1, pp. 42-47, 2003.

[16] Li, Xiaohe; Zhang, Taiyi; Shen, Xiaodong; Sun, Jiancheng, "Object tracking using an adaptive Kalman filter combined with mean shift," *Optical Engineering Letters*, Vol. 49, Issue 2, pp. 020503-1-3, 2010.

[17] T. Kailath;, "The Divergence and Bhattacharyya Distance Measures in Signal Selection, *IEEE Trans. Comm. Technology*, Vol. 15, pp. 52-60, 1967.

[18] Kyle Ausfeld, Zoran Ninkov, Paul P. K. Lee, J. Daniel Newman and Gregory Gosian, "Polynomial fitting adaptive Kalman filter tracking and choice of correlation coefficient", *Proc. SPIE* 8395, 83950R (2012).

Chapter 3

Results

This chapter contains the application of these algorithms to various situations, showing both the strengths and weaknesses of the algorithms presented in Section 2. The first discussion that must be dealt with is that of correlation coefficient and which one should be used when.

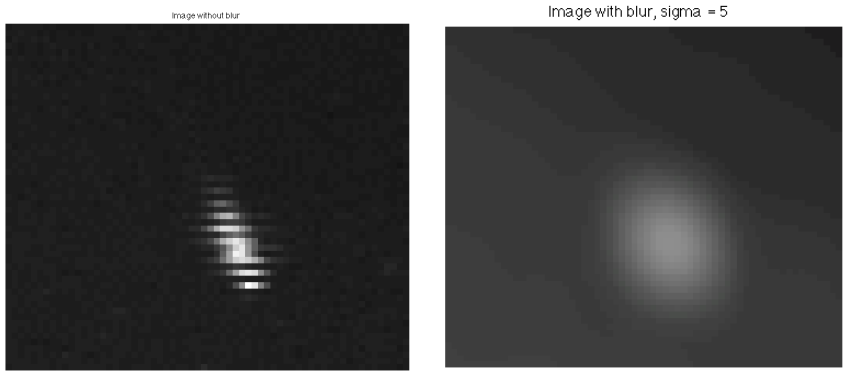
3.1 Correlation Coefficient Comparison

The correlation coefficients introduced in Section 2.4 both have attributes that make them strong metrics for comparison under different circumstances. A test was performed to look at the shape of a couple different objects and how their correlation coefficient changes as the object is shifted off from center. The ideal function is close to a delta, as it would give a very sharp location for when they are aligned.

The first test was performed using an image of the MX2 aerobatic airplane taken during the 2011 ESL Rochester International Air Show [1] (camera information found in Section 1.2). The image was then blurred using a Gaussian with varying standard deviations ($\sigma = 1 - 5$). This would simulate the difference between the an object with very clear edges and structure (in this case, false structure) versus a lower resolution, more blurry image. The original image is shown in Figure 3.1(a) while a blurred version is shown in Figure 3.1(b) (with $sigma = 5$).

The correlation coefficients were calculated using equations (2.23) and (2.24). At each different blur level, the correlations were calculated by offsetting the image from it's original location by up to 5 pixels in each direction, creating a map of correlation versus offset. This gives an idea of how well an object can be distinguished from itself when offset by varying amounts. The ideal metric would produce something close to a Gaussian with a single clear peak of one, and would rapidly drop to zero once the object is shifted well outside the object's central region.

The MX2 was chosen because it demonstrates how BC may appear to be a better



(a) (b)

Figure 3.1: (a) MX2 aerobatic airplane during the ESL Rochester International Air Show in 2011. Note striation pattern due to the way the camera reads data off the array. (b) Same image as in (a), however, a Gaussian blur was performed with $\sigma = 5$ to eliminate the fixed pattern noise present. Note: contrast enhanced for clarity, further calculations performed on original image.

metric because of the localized peaks. However, the data has a fixed pattern noise, due to how the image was read off the detector. This pattern is exploited by BC's divergence-like properties, even though this extra "information" is in fact noise. Figures 3.2 and 3.3 show the correlation comparison of BC versus PPMCC for the blurred image. In Figure 3.2, it is important to note that it appears to have very distinct peaks, each separated by an area of zero correlation. This is due to the divergence characteristics of this method of correlation, giving rise to a measurement based on structure of the object in question. However, also note the fixed pattern noise mentioned above and how this structure is not real structure intrinsic to the airplane, but noise from the camera used on for the data collect. The PPMCC has similar results, however the peaks are connected by areas that do not drop to zero, making this correlation map or space more easily searched to find the absolute maximum. This is because it ignored the fine object structure which throws the BC off, making PPMCC better for use on noisy data.

Figure 3.3 shows the correlation comparison on the blurred image. This image represents a low resolution image, showing that with low noise and low structure, the PPMCC is still a better metric. This is because BC has very little divergence information to work with, so it results in a large area for a peak, while not dropping down to zero well outside the region of interest. PPMCC is able to drop closer to zero on the edge while having a much smaller, more defined peak. *In short, PPMCC is a strong metric to use for correlation when working with either noisy data or low resolution data where either false structure or little structure is present.* These situations are not uncommon when working

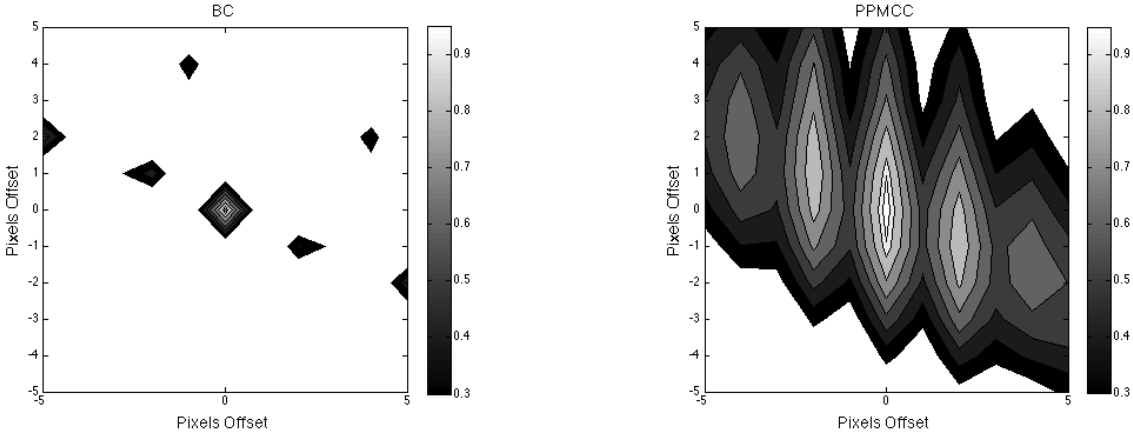


Figure 3.2: Correlation comparison on the MX2 aerobatic airplane shown in Figure 3.1(a). Note the well defined peaks for the BC, while PPMCC has defined peaks but are more spread out. The peaks here may seem as though it corresponds to a stronger metric, but recall these patterns are noise and do not represent real-world information.

with infrared data, due to the expense and availability of high quality instruments, as well as the noise due to dark current if a system is not cooled.

3.2 Algorithm Testing

This section will show the strengths and weaknesses of the three tracking algorithms discussed in Chapter 2. In lieu of breaking this section up by algorithm, it is easier to discuss algorithm results based on each data set tested on.

3.2.1 RIT Parking Lot Data

The first data set testing was performed on was taken overlooking parking lots F and J from the roof of the Chester F. Carlson Center for Imaging Science building. The camera used was an EzTherm microbolometer[2] with a $\sim 10\mu$m filter. The data collect was performed on 8 October 2010 at around 14:00. Visible data was also collected for demonstrative purposes but was not used for tracking. A visible image of the scene can be seen in Figure 3.4, while the infrared is shown in Figure 3.5. A person will be tracked because a person happens to be about the same size as many vehicles are during persistence surveillance situations.

Figure 3.6 shows a small area of the scene shown in Figure 3.5. The leftmost image shows that the MS algorithm has already lost the person, due to an occlusion by a passing van. However, because the AKF and PFAKF were able to identify the van as an occlusion

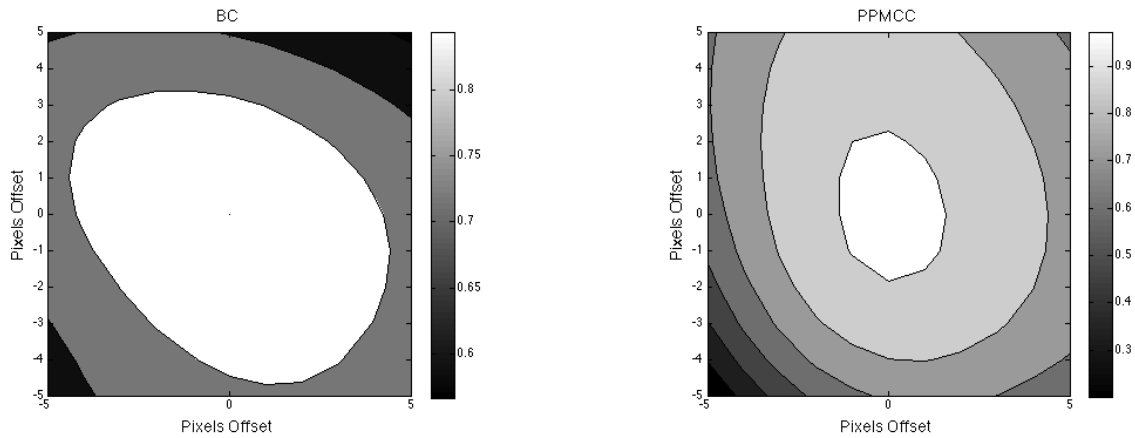


Figure 3.3: Correlation comparison on the MX2 aerobatic airplane shown in Figure 3.1(b). The BC has a very large area of high correlation and does not drop off to zero as quickly as would be desirable. However, PPMCC has a much more narrow peak and drops off more quickly, giving a more accurate representation of actual correlation. This demonstrates that with very low detailed objects (such as common in low resolution imagery) the PPMCC should be used as a correlation coefficient.

via a drop in correlation between the target candidate and the target model. This allowed these predictive algorithms to compensate by predicting the person's path until the van has passed, reacquiring the person and continues to track.

Although the predictive algorithms were able to continue tracking the person after the van occluded, another problem arises. The problem is shown in Figure 3.7. This figure shows that both algorithms loose the person in the end, due to the near-by passing of a person on a bicycle with similar intensity. The fact that the person and the biker are of similar appearance causes the MS algorithm, used for the measurement, to falsely move the measured center away from the person (shown in Figure 3.8(a)-3.8(c)). This false motion tripped up both the AKF and PFAKF in different ways, both loosing the person in the end. This shows one limitation in the use of the MS algorithm as a measurement algorithm.

3.2.2 ESL Rochester International Air Show

The next data set these algorithms were tested on was collected from the roof of the Chester F. Carlson Center for Imaging Science building again. This time the camera was the Kodak KIR-310 PtSi 640x480 pixel array with a germanium lens with focal length of 200 mm and an f/2. The filter was $3.2 - 4.1\mu$m and was collected at 24 frames per second. The targets were various airplanes flying in the ESL Rochester International Air Show in

Figure 3.4: Visible image overlooking RIT parking lots F and J. Note: vertical lines represent the field of view for the infrared imagery, shown in Figure 3.5.

2011[1]. The distance from RIT to the Rochester International Airport is roughly 4 km. The aircraft focused on was an MX2 aerobatic airplane performing various dives beneath the tree-line, reappearing a short time later (visible image shown in Figure 3.9, the full field shown in Figure 3.10).

There were two sequences chosen to demonstrate the strengths and weaknesses associated with the PFAKF. The first sequence is shown in Figure 3.11, and shows that it is able to acquire the target after the occlusion, while neither the MS or AKF algorithms were able to. The MS algorithm just finds an area in the trees and stays there, while the AKF predicts a linear path, eventually moving the track right off the screen. The PFAKF is able to predict a polynomial due to the last minute pulling up just before dropping below the trees, allowing a path to be predicted. After the MX2 appears from behind the trees, the PFAKF is able to give the MS algorithm a close enough prediction to find the target again, and tracks as usual.

As noted in Section 2.5, the threshold for correlation had to be dropped after 50 frames (~ 2 seconds) in order for the Kalman filter to trust the measurement again. This is because of the change in orientation, as well as in shape. Figure 3.12 shows a plot of the PPMCC correlation during all three tracking algorithm runs. Note how there is a rise in correlation which drops again before rising and staying risen. This is because the

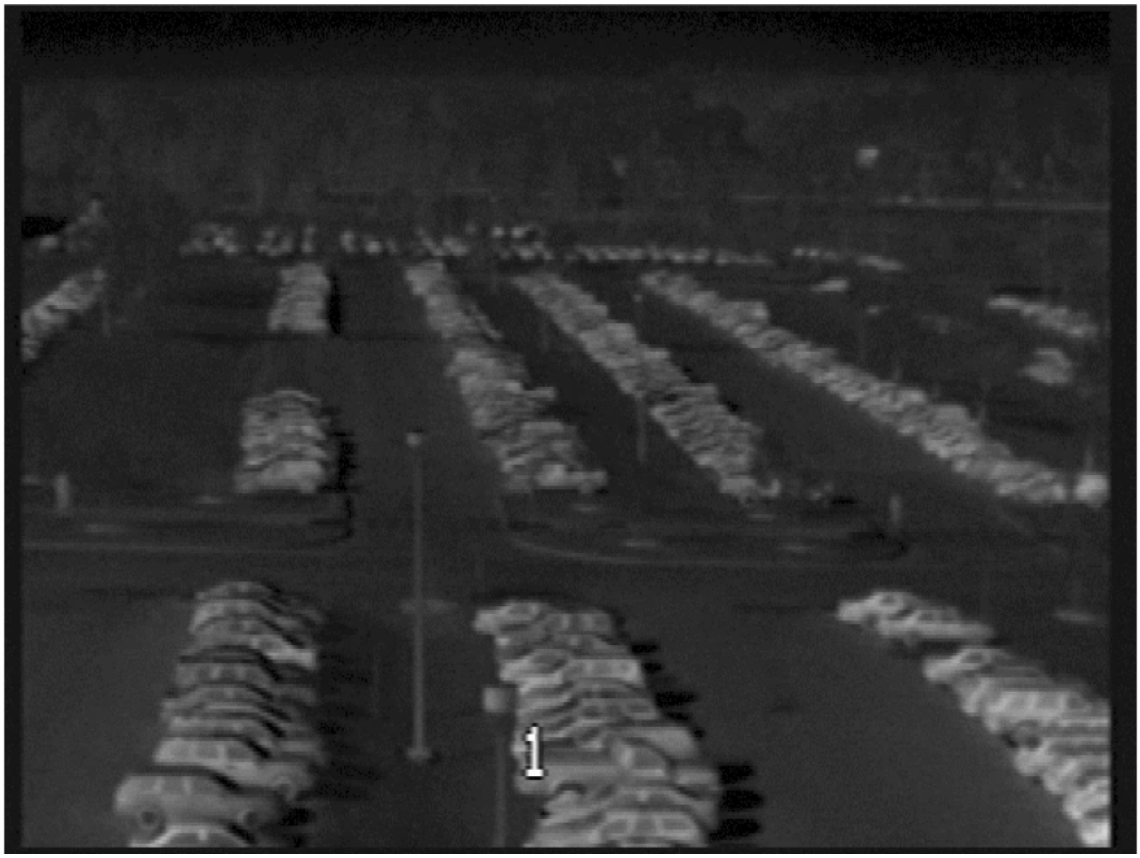


Figure 3.5: Infrared image overlooking RIT parking lots F and J. Note the #1 seen in the bottom of the image is a watermark of what filter the camera was using. A black box was put over this watermark during tracking to avoid confusion.

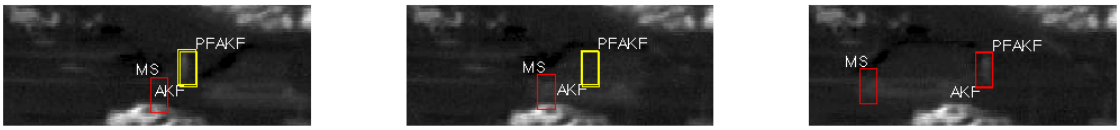


Figure 3.6: Section of scene from Figure 3.5. The person being tracked gets occluded by a vehicle. Three tracking algorithms were used, however only the AKF and PFAKF were able to continue tracking after the occlusion. Note that all three algorithms were tracking the person using the measurement prior to the shown frames.

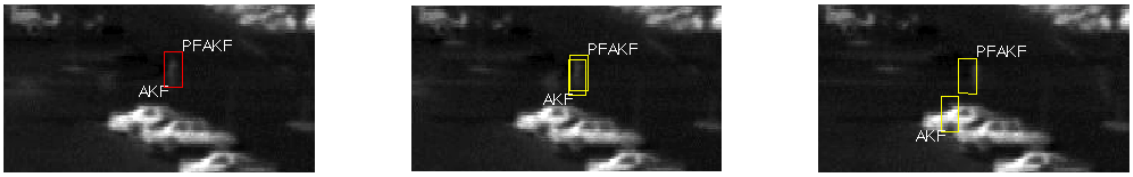


Figure 3.7: This sequence is a continuation from the tracking sequence shown in Figure 3.6. Note that when the biker passes by the person walking, both algorithms are "deflected," albeit in different ways. This is due to the measurement step, MS, causing false motion by picking up the biker and person, instead of just the person. This is shown in Figure 3.2.1.

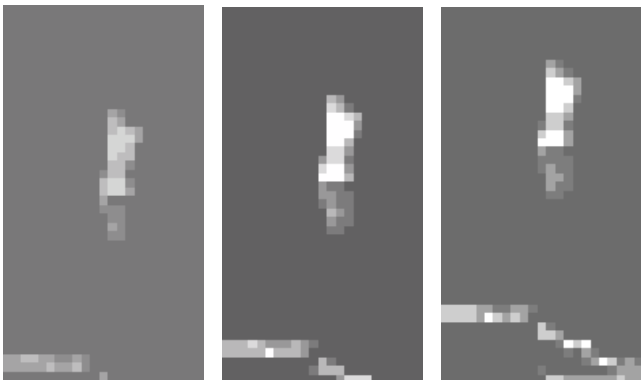
measurement wants to say the tower is the same as the plane, which is ignored because the correlation threshold has not dropped yet. Also note the line corresponding to the AKF stops short because the prediction runs off of the screen.

Figure 3.13 shows a difficulty in tracking in the infrared under these conditions. The sequence is from a similar run as Figure 3.11, except the airplane's appearance is drastically different after the occlusion. This is because after diving, the airplane flies away from the camera, changing both the shape of the object as well as the amount of signature received from the airplane. Recall that this camera has a filter passband of $3.2 - 4.1\mu$m, making it within the crossover region between emissive and reflective dominated regions. This means the airplane signature was likely dominated by glint during the initial dive and while climbing the signature was no longer from glint, making it very faint. So although the PFAKF made a close approximation as to where the airplane would end up, the measurement was never strong enough to start tracking again. This demonstrates a weakness with using the MS algorithm as a measurement step, meaning another measurement may provide more robust tracking with the AKF and the PFAKF. A further discussion of this will be undertaken in Chapter 4.

3.2.3 WAPS

The Wide Area Persistence Surveillance (WAPS) sensor also provided a potential dataset to test algorithms on, testing more realistic data. The sensor is ITT Exelis Geospatial Systems' and was flown on 13 August 2010 around 09:00 EST at 2 Hz and centered around 4μm. The data came in the NITF standard, with the images embedded as JPEG2000. This data presented a few challenges, from extracting the images to lace together as a video for the tracking algorithm to the alignment of the frames.

Extracting the image from the NITF format could have be performed manually using a program such as ITT Exelis VIS' ENVI. However, due to the number of images, it was more advantageous to create a MATLAB script that was able to open and extract the images. From the extracted images, it is simple to put them into a video file, and the AVI



(a) (b) (c)

Figure 3.8: (a) This is the first MS iteration when the biker is nearby. Note how you can already see the probability is no longer going to be centered on the person. (b) Same image as in (a) except, as predicted, the center of the target is shifted to a location between the person and biker on the second MS iteration. (c) The third MS iteration shows the drastic change in measurement, which causes a false motion to be detected. From this, the Kalman filters predict false motion, causing a loss in target.

video format was chosen. The code is given in Appendix B.4.

The next step would be to register the images so accurate tracking could be performed. At first, the data from the header was used to distort the corner points to register the images but that data was only accurate to ∼100 m. Although there is IMU and bore angle data available, the registration of images using this technique was more involved than necessary for this work. Instead, an FFT-based registration technique was chosen.

FFT-Based Image Registration

This technique takes advantages of some properties of Fourier transforms, such as convolution in space is simply multiplication in frequency space. Complete details are left to a reference [3]. Essentially, we want to create an edge image, identifying roads, and perform registration on these roads. This is very similar to performing a match filter, by convolving with a replica of what is trying to be "matched". The edge image is used for registration because roads should be the easiest thing to align and are simple to find with an edge detector. The edge detector used was the Sobel edge detection filter, although the Canny works as well.

To start, a zero-padded image is created with the first frame centered. This allows room for the second image to shift to where it needs to go, which shouldn't be centered for a moving platform. The analysis will be first performed on the second frame, then third and so on. I will define f_i as the current frame, and f_{i-1} as the previous frame we

Figure 3.9: Visible image of the MX2 aerobatic airplane from the ESL Rochester International Air Show website [1].

Figure 3.10: Infrared image of MX2 aerobatic airplane from the ESL Rochester International Air Show as seen from RIT.

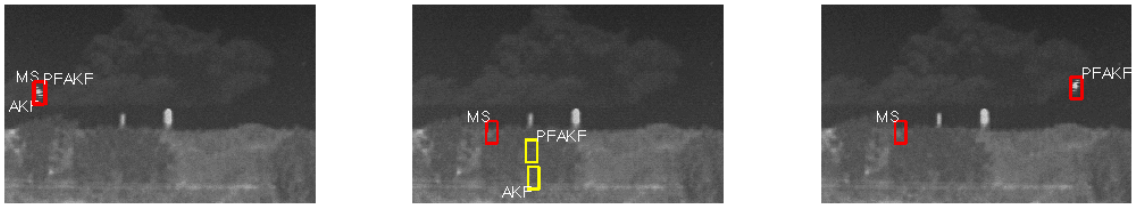


Figure 3.11: Infrared image sequence of the MX2 with all three tracking algorithms overlaid.

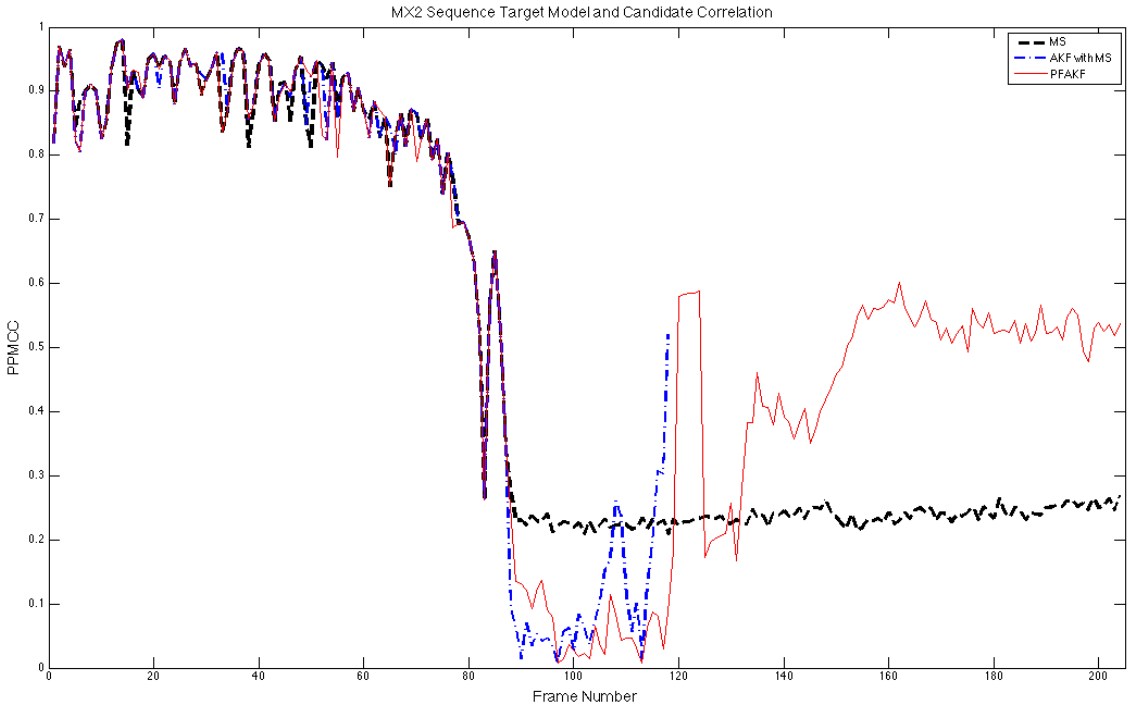


Figure 3.12: PPMCC correlations from the tracking results shown in Figure 3.11. Note the rapid drop in correlation as the plane passes below the tree-line, and an increase when it reappears, however at a lower value. Also note that the AKF with MS line stops short, due to the prediction being outside of the image window.

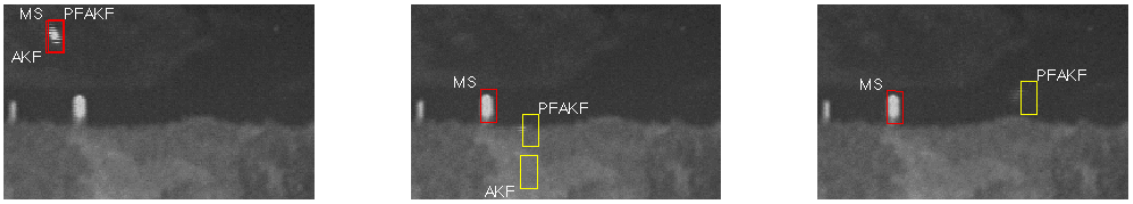


Figure 3.13: Infrared image sequence of the MX2 with all three tracking algorithms overlaid.

are registering to. To perform the match filter and to create a map of "correlations," we want the normalized convolution of f_i and f_{i-1}. Note that $F_i = \mathcal{F}\{f_i\}$ will denote the Fourier transform of f.

$$R_{\text{map}} = \frac{F_i^* F_{i-1}}{\|F_i^* F_{i-1}\|}, \tag{3.1}$$

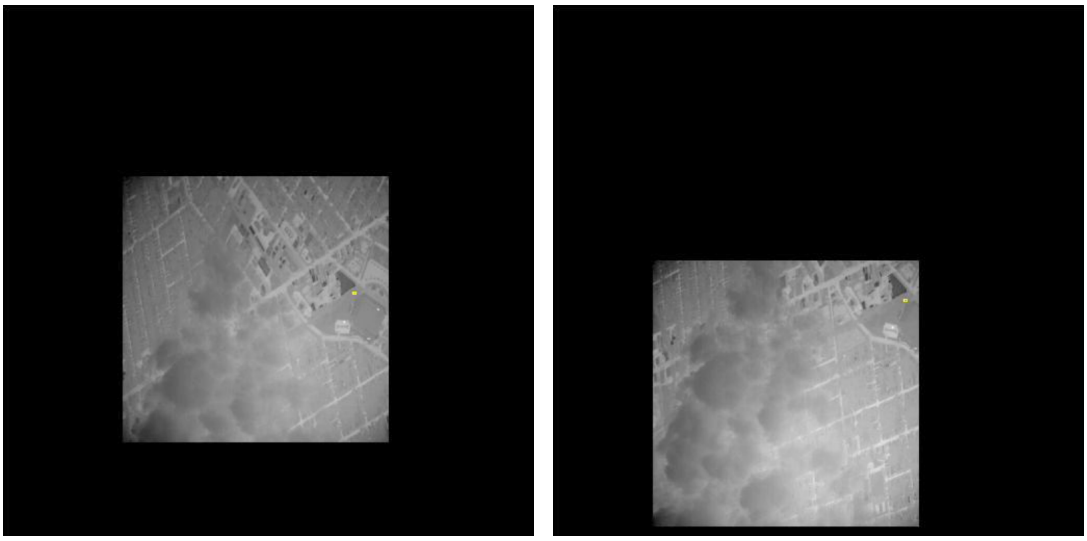
where * is the complex conjugate. The maximum point in this map of "correlations" gives the amount of displacement the current image needs to be shifted to be registered to the previous. Registration may also take scale and rotations into account [3], however, translational registration was deemed sufficient for this work. There are a couple of other tricky steps that are left to Appendix B.5 to explain, and mostly deal with indexing and keeping track of shifts from frame to frame.

After the registration was complete, object tracking can be performed. To test and see if tracking was even possible (as well as testing how well the registration was performed), a stationary, clearly distinct object was chosen from the scene and tracked using the MS algorithm. The object chosen was a light post overlooking a soccer field, and two frames are shown in Figure 3.14(a)-3.14(b). This demonstrates that objects in this dataset that are distinct from the background are able to be tracked using the simple translational registration.

Next step was to see if tracking could be performed on a vehicle. The MS algorithm was again chosen, because the variation in the accuracy of the registration would cause a KF to incorrectly estimate the position. Figure 3.15 shows the full frame from the sequence along with the region of interest for the following figure. The tracking is shown on two frames in Figures 3.16(a) and 3.16(b).

The MS algorithm was able to converge on the vehicle on the second frame (Figure 3.16(a)) but lost it by the next frame (Figure 3.16(b)). This is due to the similarity of the target vehicle and the background. The images were collected during the morning, so the road has not had much time to heat up. This makes the vehicles and the road very similar, but the problem has more to do with the signal from the side of the road. The intensity range for the vehicle is 5863-5983, while just off to the side of the road has intensity range 5815-5954. The amount of overlap easily confuses the MS algorithm which

object is the target and which is just background. The spatial kernel for the MS algorithm could be adjusted to be smaller, in an attempt to eliminate the side of the road confusing the algorithm. However, the distance between the vehicle and road-side is only a couple of pixels at the most. This means MS would not be able to find the target either, because it would have moved outside of the MS search window due to the low frame rate.



(a) (b)

Figure 3.14: This data is courtesy of ITT Exelis Geospatial Systems and is from the Wide Area Persistence Surveillance (WAPS) centered at 4 μm. The yellow box is the MS target tracking result. (a) An early frame in the sequence with the MS algorithm set to track a distinct target to see if translational registration is sufficient. The object is a light post over the Sahlen's Stadium in Rochester New York. (b) A later frame to show the light post is still being tracked, as well as to demonstrate how well the registration does for not taking scale and rotations into account. this works so well because the frame rate is high enough that rotational changes from frame to frame are sufficiently small enough.

3.2.4 Synthetic

To avoid the difficulties involved with the collection of data, radiometrically accurate synthetic data would be generated for the testing of tracking algorithms. The generation of this data was performed with many software packages, but DIRSIG (Digital Imaging and Remote Sensing Image Generation model)[4] is the main component. It contains built in 3D map of a suburban neighborhood near Rochester NY, called Megascene 1. It contains buildings, roads, trees and other materials found in the area. This allows DIRSIG and

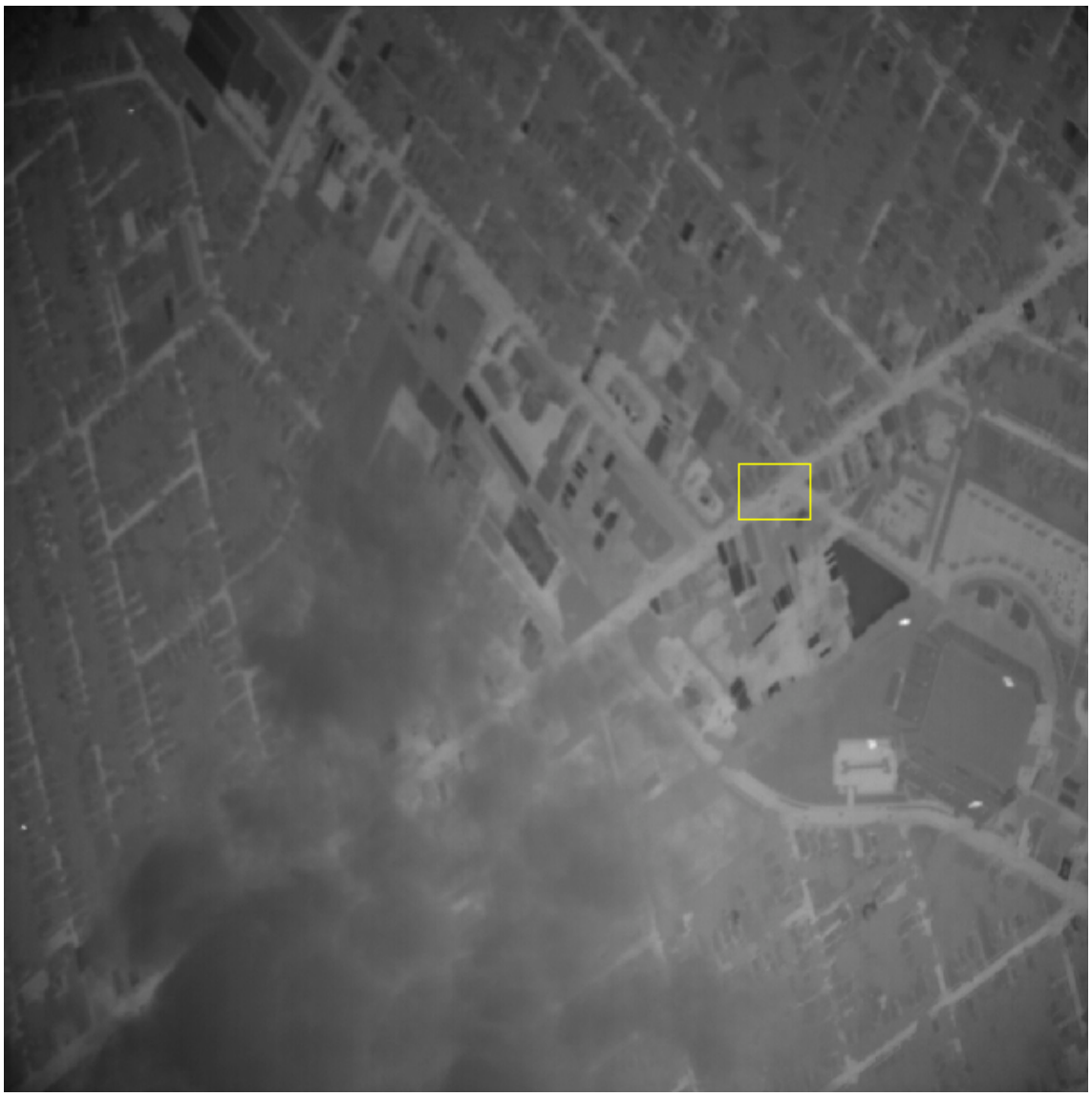


Figure 3.15: This data is courtesy of ITT Exelis Geospatial Systems and is from the Wide Area Persistence Surveillance (WAPS) centered at 4 μm. The yellow box indicates a region of interest shown in Figure 3.16(a)-3.16(b) where vehicle tracking is attempted.

other software packages to perform radiometrically accurate ray tracing, simulating real data collections. The vehicles are modeled separately in ThermoAnalytics MuSES (Multi-Service Electro-optic Signature)[5], and moved around within the scene using SUMO (Simulation of Urban Mobility)[6]. The method for creation of these simulated images will not be covered in this work. Instead, more information on the modeling of these scenes in the infrared will be deferred to Rhodes et al. 2010[7] and 2012[8].

The relevant frames for this work are from a simulated run in the Rochester area, taken mid August during the afternoon. The vehicle model used is that of a Toyota Tundra pickup. There were two bands simulated, one centered at 3.7 μm (Figure 3.17) and the other at 9 μm (Figure 3.18). The 3.7 μm wavelength was chosen because the WAPS imager (Section 3.2.3) is centered at 4 μm and the Kodak KIR-310 imager (Section 3.2.2) is centered at 3.7 μm. The 3.7 μm images represent the mid-wavelength infrared while the 9.0 μm images represent the long-wavelength infrared.

Tracking was only performed on the long-wavelength image sequences, shown in Figures 3.19 and 3.20. Figure 3.19 shows the results of a short straight-line path tracking with a tree occluding the vehicles momentarily. The MS algorithm does as expected, and looses the vehicle while passing under an overhanging tree. The AKF is expected to have good tracking results, however, it also looses the vehicle. The vehicle being tracked is lost because it slows down after passing under the tree, something an AKF cannot predict. The vehicles in the scene are not set to move at constant speeds in order to make the model more realistic, as well as to confuse tracking algorithms. The changes in speed are random and therefore unpredictable. The PFAKF was able to deal with the vehicle slowing down under the tree, however, this is alarming. This may be due to the way the distance in which the prediction is to predict is done in an iterative way. The error associated with calling the distance calculated correct will be an undershoot of the actual distance. This is not a big deal with large objects in small scenes, but does show up in large scenes with small targets. This could be corrected by using tighter tolerances within the iterative process.

Figure 3.20 shows a larger scene with a longer occlusion time. Once again, the vehicles in the scene change speed, both speeding up and slowing down. The MS algorithm again looses the target early on, yet the AKF and PFAKF are able to keep it through the first occluding tree. However, the PFAKF looses the target when it speeds up. A short time later, it momentarily mistakes another vehicle as the one it is after, causing confusion and forces the prediction to have no change, loosing it again. The AKF doesn't do much better. Although it is able to make it through the first two occluding trees, the vehicle speeds up, causing the AKF to lose it during the last tree. The linear prediction continues to track nothing, until a vehicle comes up from behind and is mistaken as the vehicle of interest. This demonstrates a limitation when modeling all vehicles to be identical. This makes tracking an object more difficult by making the target of interest indistinguishable from other targets. However, this dataset does show utility in displaying a tracking algorithm's

strengths and weaknesses.

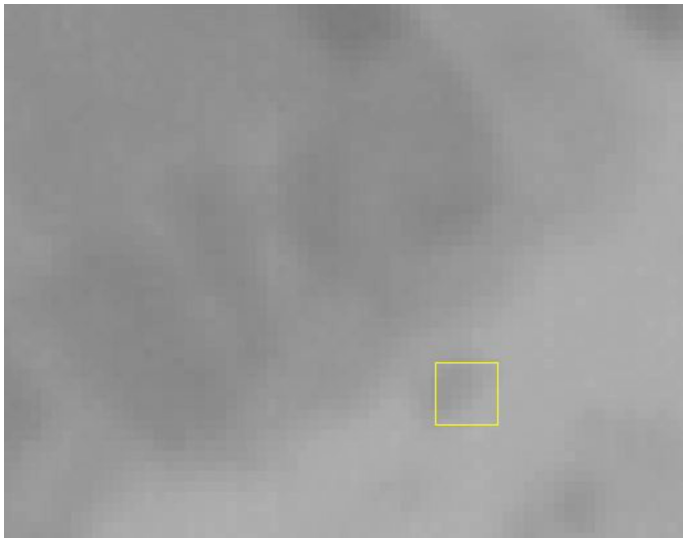
3.2.5 Bird Tracking

The main focus of this work is mostly on vehicle and people, making it very military/security focused. However, there are other applications of tracking algorithms that can be of interest. The migration of caribou in Alaska could be recorded and then tracked to study their migration habits to see if a pipeline would disrupt their movements. Or animals in a large group could be recorded to allow scientists to study animal group behavior.

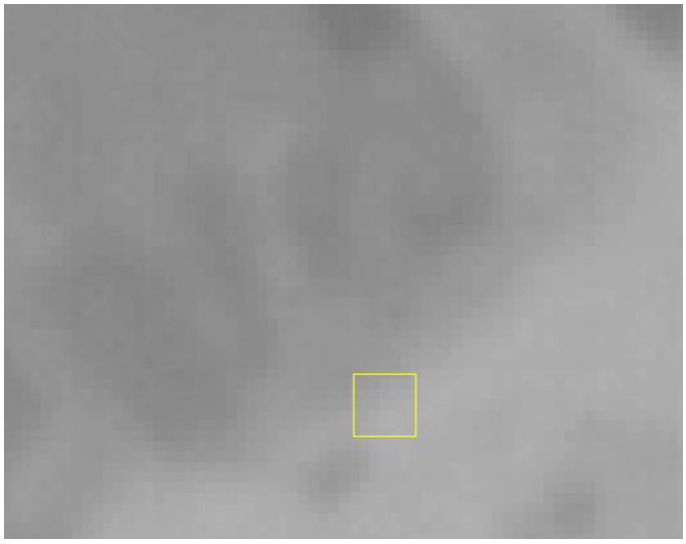
This presents a challenge for tracking algorithms. Many animals have three degrees of freedom, while the camera recording the behavior will only be a two dimensional projection of this. Although multiple cameras could used to create a stereo videographic dataset, however this is very complicated[9, 10].

To demonstrate the utility of object tracking on animals, a sample snippet of a flock of birds taking off was taken. The video was registered (via the method discussed in Section 3.2.3) so tracking could be performed, then inverted to make the image appear as though it was taken in the infrared. The AKF with MS algorithm is shown here to show the difficulties with these scenarios.

Figures 3.21(a)-3.21(c) show three frames from a single bird being tracked in a flock. Expected results should be pretty poor due to lots of confusion within the scene. All birds appear to be identical, and occlude each other, and have an extra dimension of freedom that is lost in the projection onto the camera. However, the tracking is not bad for a while, until a confusion turns the KF on, causing the algorithm to fail. The MS alone does not do any better, and simply jumps from bird to bird, not being able to distinguish them (not shown).



(a)



(b)

Figure 3.16: This data is courtesy of ITT Exelis Geospatial Systems and is from the Wide Area Persistence Surveillance (WAPS) centered at 4 μm. The yellow box is the MS target tracking result, and the area shown is the same area as shown by the yellow box in Figure 3.15. (a) The second frame from a 4 frame sequence, showing that MS was able to get from the initial location in the first frame and find the vehicle in the next frame. (b) Although MS was able to track the vehicle for one frame, the following frame looses the vehicle and converges to a spot on the side of the road. This is due to the similarity of the background and the target object, making distinction between them difficult.

Figure 3.17: 3.7 μm simulated image using DIRSIG. Simulation time set for late summer in the afternoon, causing the roofs of the buildings to be hot. Image courtesy of David Rhodes.

Figure 3.18: 9.0 μm simulated image using DIRSIG. Simulation time set for late summer in the afternoon, causing the roofs of the buildings to be hot. Image courtesy of David Rhodes.

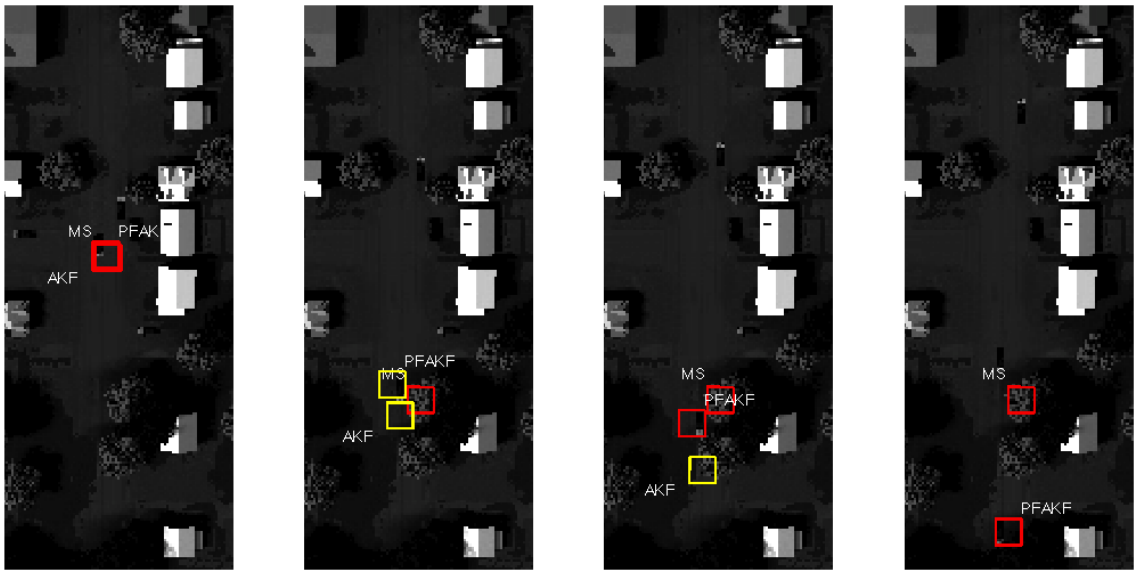


Figure 3.19: This is a synthetic scene generated by DIRSIG at 9.0 μm. The three tracking algorithms were tested on this data set, with unsurprising results. The MS algorithm loses the object when passing under the tree, while the AKF and PFAKF are able to continue searching for the object. However, since the object slows down after passing under the tree, predicting the location is made very difficult. Therefore the AKF is not able to continue tracking the object, while the PFAKF does slow down to reconvene with the target. The PFAKF is able to slow down due to the iterative approach used to calculate the projecting distance. This is discussed further in the text.

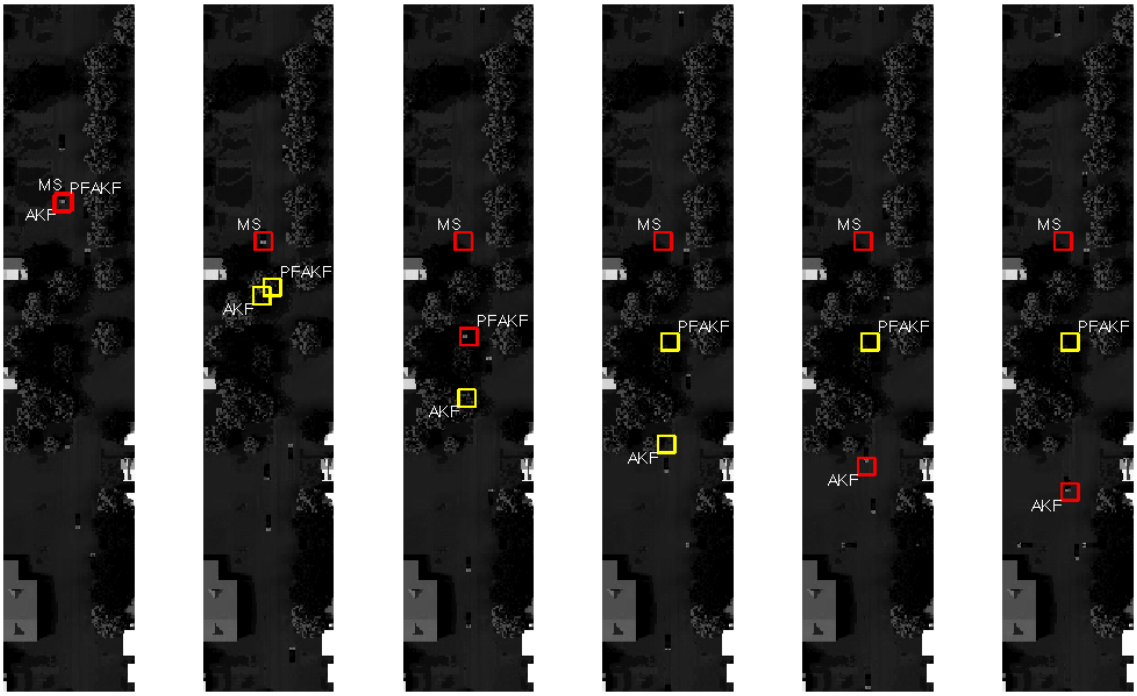


Figure 3.20: This is a synthetic scene generated by DIRSIG at 9.0 μm, with the three tracking algorithms results overlaid. Again, the MS algorithm loses the object upon occlusion, while the PFAKF and AKF continue with prediction. However, the PFAKF has problems when the object speeds up under the occluding tree, causing it to pick up the next vehicle due to identical vehicles in the scene. The short window in which the PFAKF is able to converge on the target confuses it into believing the target has stopped. The AKF keeps moving until the target speeds up again, in which it loses it. However, it does pick up the next vehicle and continues to track it until the sequence ends.

(a) (b) (c)

Figure 3.21: This image sequence was made by recording birds taking off, registering the images, then performing an AKF with MS tracking. Due to errors in registration and similarity of targets with lots of confusion and occlusions, expected results are poor. Red indicates the object is being tracked using the MS measurement, while the yellow box indicates the AKF predicted is being trusted. (a) Near the beginning, this bird does not get occluded, and allows tracking to be performed very simplify. (b) Note how as the bird being tracked gets further along, other birds start to get closer. What cannot be seen in these couple frames is that this is a different bird than the original one. (c) Another bird crossed paths with the previously tracked bird, causing the correlation to drop. This triggered the KF, causing the prediction to run away, loosing track of any bird.

Bibliography

[1] http://www.rochesterairshow.com/performers.php, 25 Jan. 2012.

[2] http://www.thermal-cameras.com/th-portable.htmlez885, 3 Apr. 2012.

[3] Reddy, B.S.; Chatterji, B.N.; "An FFT-based technique for translation, rotation, and scale-invariant image registration," *Image Processing, IEEE Transactions on*, vol.5, no.8, pp.1266-1271, Aug 1996.

[4] http://dirsig.org/, 29 May 2012.

[5] http://www.thermoanalytics.com/products/muses/index.html, 29 May 2012.

[6] http://sumo.sourceforge.net/, 29 May 2012.

[7] David B. Rhodes, Zoran Ninkov, Judith L. Pipher, Craig W. McMurtry, J. Daniel Newman, Paul P. K. Lee, Gregory J. Gosian and Michael D. Presnar, "Synthetic scene building for testing thermal signature tracking algorithms", Proc. SPIE 7813, 781309 (2010).

[8] David B. Rhodes, Zoran Ninkov, J. Daniel Newman, Paul P. K. Lee and Gregory J. Gosian, "Development of radiometrically accurate synthetic thermal infrared video for tracking algorithm evaluation", Proc. SPIE 8403, 840306 (2012).

[9] Viscido, Steven V. et al., "Individual behavior and emergent properties of fish schools: a comparison of observation and theory," *Marine Ecology Progress Series*, Vol 273, 10.3354/meps273239, pp. 239-249, 2004.

[10] Ballerini, Michele; Cabibbo, Nicola; Candelier, Raphael; Cavagna, Andrea; Cisbani, Evaristo; Giardina, Irene; Orlandi, Alberto; Parisi, Giorgio; Procaccini, Andrea; Viale, Massimiliano; Zdravkovic, Vladimir;, "Empirical investigation of starling flocks: a benchmark study in collective animal behaviour", *Animal Behaviour*, Volume 76, Issue 1, July 2008, Pages 201-215, ISSN 0003-3472, 10.1016/j.anbehav.2008.02.004.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

Target tracking of objects in the infrared presents various challenges. Occlusions, non-predictable motion, appearance changes, and crowded fields are just a few of these challenges encountered during this work. The mean shift algorithm has been shown to provide robust tracking in simple infrared scenarios. However, situations where occlusions do not occur are quite rare, especially in the field of persistence surveillance. Other difficulties for mean shift include that of nearby target confusion, drastic changes in appearance, and rapid object motion. To overcome these shortcoming, an adaptive Kalman filter was added as an object prediction filter. This would allow the mean shift algorithm to begin its search at where the object was predicted to be. If the correlation between the mean shift stopping point and the model varied too greatly, the prediction would be trusted. This combined tracking algorithm is able to deal with linear motion occlusions. This algorithm is better at dealing with rapid object motion and occlusions, but still has difficulty with nonlinear motion.

Non-linear motion is very common in some scenarios, such as cars driving along a road or airplane motion. The object tracking algorithm presented was improved by incorporating a polynomial fit to the framework of the adaptive Kalman filter and mean shift tracker. This algorithm allows the Kalman filter to predict motion and the mean shift algorithm to converge on the object of interest. However, the predicted path is comprised of a polynomial fit to the past known locations, along with a distance along this curve the object should have gone.

The presented algorithm is more robust than the mean shift algorithm alone or the adaptive Kalman filter with mean shift algorithm, but still has difficulties. A Kalman filter assumes Gaussian noise, which may be a reasonable assumption, it is not always correct. Additionally, it makes use of a linear prediction state model, which may work well for "normal" object motion, but sudden motion cannot be predicted for. Although other

algorithms that perform predictions exist without these assumptions, the Kalman filter is still a useful tool. It can converge on an object quickly, allowing for motion prediction not able to be performed as quickly with other algorithms.

Although this approach is an improvement upon the adaptive Kalman filter and mean shift tracking algorithms, there is still room for improvement for this tracking algorithm. A more robust measurement algorithm to replace the mean shift algorithm would help to eliminate many of the difficulties the current implementation has due to poor measurements (e.g. confusion with nearby objects, difficulties converging with object of interest after appearance change). An example algorithm is any of the contour evolution/matching algorithms discussed in Yilmaz et al. 2006[1]. These solutions do have there own set of difficulties, but could produce more robust tracking results. Another prediction method could also be incorporated and used if the polynomial fitting adaptive Kalman filter and mean shift do not converge to the object of interest. An area search algorithm such as one based on the FFT based registration algorithm to find the best match would be a novel method*. A particle filter is another algorithm used commonly in tracking that makes a location estimation based on weights of particles[1]. The particles are spread out around an area and the weighting can be viewed as a fitness function. This allows non-Kalman filter predicted motion to be found, potentially eliminating another difficulty with object tracking.

4.2 Future Work

To properly test and compare these and other tracking algorithms, a tracking algorithm test platform with tracking metrics must be developed. This will allow the rapid testing and quantification of tracking performance on various scenarios, eventually building up a database of algorithms and effective settings/results. From this, the development of new algorithms or choosing the correct algorithm for a purpose will be made easier. The general layout of this platform has been started with coding done in Python[3]. Python was chosen because it is open source, cross-platform, and does not require compiling. Additionally, it is capable of running C/C++ code (and various others) in the background, making it able to test a wide variety of tracking algorithms written in various languages.

The plan is to start out with a simple command line interface, using a configuration file to load a tracking algorithm, video scenario, and choose the various inputs necessary to initiate the tracking algorithm. The output could be a list of locations saved in a text file, or a video with the information overlaid for visualization purposes. Another useful output would be that of a metric so a quantitative comparison of algorithms could be performed. For this, the known object locations would be loaded to compare the tracking

*However, it would have to be fully extended to deal with the scale/rotational changes in the object of interest. This is already done[2], however, it has not been used for object tracking to the author's knowledge.

results to, using a couple of metrics presented by Presnar 2010[4]. The track purity is a measure of how "pure" the tracking results are. This is done by comparing the measured location of the object with the known location. This is able to score how well the objects are being tracked, with a 1 being perfect, and a 0 being not at all. The track completeness is a measure of how many objects were tracked compared to how many should have been tracked. This measure is used to make sure only objects of interest are tracked, and that no objects go untracked. Similarly to the track purity, a score of 1 is a perfect score, while 0 means no objects that should have been tracked were tracked. In addition to these types of inputs and outputs, a graphical user interface (GUI) would eventually be desired. A sample GUI layout and initial platform code can be found in Appendix C.

Bibliography

[1] Yilmaz, Alper; Javed, Omar; Shah, Mubarak; "Object Tracking: A Survey," *ACM Computing Surveys*, vol.38, no.4, pp.32-36, Dec 2006.

[2] Reddy, B.S.; Chatterji, B.N.; "An FFT-based technique for translation, rotation, and scale-invariant image registration," *Image Processing, IEEE Transactions on*, vol.5, no.8, pp.1266-1271, Aug 1996.

[3] http://www.python.org/.

[4] Mike Presnar, "Modeling and simulation of adaptive multimodal optical sensors for target tracking in the visible to near infrared", Rochester Institute of Technology, PhD in Imaging Science, (2010), https://ritdml.rit.edu/handle/1850/12779.

Appendix A

Figures for Section 2.1

The figures in this appendix go along with the discussion about target detection techniques in Section 2.1. Figure A.1 is the original image that the single frame difference and double frame differencing are then applied to, shown in Figures A.2 and A.3.

Figure A.2 shows how the regular frame difference technique causes noise to show through. From Figure A.1, the image appears to be noise free, and of reasonably high quality. The fact that this clear image shows difficulties that show the necessity of a more robust technique for target detection.

Figure A.3 shows drastic improvement over the regular frame differencing, by removing all noise via the threshold step. Another point to notice is the only part of the object visible are the edges. This is because the object is not moving fast enough for the whole object to pass through without being thresholded down to zero.

The next set of images is also on the discussion of target detection, however, they focus on the statistical differencing method. Figure A.4 is the original image, taken in the infrared using a microbolometer. The image was generated using filter number 1, seen in Figure A.4 as a watermark, and was taken overlooking an RIT parking lot. This data had a lot of noise, most likely due to not being a cooled infrared detector.

Of the statistical methods, mode (Figure A.5) and median (Figure A.6) work by far the best on this noisy data. It is able to subtract off the background along with the noise most reliably, leaving distinct clumps where the moving objects lie. Using the mean (Figure A.7) results in an image that leaves some false objects, likely to be trails of the moving objects.

Figure A.1: Sensiac image that the single frame difference and the double difference techniques are applied to. Note the lack of visible noise and the relatively constant grey-level along each section of the vehicle.

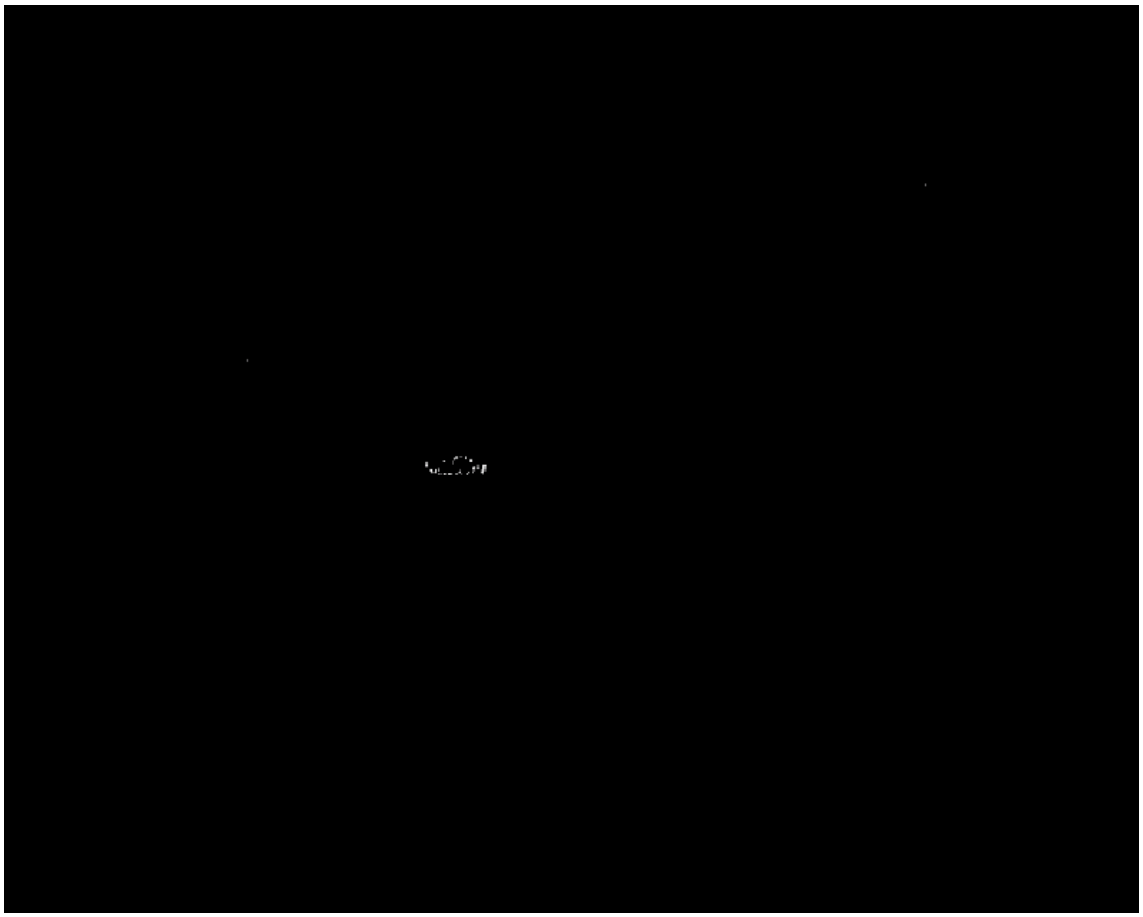


Figure A.2: Sensiac image after application of a single frame difference. Note the noise appearing though the differencing, giving potential difficulties with target detection.

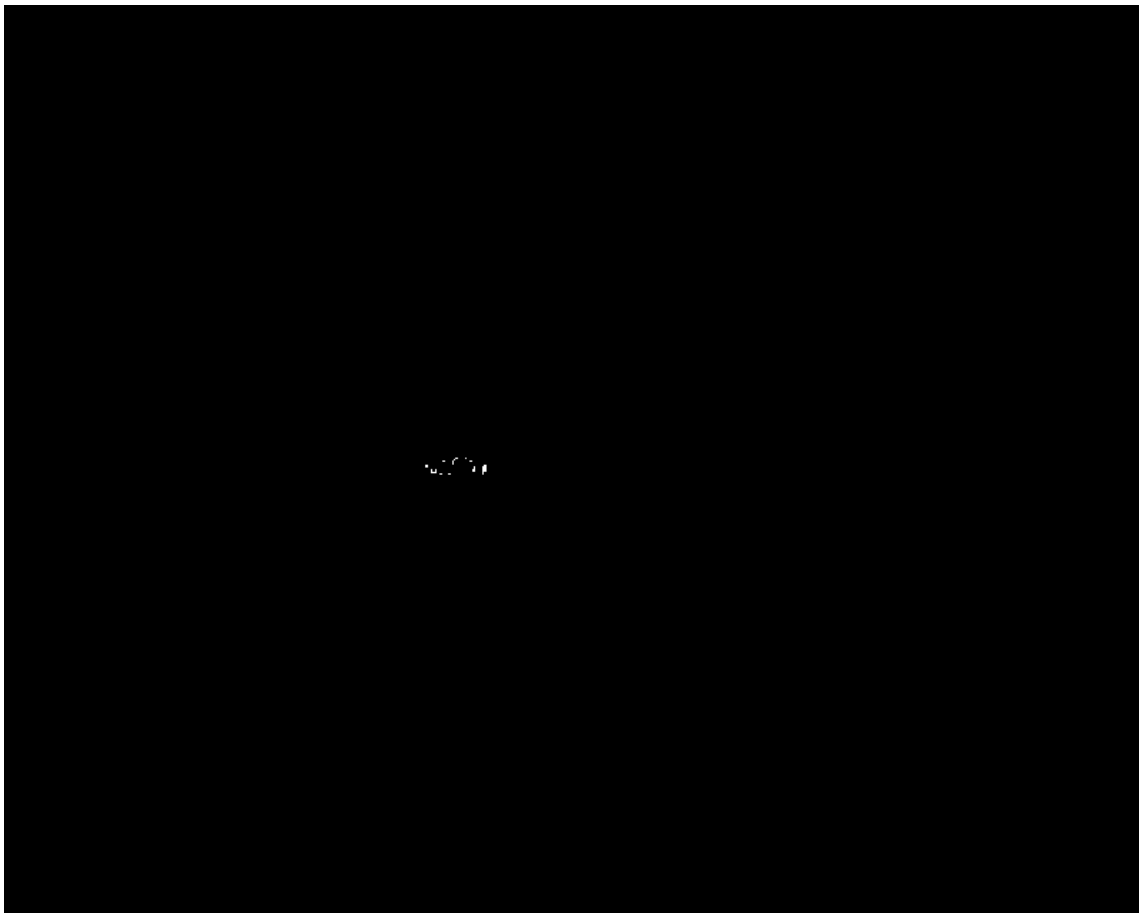


Figure A.3: Sensiac image after application of the double differencing technique. Notice how the noise seen in Figure A.2 is completely eliminated, showing just the outside edge of the vehicle.

Figure A.4: Noisy infrared image taken with a microbolometer. "1" in the image represents a watermark labeling which filter was being used. The only moving targets are the people, two to the left and one to the right.

Figure A.5: Image from Figure A.4 with a statistical mode difference method of target detection. Note how noise still gets though, but the people in the frame are still very distinct. 100 frames where used to calculated the mode.

Figure A.6: Image from Figure A.4 with a statistical median difference method of target detection. Note how noise still gets though, but the people in the frame are still very distinct. 100 frames where used to calculate the median. This technique has an output very similar to using the mode for this data, so the more computationally efficient method would be preferable.

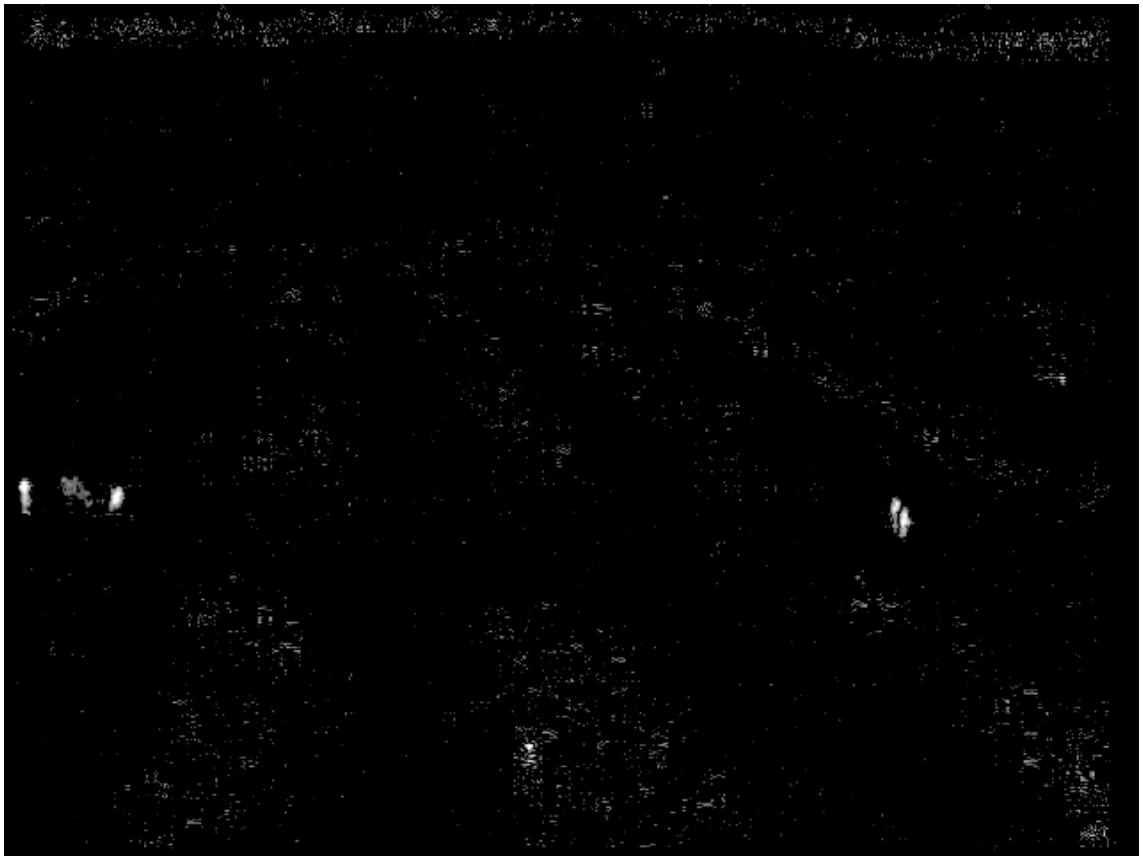


Figure A.7: Image from Figure A.4 with a statistical mean difference method of target detection. This method does not work well, letting more noise through than either the median or mode did, as well as highlighting a tree, which could be confused for a fourth person. 100 frames where used to calculate the mean.

Figure A.8: Image from Figure A.4 with a single frame difference method of target detection. Notice how much noise gets though, completely loosing any information on the people's locations. This technique is unusable in data with anything with a fair amount of noise.

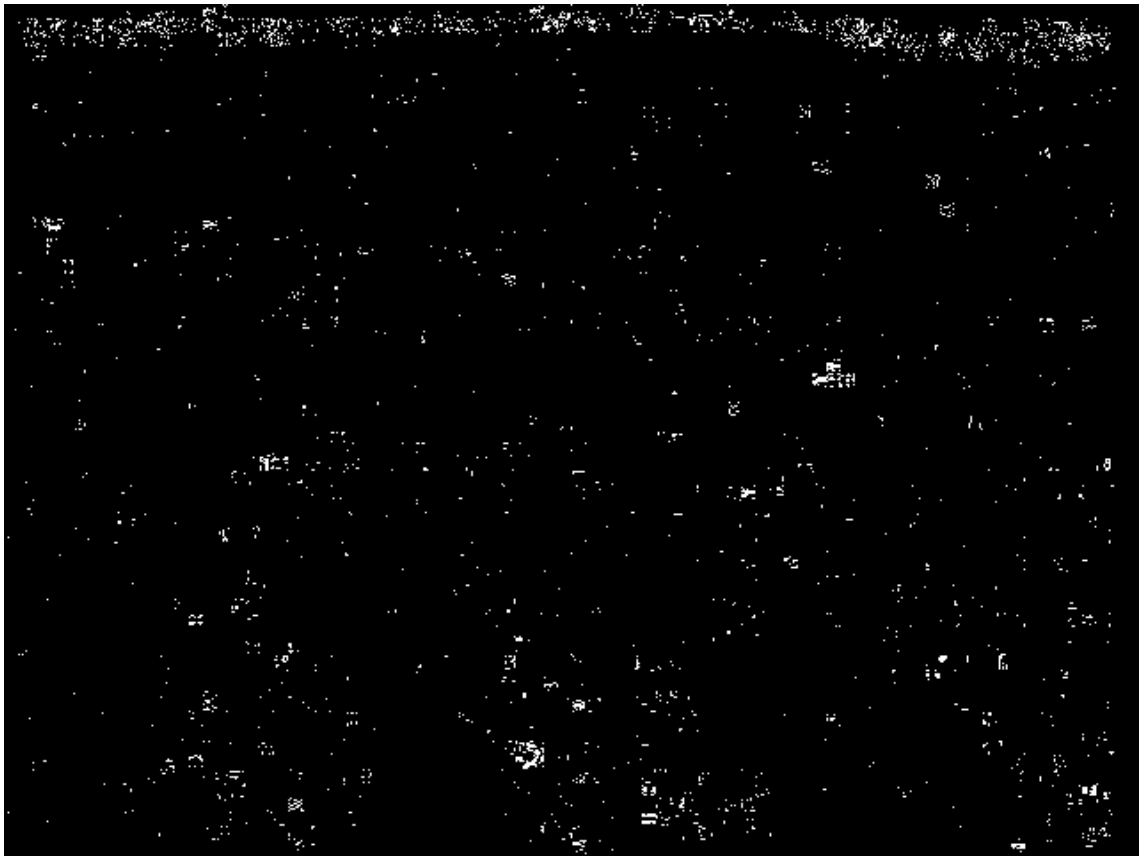


Figure A.9: Image from Figure A.4 with a double difference method of target detection. Much like the single frame difference in Figure A.8, this technique eliminates all trace of the people's locations, as well as letting noise through. Although there is less noise than the single frame difference, this technique is still unusable for noisy data.

Appendix B

MATLAB Implementation of Algorithms

This appendix contains a few lines of code that was used throughout this work. The point is to get the idea of how the algorithms were coded, and not to be pieced together blindly for use. Before any of these are initiated, a video file must be read into a readable format. The following is a sample of importing an infrared AVI video into a data array in MATLAB.

```
%   Import Video File:
vid=VideoReader('IR_10_08_2010_1.avi'); %sample video

%   Convert to Data Array:
frames=vid.NumberOfFrames;     % Number of frames in video
vidsize=[vid.Height,vid.Width];        % Obtains the size of each frame
imdata=zeros(vidsize(1),vidsize(2),frames);     % Creates an array the proper
    %size to store images in
for m=[1:1:frames]      % Changes each frame into an image, then stores
    temp1=read(vid,m);      % Turns frame into image
    temp1=temp1(:,:,1);     % Grabs only one of the RGB 'columns' only needed
        %for greyscale non indexed video
    imdata(:,:,m)=temp1;        % Puts the frames into the data array
end
high=max(max(max(imdata)));     % Finds the maximum value
data=imdata/high;       % Normalizes the data, so it is accurate
clear imdata
clear vid
```

B.1 Mean Shift

B.1.1 Distance Kernel

The creation of the distance kernel is straight forward, and can easily be modified to have any isotropic, monotonically decreasing, and convex kernel profile. For this, $d = 0.075$, but I have found it ranges from about 0.007 to around 0.09.

```
for i=1:1:size(oldtemp,2)%run over columns
    for j=1:1:size(oldtemp,1)%run over rows
        k_d(j,i)=(2/pi)*(1-sqrt(((x(i)-xc(count-1))^2+...
            (y(j)-yc(count-1))^2)/d^2);
        if k_d(j,i)<0
            k_d(j,i)=0; %if the kernel goes negative, make zero
        end
        bq(j,i)=find(imhist(oldtemp(j,i))==1);
    end
end
```

x runs over the columns, while y runs over the rows, and (xc, yc) gives the respective centers. The count we will see is used to keep track of how many times the MS iterative steps have been run, and rebuilds the kernel about the last center location found.

B.1.2 Histogram Kernel Selection

The creation of the intensity histogram kernel (k_h), as mentioned in Section 2.2, can be done in a few different ways. By manually selecting u_{peak}, accurate tracking can be performed very simply and accurately (assuming proper selection of h as well). However, if looking at infrared data, where the target is much brighter (or darker) than the background, the histogram will tend to appear bimodal. The secondary peak may be found, giving and automated way to find u_{peak}. Choosing a proper h still needs to be considered or automatically chosen. I did not automatically find h, but it wouldn't be too hard to find a way to do so if the histogram appears bimodal. The following is sample code for the automatic selection of u_{peak}

```
% Finds peaks near the higher bins of the model (oldtemp)
% Then finds the regional maxima near the end of the histogram
%    Note xq is the bin number, yq is the bin value
% Takes the first of these peaks, we do not want the first because
%    it could just be noise or the border of the image if white.

oldtemphist=imhist(oldtemp); %makes histogram of target model
```

```
[xq,yq]=find(imregionalmax(oldtemphist)==1,5,'last'); %finds last 5 maxima
upeak=xq(find(max(oldtemphist(xq))==oldtemphist(xq),1,'first'));
    %calls the first peak the histogram bin we want to center around
```

From this, we can simply calculated the histogram kernel, using $h = 10$ with the following.

```
q=zeros(size(k_d)); %make sure q is the same size as the distance kernel
for u=1:1:length(imhist(oldtemp)) %run though all histogram bins
    k_h(u)=(1-sqrt((u-upeak)^2/(10)^2));
        if k_h(u)<0
            k_h(u)=0; %makes sure it is either positive or zero
        end
    q=q+(k_d(:,:).*eq(b_q(:,:),u))*k_h(u); %calculation of q
end
q=q/max(max(q)); %want our probability normalized to [0,1]
```

The calculation of q is easy to perform within this same for loop, as shown. Note when calculating q, k_d is the distance kernel, and b_q is a helper function. It is simply the sub-image that contains the histogram bin number for each pixel. This allows the calculation of q to be made without use of for loops over each pixel. The MATLAB function "eq.m" is just a Kronecker delta, as defined in equation (2.6). What this is doing is summing up the probability each pixel belongs to the object of interest, and is weighted by $w(i, j, u) = k_d(i, j) \cdot k_h(u)$.

B.1.3 Iterative Step

To actually implement the MS algorithm, we just need to put the pieces together above together in a while loop.

```
%set starting threshold, and set current value higher than it
dchange=5;
count=0;
dthresh=1;
countmax=15; %generally between 15-20
while (dchange>dthresh)&&(countmax>count)
    count=count+1
    if count==1 %or if q needs to be recalculated
        %CALCULATE k_d for q

        %FIND upeak
```

```
        %CALCULATE k_h for q, then calculate q
    end
    %CALCULATE k_d for p

    %CALCULATE p, using k_h from above

    %CALCULATE CENTER LOCATIONS
    m00=[]; m01=[] m10=[];
    m00=sum(sum(p));
    m10=sum(sum(p*x.')); %x is a vector of pixel column values
    m01=sum(sum(y*p)); %y is similar to x but rows

    %CALCULATE NEW CENTERS
    xc(count)=m10/m00+xc(count-1); %creates the new center location
    yc(count)=m01/m00+yc(count-1);
    %note xc(count)=l_c and yc(count)=l_r

    %CALCULATE DISTANCE CHANGE
    dchage=sqrt((xc(count)-xc(count-1))^2+(yc(count)-yc(count-1))^2);
end
```

Although using moments is the easiest way to calculate the central locations, the other way is shown in case it is desired below.

```
for i=1:1:size(p,2)
    for j=1:1:size(p,1)
        %With Kernel and Meanshift Vector
        for u=1:1:256
            if p(j,i)==0
                %avoids divide by zero issues
                p(j,i)=0.001;
                flagp=1;
            elseif q(j,i)==0
                q(j,i)=0.001;
                flagq=1;
            end
            % if this is going to be used, can condense by
            % calculating w ahead
            deltx=deltx+(x(i)*k_h(u)*sqrt(q(j,i)/p(j,i))*eq(bp(j,i),u));
            delty=delty+(y(j)*k_h(u)*sqrt(q(j,i)/p(j,i))*eq(bp(j,i),u));
            deltund=deltund+(k_h(u)*sqrt(q(j,i)/p(j,i))*eq(bp(j,i),u));
```

```
            if flagp==1
                p(j,i)=0;
                flagp=0;
            elseif flagq==1;
                q(j,i)=0;
                flagq=0;
            end
        end
    end
end

xc(count)=deltx/deltund+xc(count-1);
yc(count)=delty/deltund+yc(count-1);
```

Just don't forget when using this method that the various temporary values must be reset to zero after each iteration. Also note that these snippits of code will not run accurately, as they were slightly modified/incomplete for the sake of clarity.

B.2 Kalman Filters

B.2.1 Kalman Filter Prediction Step

The first step in the KF is to calculate A, the state matrix. Recall that l_c is the stored column locations, while l_r is the stored row locations. $X_{x,o}$ is calculated the current position, velocity, and acceleration for the column locations. $X_{x,n}$ is the new projected column location, velocity, and acceleration.

```
averageover=5; %number of used locations to calculate X
% this implies you need at least 2*averageover+1 known locations
%   to run this filter. Usually you just use the measurement until
%   enough locations are measured, then you initiate this.
Xxo=[lc(size(lc,2)),(lc(size(lc,2))-...
    lc(size(lc,2)-averageover))/(averageover),...
    (((lc(size(lc,2))-lc(size(lc,2)-averageover))...
    /(averageover))-((lc(size(Multc,2)-averageover)-lc(size(lc,2)...
    -2*averageover))/(averageover)))/(2*averageover+1)];
Xyo=[lr(size(lr,2)),(lr(size(lr,2))-lr(size(lr,2)-...
    averageover))/(averageover),(((lr(size(lr,2))-lr(size(lr,2)-...
    averageover))/(averageover))-((lr(size(lr,2)-averageover)-...
    lr(size(lr,2)-2*averageover))/(averageover)))/(2*averageover+1)];
```

```
%Pmeas is the measurement error (estimation)
%Q is the state covariance
% and both must be chosen for a system/measurement technique

A=[1,1,0.5*1^2;0,1,1;0,0,1];     % State transition matrix
% NOTE: dt=1 frame for this work
%  A=[1,dt,0.5*dt^2;0,1,dt;0,0,1];     % State transition matrix

H=[1,0,0];      % Measurment matrix
Xxn=A*Xxo.';     % New state vector (predict)
Xyn=A*Xyo.';     % New state vector (predict)
Pest=A*Pmeas*A'+Q;        % Estimated error
```

B.2.2 Adaptive Kalman Filter Prediction Step

The only difference between the two is the calculation of the state covariance matrix must now be performed instead of just assuming it hasn't changed. This is done with the factors of σ and λ from Section 2.3.2. Note that CC below is the stored correlation coefficients, and n is within a for loop running through each frame, with n being the current one.

```
Pmeas=0.05;      % starting measurement error
if CC(n-1)>=CCThreshold
    sig1(n-1)=CC(n-1);
elseif CC(n-1)<CCThreshold
    sig1(n-1)=0;
end

if CC(n-2)>=CCThreshold
    sig1(n-2)=CC(n-2);
elseif CC(n-2)<CCThreshold
    sig1(n-2)=0;
end

sig1(n-1)=(1-lambda)*sig1(n-1)+lambda*sig1(n-2);
    % Temporal filtering for sigma

Q=diag([sig1(n-1),0.5*sig1(n-1),0.2*sig1(n-1)]);
    % State noise
```

B.2.3 Kalman Filter Comparison Step

The calculation of the Kalman factor (or gain) is fairly straightforward. The factor R must be estimated prior to calculation and is the measurement covariance.

```
Zx=lc(n); % New measurment vector (actually scalar)
Zy=lr(n);  % New measurment vector

Kgain=(Pest*H')'/(H*Pest*H'+R); % Kalman Gain
Pmeas=(diag([1,1,1])-Kgain*H')*Pest; % Measurement error
lc(n)=(Xxn+Kgain*(Zx-H*Xxn'))*H; % locations according to the KF
lc(n)=(Xyn+Kgain*(Zy-H*Xyn'))*H;
```

B.2.4 Adaptive Kalman Filter Comparison Step

The only addition here is the measurement covariance, R, is calculated at each frame, similar to Q above. This can be done as shown here.

```
if CC(n)>=CCThreshold
    sig2(n)=1-CC(n);
elseif CC(n)<CCThreshold
    sig2(n)=1000;
end

if CC(n-1)>=CCThreshold
    sig2(n-1)=1-BC(n-1);
elseif CC(n-1)<CCThreshold
    sig2(n-1)=1000;
end

sig2(n)=(1-lambda)*sig2(n)+lambda*sig2(n-1);

R=sig2(n)^2;     % Measurment noise
```

B.3 Polynomial Fitting Adaptive Kalman Filter

B.3.1 Polynomial Fit

The following is the implementation of the MATLAB "fit.m" function used in this work. The polynomial selected is of order two (quadratic). One thing that must be noted before implementing this is that if the object motion is generally top to bottom, the polynomial fit should use ($x = \text{columns}, y = \text{rows}$) to avoid an invalid fit (object would appear to not

be a function). However, if the motion is left to right, the order should be reversed (e.g., ($x = \text{rows}, y = \text{columns}$)).

```
% locmin is the number of positions used to calculate the parabola
xtemp=lc(n-lockmin+1:n-1);    %looks at last lockmin-1 known locations
ytemp=lr(n-lockmin+1:n-1);

historyfit=fit(xtemp',ytemp','poly2');%fits the history of locations
p=coeffvalues(historyfit);%a vector of polynomial coefficients
```

B.3.2 Calculate Distance Along Curve

To calculate the distance along the curve, we have to integrate the derivative of the fit function. Here, the position along the curve is increased by 0.1 pixels in the column direction. The numerical integration is then calculated and a distance is compared. As soon as the while loop is exited, we know how many steps it took to project the same distance (within a threshold, set here to be just under half a pixel). The new predicted column position is then updated, followed by a calculation using the fit curve to find the proper row predicted location.

```
%function needed to find arc length
lengthfun = @(x) sqrt(1+(2*(p(1))*x+p(2)).^2);

%change with transition step
distfit=sqrt((Xxo(1)-Xxn(1))^2+(Xyo(1)-Xyn(1))^2);
distactual=0;
stepcount=0;
while abs(distfit(m)-distactual(m))>0.45%this is a "threshold" of accuracy
    stepcount=stepcount+1;
    %increase range until arc length is same as linear length
    %calculate distance using a numerical integration
    distactual=quad(lengthfun,xtemp(length(xtemp)),...
        xtemp(length(xtemp))+stepcount*0.1);
end
%how far in column direction does the prediction require
Xxn(1)=xtemp(length(xtemp))+stepcount*0.1;
%evaluate the function to find the row position at above column
Xyn(1)=feval(historyfit,Xxn(1));
```

B.4 NITF Image Extraction

The code in this section will be for use with the built-in MATLAB functions for working with NITF formatted images. The title and such are pulled for the header and writes a new image wit the same title, but it is in the JPEG2000 format (shown in B.4.1). Additionally, the code for registering (B.4.2) will be provided. If a video is already created and you wish to perform the registration, see Appendix B.5.

B.4.1 NITF Extraction to AVI Video

```
clear all, close all, clc

%Kyle Ausfeld - 12 December 2011
%Opens NITF files as long as they only have one image
% and are greyscale(?) and have a name longer than 3 characters


%This will open all the files in the directory
names_files=dir;

%creates video
vidObj = VideoWriter('test.avi');
vidObj.FrameRate=2;
open(vidObj);

%This will run through them and save them as jp2 with the same file name
for i=1:size(names_files,1)%ignores last file, should be this one
    names=names_files(i).name;
    if (length(names)>3)&&min(names(length(names)-3:length(names))=='.ntf')
        im_info=nitfinfo(names);
        fids=fopen(names);
        ftemp1=fread(fids);
        names(length(names)-3:length(names))=[];%removes .ntf
        names=[names,'.jp2'];%appens with .jp2 instead
        ftemp2=fopen(names,'w');
        fwrite(ftemp2,ftemp1(im_info.NITFFileHeaderLength+...
            im_info.ImageAndImageSubheaderLengths...
            .Image001ImageAndSubheaderLengths.LengthOfNthImageSubheader...
            +1:length(ftemp1)));
        fclose(ftemp2);
        im_temp=imread(names);
```

```
        figure, imshow(im_temp,[])
        currFrame = getframe;
        writeVideo(vidObj,currFrame);
        close figure 1;
    end
end
close(vidObj);
```

B.4.2 NITF to Registration to Video

```
clear all, close all, clc

%Kyle Ausfeld - 12 December 2011
%Opens NITF files as long as they only have one image, are square,
% and are greyscale(?) with image registration.


%This will open all the files in the directory
names_files=dir;
ftemp1=[]; ftemp2=[];
size_change=2;

%creates video
vidObj = VideoWriter('test_fft.avi');
vidObj.FrameRate=2;
open(vidObj);

hold_names=[];
hold_loc=[];
hold_change=[];
%    This will run through them and save them as jp2 with the same file name
for i=1:size(names_files,1)%ignores last file, should be this one
    names=names_files(i).name;
    if (length(names)>3)&&min(names(length(names)-3:length(names))=='.ntf')
        im_info=nitfinfo(names);
        % Pull out information for registrations
        locations_raw=im_info.ImageSubheaderMetadata.ImageSubheader001...
         .ImageGeographicLocation;
        locations=[];
        counto=0; counte=0;
        for j=1:8
```

```
    if rem(j,2)==1    %odd
        counto=counto+1;
        locations(counto,1)=str2num(locations_raw(((counto-1)*7+...
         (counte)*8+1):((counto)*7+(counte)*8)));
    else    %even
        counte=counte+1;
        locations(counto,2)=str2num(locations_raw(((counto)*7+...
         (counte-1)*8+1):((counto)*7+(counte)*8)));
    end
end
hold_loc(:,:,i)=locations;
%create image
fids=fopen(names);
shift_trigger=isempty(ftemp1);
ftemp1=fread(fids);
names(length(names)-3:length(names))=[];%removes .ntf
names=[names,'.jp2'];%appens with .jp2 instead
hold_names=[hold_names;names];
ftemp2=fopen(names,'w');
fwrite(ftemp2,ftemp1(im_info.NITFFileHeaderLength+im_info...
 .ImageAndImageSubheaderLengths.Image001ImageAndSubheaderLengths...
 .LengthOfNthImageSubheader+1:length(ftemp1)));
fclose(ftemp2);
if shift_trigger==1
    im_temp=imread(names);
    im_temp2=zeros(size(im_temp)*size_change);
    im_temp2(floor(size(im_temp,1)*((size_change-1)/2))+1:...
     floor(size(im_temp2,1)-size(im_temp,1)*((size_change-1)/2))...
        ,floor(size(im_temp,2)*((size_change-1)/2))+1:floor(size(...
        im_temp2,2)-size(im_temp,2)*((size_change-1)/2)))=im_temp;
    figure, imshow(im_temp2,[min(min(im_temp)) max(max(im_temp))])
else
    im_old=double(imread(hold_names(size(hold_names,1)-1,:)));
    im_old=edge(im_old/max(max(im_old)),'canny');
    im_temp=double(imread(names));
    im_temp=im_temp/max(max(im_temp));
    f2=edge(im_temp,'canny');
    F1=fft2(im_old);
    F2=fft2(f2);
    RF=(F1.*conj(F2))./(abs(F1).*abs(F2));
```

```
            displacement=abs(ifft2(RF));
            [v,x]=max(max(displacement'));
            [v,y]=max(max(displacement));
            if x>size(f2,2)/2
                x=x-1-size(f2,1);
            else
                x=x-1;
            end
            if y>size(f2,2)/2
                y=y-1-size(f2,2);
            else
                y=y-1;
            end
            hold_change(i,1)=x;
            hold_change(i,2)=y;
            im_temp2=zeros(size(im_temp)*size_change);
            im_temp2((floor(size(im_temp,1)*((size_change-1)/2))+1:...
             floor(size(im_temp2,1)-size(im_temp,1)*((size_change-1)/2)))+...
             sum(hold_change(:,1)),(floor(size(im_temp,2)*...
             ((size_change-1)/2))+1:floor(size(im_temp2,2)-...
             size(im_temp,2)*((size_change-1)/2)))+sum(hold_change(:,2)))...
             =im_temp;
            figure, imshow(im_temp2,[min(min(im_temp)) max(max(im_temp))])
        end
        currFrame = getframe;
        writeVideo(vidObj,currFrame);
        close figure 1;
    end
end
close(vidObj);
```

B.5 FFT Image Registration

The following code simply creates a zero-padded image and centers the first frame. From there, the following images are then registered to the previous frame, creating a simulated registration that only accounts for translational changes.

```
close all, clear all, clc;

%Takes video sequence, and turns into a new video that is
```

```
% registered using an fft technique
%   Kyle Ausfeld

%%   Import Video File:

vid=VideoReader('IMG_0087_1.mov');


%%   Convert to Data Array:

frames=vid.NumberOfFrames;     % Number of frames in video
vidsize=[vid.Height,vid.Width];        % Obtains the size of each frame
imdata=zeros(vidsize(1),vidsize(2),frames);     % Creates an array
the proper size to store images in
for m=[1:1:frames]      % Changes each frame into an image, then stores
    temp1=read(vid,m);      % Turns frame into image
    temp1=temp1(:,:,1);     % Grabs only one of the RGB 'columns' only
     needed for greyscale non indexed video
    imdata(:,:,m)=temp1;        % Puts the frames into the data array
end
high=max(max(max(imdata)));     % Finds the maximum value
data=imdata/high;       % Normalizes the data, so it is accurate
clear imdata


%%  Register

vidObj = VideoWriter('video_registered.avi');%,'Uncompressed AVI');
vidObj.FrameRate=vid.FrameRate;
vidobj.Quality=100;
clear vid
open(vidObj);
size_change=[1.5,2];

for n=1:size(data,3)
    if n==1
        temp1=zeros(size(data,1)*size_change(1),size(data,2)*size_change(2));
        temp1(floor(size(data(:,:,n),1)*((size_change(1)-1)/2))+1:...
         floor(size(temp1,1)-size(data(:,:,n),1)*((size_change(1)-1)/2))...
            ,floor(size(data(:,:,n),2)*((size_change(2)-1)/2))+1:floor(size(...
```

```
            temp1,2)-size(data(:,:,n),2)*((size_change(2)-1)/2)))...
            =data(:,:,n);
        figure, imshow(temp1,[])
        currFrame = temp1;
        writeVideo(vidObj,currFrame);
        close figure 1;
    else
        temp2=zeros(size(data,1)*size_change(1),size(data,2)*size_change(2));
        temp2(floor(size(data(:,:,n),1)*((size_change(1)-1)/2))+1:...
         floor(size(temp1,1)-size(data(:,:,n),1)*((size_change(1)-1)/2))...
            ,floor(size(data(:,:,n),2)*((size_change(2)-1)/2))+1:...
            floor(size(temp1,2)-size(data(:,:,n),2)*((size_change(2)-1)/2)))...
            =data(:,:,n);
        % FFT Register

        f1=edge(temp1,'canny');
        f2=edge(temp2,'canny');
        F1=fft2(f1);
        F2=fft2(f2);
        RF=(F1.*conj(F2))./(abs(F1).*abs(F2));
        displacement=abs(ifft2(RF));
        [v,x]=max(max(displacement'));
        [v,y]=max(max(displacement));
        if x>size(f2,1)/2
            x=x-1-size(f2,1);
        else
            x=x-1;
        end
        if y>size(f2,2)/2
            y=y-1-size(f2,2);
        else
            y=y-1;
        end
        hold_change(n,1)=x;
        hold_change(n,2)=y;
        temp3=zeros(size(data,1)*size_change(1),size(data,2)*size_change(2));
        temp3((floor(size(data,1)*((size_change(1)-1)/2))+1:...
         floor(size(temp2,1)-size(data,1)*((size_change(1)-1)/2)))+...
         sum(hold_change(:,1)),floor(size(data,2)*((size_change(2)-1)/2))+...
         1:floor(size(temp2,2)-size(data,2)*((size_change(2)-1)/2)))+...
```

```
         sum(hold_change(:,2)))=data(:,:,n);
        temp1=temp2;

        figure, imshow(temp3,[])
        currFrame = temp3;%getframe;
        writeVideo(vidObj,currFrame);
        close figure 1;
    end
end
close(vidObj);
```

Appendix C

Tracking Algorithm Testing Platform

put two figures in here along with metrics and code.

C.1 Tracking Algorithm Metrics

As mentioned in Chapter 4, the metrics proposed for future work will be the ones used by Mike Presnar. A description of them are given here, along with an equation for each.

(C.1)

The track purity is a measure of how "pure" the tracking results are. This is done by comparing the measured location of the object with the known location. This is able to score how well the objects are being tracked, with a 1 being perfect, and a 0 being not at all.

(C.2)

The track completeness is a measure of how many objects were tracked compared to how many should have been tracked. This measure is used to make sure only objects of interest are tracked, and that no objects go untracked. Similarly to the track purity, a score of 1 is a perfect score, while 0 means no objects that should have been tracked were tracked.

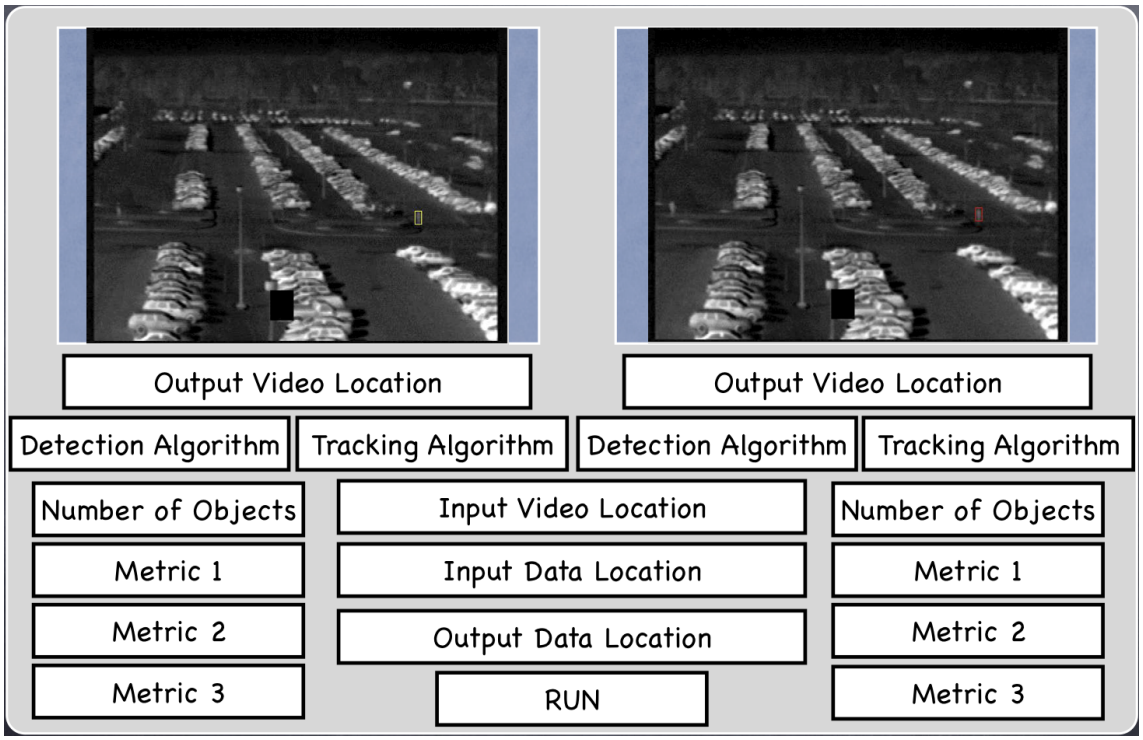


Figure C.1: This could be the layout if the user wanted to compare two different tracking algorithms on the same data set. This would be useful if the user was interested in one type of scene and wanted to choose the most robust algorithm for that scenario.

C.2 Testing Platform GUI Layout Possibilities

Two different sample layouts were made, leaving possibilities for use of other metrics/different shown outputs of interest. These were not coded up, but were made for a pictorial representation of potential layouts. The first layout (Figure C.1) shows a scenario where a comparative analysis of two algorithms on the same data set were of interest. The second layout (Figure C.2) shows a scenario where the same algorithm is tested on two different scenarios to test which scene the algorithm performs more robustly on.

C.3 Python Code

This section will be broken up into the different parts of the current code that has been written. Comments will be added where deemed necessary.

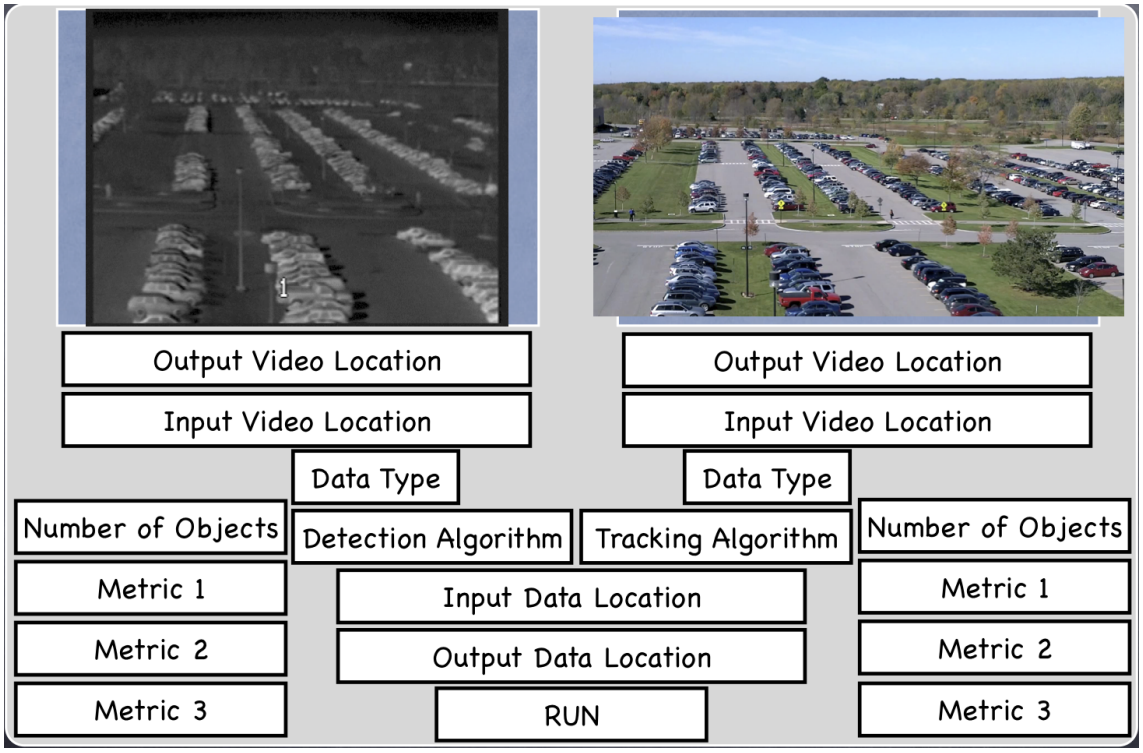


Figure C.2: This could be the layout if the user wanted to compare the tracking robustness of a tracking algorithm on two different data sets. This would be of interest to users who have to have robust tracking in a variety of scenarios, making the testing on various scenarios beneficial.

C.3.1 Configuration File

The following is the proposed configuration layout, to be added onto as necessary. These are not complete, but a work in progress.

```
CONFIGURATION FILE OPTIONS:

TRACK TYPE:
SINGLE VIDEO-single movie track
MULTIBAND VIDEO-comparison between videos, say one vis one ir
MULTIPLE VIDEO-track multiple videos independently

BAND TYPE:
INFRARED
VISIBLE
2 COLOR INFRARED
POLARIZED

DETECTION ALGORITHM:
GIVEN INFORMATION
MODE DIFFERENCING

TRACKING ALGORITHM:
MEAN SHIFT
MEAN SHIFT AKF

OBJECT SIZE:
(ROWS,COLUMNS)
? ([ROWS,COLUMNS],[ROWS,COLUMNS],)-for multiple objects
NONE-in the case of using an actual detection algorithm

OBJECT LOCATION:
(ROWS,COLUMNS)
NONE-in the case of using an actual detection algorithm

DETECTION INFORMATION:
For GIVEN INFORMATION: ()-isn't read and will be ignored, best left blank
For MODE DIFFERENCING: (time,other stuff not yet defined)

TRACKING INFORMATION:
For MEAN SHIFT: (h,max_iterate,enlargewindow,dchangethresh)
```

```
h=0.075-Mean Shift bandwidth
     max_iterate=20-Maximum number of Mean Shift iterations per object
     per frame
     enlargewindow=0-How much the tracking window is expanded by, use
     for fast moving objects
     dchangethresh=1-How far the centerpoint can change by in Mean Shift
For MEAN SHIFT AKF: (h,lockmin,max_iterate,enlargewindow,dchangethresh,
lambda,BCThresh,Tkf,Pmeasure)
h=0.075-Mean Shift bandwidth
     lockmin=10-Delay to start AKF, use just MS until locked
     max_iterate=20-Maximum number of Mean Shift iterations per object
     per frame
     enlargewind=0-How much the tracking window is expanded by, use for
     fast moving objects
     dchangethresh=1-How far the centerpoint can change by in Mean Shift
     lambda=0.9-Kalman Filter forgetting factor (lower forgets faster)
     BCThresh=0.5-Threshold for the Bhattacharyya coefficient used during
     the AKF/CAMSHIFT
     Tkf=1000-Large value for use during Kalman Filter
     Pmeasure=0.05-Starting measurement error
```

An eventual goal would be a program to produce these configuration files for certain standard tests so they would not have to be written by hand for every algorithm. A sample configuration file could look like the following.

```
OBJECT TRACKING CONFIGURATION FILE

TRACK TYPE: SINGLE VIDEO

BAND TYPE: INFRARED

DETECTION ALGORITHM: GIVEN INFORMATION

TRACKING ALGORITHM: MEAN SHIFT

OBJECT SIZE: (20,10)

OBJECT LOCATION: (283,505)

DETECTION INFORMATION: ()

TRACKING INFORMATION: (0.075,20,0,1)
```

C.3.2 Python Read in Configuration File

The Python code currently has two important parts, the main file and the configuration read in file. Both will have to be expanded with more algorithms and options. The main python file is currently the following:

```
#! usr/bin/python 2.7
"""

main.py
        This is the main script used to run all the modules needed to
            run the Tracking Algorithm Testing Environment.
        The following scripts are necessary:
            - readconfig.py: reads the configuration file to load all
                the necessary inputs.
            - sys: used for stopping at errors
            - numpy: used for array manipulation and representation

    Kyle Ausfeld - 7/5/11 - Rochester Institute of Technology

"""


#STARTUP

print __doc__
raw_input('Please press [enter] or [return] to continue: ')
print " "


#IMPORT LIBRARIES

import sys
import numpy as np
import readconfig


#GET CONFIGURATION FILE INFORMATION

trackinginputs=readconfig.getinputs()
print " "
```

```
#ALLOWED VALUES FOR CONFIGURATION INPUTS

config0=["SINGLE VIDEO", "MULTIBAND VIDEO", "MULTIPLE VIDEO"]
config1=["INFRARED","VISIBLE","2 COLOR INFRARED","POLARIZED"]
config2=["GIVEN INFORMATION","MODE DIFFERENCING"]
config3=["MEAN SHIFT","MEAN SHIFT AKF"]

config=[config0,config1,config2,config3]
objectsizelocation=4
objectlocationinfo=5
detectioninfoloc=6
trackerinfoloc=7


#CHECK INPUT VALUES AND RETURN ERRORS

readconfig.checkconfigin(config, trackinginputs)

trackinginputs,objectsize,objectloc,detectioninfo,trackerinfo=readconfig.
getnumericalinfo(config, trackinginputs, objectsizelocation,
objectlocationinfo, detectioninfoloc, trackerinfoloc)


#GET VIDEO



#FIRST TIME BELOW IS CORRECT, WORK INTO FRAME-BY-FRAME WORKINGS

#   DETECTION ALGORITHM

#if (trackinginputs[2]==config[2][0]) or (detectioninfo==[]):
    #If no detection information exists, use given information
    #Make into an array of objects and locations in one array,
    #object_track[{object#}][{frame#}][0=>row,1=>column]
#elif (trackinginputs[2]==config[2][1]):
    #Makes information from detectioninfo usable and runs
    #desired algorithm.
```

```
    #print 'Not yet configured, Check back at a later date.'
    #sys.exit("Change DETECTION ALGORITHM in configuration file.")


#   TRACKING ALGORITHM


#END
```

The readconfig.py is shown below.

```
#! usr/bin/python 2.7
"""

readconfig.py
    Reads in a configuration file to start running a tracking script.
        The various functions listed below are used to do different
        parts of the task. For individual function help, please read
        below.

    Kyle Ausfeld - 7/13/11 - Rochester Institute of Technology

"""

def getinputs():
    """This function imports the configuration file and
            reads it out into an array.


        NO INPUTS


        Outputs are:

        trackinginput - Takes configuration file information
            and creates a readable list for later use.
    """

    #IMPORT LIBRARIES

    import numpy as np
```

```
    #LOAD FILE

    configlocation=raw_input('Type the location for the configuration file: ')
    configfile=open(configlocation.strip())
    ##configuration=open('/Users/kyleausfeld/Desktop/configlayout.txt')


    #SORT OUT INPUT VALUES INTO NEW ARRAY
    configure=np.array(configfile.readlines())
    configure=configure[np.arange(0,len(configure),2)]
    trackinginput=[]
    for n in range(1,len(configure)):
        for m in range(0,len(configure[n])):
            if str(configure[n])[m]==":":
                mm=m+1,
                break
        trackinginput.append(configure[n][mm[0]:].strip())
    #print mm, trackinginput
    return trackinginput
#END getinputs()


def checkconfigin(config, trackinginputs):
    """Check input values and return errors.


        Inputs are:

        config - The configuration layout array (should be defined in main.py).
        trackinginputs - Array from the function getinputs().


        NO OUTPUTS.
        """

    #IMPORT LIBRARIES

    import sys
```

```
    #CHECK CONFIGURATION FOR ERRORS

    for n in range(0,len(config)):
        if (trackinginputs[n] not in config[n]):
            print "Error in configuration line: ", (2*n+3)
            print "Allowed inputs are as follows: "
            print "     ", config[n]
            sys.exit("Check configuration file.")
#END checkconfigin()


def getnumericalinfo(config, trackinginputs, objectsizelocation,
objectlocationinfo,detectioninfoloc,trackerinfoloc):
    """Checks for errors in needed numerical information,
        then returns arrays with the proper information.


        Inputs are:

        config - The configuration layout array (should be defined in main.py).
        trackinginputs - Array from the function getinputs().
        objectsizelocation - Gives location (within config) of object size,
            current default is 4.
        objectlocationinfo - Gives location (within config) of object initial
            location, current default is 5.
        detectioninfoloc - Gives location (within config) of detection algorithm
            inputs, current default is 6.
        trackerinfoloc - Gives location (within config) of tracking algorithm
            inputs, current default is 7.


        Outputs are:

        trackinginputs - Array from the function getinputs(), but updated.
        objectsize - Array of objects and their sizes. Each row is a new
            object, 0th column is # of rows spanned by object, 1st column
            is # of columns spanned by object.
        objectloc - Array of objects and their initial location. Each row
```

```
        is a new object, 0th column is initial row position, 1st column
        is initial column position of object.
    detectioninfo - Array (vector) of information necessary to perform
        detection of objects of interest as defined in the configuration
        layout file.
    trackerinfo - Array (vector) of information necessary to perform the
        tracking of objects as defined in the configuration layout file.


    NOTE: This function will need to be updated with
        time. If a new vector of information is needed,
        such as if a new tracking algorithm is added,
        and needs a set of default inputs, this must
        be expanded.
"""

#IMPORT LIBRARIES

import sys
import numpy as np


#SECTIONED OUT BELOW:

for n in range(len(config),len(trackinginputs)):
    if trackinginputs[2]==config[2][0]:
        #GET INFORMATION FOR GIVEN BEGINING LOCATION
        if n==objectsizelocation:
            #GET OBJECT SIZE
            if trackinginputs[n]=="NONE":
                print "Error in configuration line: ", (2*n+3)
                print "Object size information needed."
                sys.exit("Check configuration file.")
            elif trackinginputs[n].count(',') %2==0:
                print "Error in configuration line: ", (2*n+3)
                print "Missing/excess object size information given."
                sys.exit("Check configuration file.")
            elif trackinginputs[n].count(',') %2==1:
                trackinginputs[n]=trackinginputs[n][1:len(trackinginputs[n])-1]
                objnum=(trackinginputs[n].count(',')+1)/2
```

```
            comloc=[-1]
            temp1=0
            while temp1 < len(trackinginputs[n]):
                temp1=trackinginputs[n].find(',',temp1)
                if temp1==-1:
                    comloc.extend([len(trackinginputs[n])])
                    break
                comloc.extend([temp1])
                temp1=temp1+1
            objectsize=np.zeros([objnum,2])
            for m in range(0,2*objnum):
                if m %2==0:
                    #ROW SIZE
                    objectsize[m/2][0]=trackinginputs[n][comloc[m]
                     +1:comloc[m+1]]
                elif m %2==1:
                    #COLUMN SIZE
                    objectsize[m/2][1]=trackinginputs[n][comloc[m]
                     +1:comloc[m+1]]
        elif n==objectlocationinfo:
            #GET OBJECT LOCATION
            if trackinginputs[n]=="NONE":
                print "Error in configuration line: ", (2*n+3)
                print "Object location information needed."
                sys.exit("Check configuration file.")
            elif trackinginputs[n].count(',') %2==0:
                print "Error in configuration line: ", (2*n+3)
                print "Missing/excess object location information given."
                sys.exit("Check configuration file.")
            elif trackinginputs[n].count(',') %2==1:
                trackinginputs[n]=trackinginputs[n][1:
                 len(trackinginputs[n])-1]
                objnum=(trackinginputs[n].count(',')+1)/2
                comloc=[-1]
                temp1=0
                while temp1 < len(trackinginputs[n]):
                    temp1=trackinginputs[n].find(',',temp1)
                    if temp1==-1:
                        comloc.extend([len(trackinginputs[n])])
                        break
```

```
                    comloc.extend([temp1])
                    temp1=temp1+1
                objectloc=np.zeros([objnum,2])
                for m in range(0,2*objnum):
                    if m %2==0:
                        #ROW SIZE
                        objectloc[m/2][0]=trackinginputs[n][comloc[m]
                         +1:comloc[m+1]]
                    elif m %2==1:
                        #COLUMN SIZE
                        objectloc[m/2][1]=trackinginputs[n][comloc[m]
                         +1:comloc[m+1]]
            elif n==detectioninfoloc:
                #GET DETECTION INFO
                #not used if no detection algorithm is chosen
                detectioninfo=[]
            elif n==trackerinfoloc:
                #GET TRACKER INFO
                if trackinginputs[n].count(',')!=3:
                    print "Error in configuration line: ", (2*n+3)
                    print "Missing/excess commas, possible sytax error."
                    sys.exit("Check configuration file.")
                elif trackinginputs[n].count(',')==3:
                    trackinginputs[n]=trackinginputs[n][1:
                     len(trackinginputs[n])-1]
                    objnum=4
                    comloc=[-1]
                    temp1=0
                    while temp1 < len(trackinginputs[n]):
                        temp1=trackinginputs[n].find(',',temp1)
                        if temp1==-1:
                            comloc.extend([len(trackinginputs[n])])
                            break
                        comloc.extend([temp1])
                        temp1=temp1+1
                    trackerinfo=np.zeros([objnum])
                    for m in range(0,objnum):
                        trackerinfo[m]=trackinginputs[n][comloc[m]
                         +1:comloc[m+1]]
        else:
```

```
            #Will hold other types of numerical information
             necessary not from given information
            print "Not yet configured."
            sys.exit("Check for ability in next release")
    return trackinginputs,objectsize,objectloc,detectioninfo,trackerinfo
#END getnumericalinfo()


#END readconfig.py
```