12-1-2010

Signal Conditioning and Feature Estimation for Profiling Sensor Systems

Jeremy Benjamin Brown

SIGNAL CONDITIONING AND FEATURE ESTIMATION FOR PROFILING SENSOR SYSTEMS

by

Jeremy B. Brown

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Electrical and Computer Engineering

The University of Memphis

December 2010

Abstract

Brown, Jeremy Benjamin. M.S. The University of Memphis. December 2010. Signal Conditioning and Feature Estimation for Profiling Sensor Systems. Major Professor: Eddie L. Jacobs, D. Sc.

This paper presents techniques for signal conditioning and feature estimation to be used for profiling sensors which perform broad scale classifications. Methods for improving existing feature estimation techniques for linear array profiling systems are provided. A 360 profiling system and the accompanying algorithms used for signal conditioning and feature estimation are presented. The algorithms are validated by demonstrating classification results for the broad scale human, animal, and vehicle three class problem for the linear array, and the human and animal two class problem for the 360 profiling sensor.

Table of Contents

List of Figures

1. Introduction

1.1 Profiling Sensors

In many security applications, it is necessary to be able to distinguish human, animal, and vehicles. These security applications are typically oriented toward, but certainly not limited to, military and government. There is a need for solutions to persistent and covert surveillance. One particular security application is border security. It can be difficult for a country to man the entire range of its border. The US-Mexico border is an example, where large amounts of illegal drugs are smuggled across this borders. With limited manpower, an automated system is needed to be able to give information on potential crossings. The concept of the profiling sensor was introduced by Ronnie Sartain [1] to provide a solution to this problem. The profiling sensor was defined as a sensor system which collected enough information to classify targets into the three main categories of interest: human, animal, and vehicle. Classifications would be performed based on a target's shape. For practical field use, the profiling sensor solution must be small, low power, low bandwidth, and low cost.

To be considered for use in many applications, the system must be low cost. For border patrol use, a large number of profiling systems would need to be employed to provide surveillance across miles of territory. The initial deployment and maintenance cost should be minimized. The cost of replacement should also be considered. Threatening targets may wish to destroy deployed sensors in order to remain undetected. To be used in practical application, the profiling sensor must be relatively inexpensive, yet still comply with the above mentioned restrictions of size, power, and bandwidth.

The restrictions imposed upon the profiling sensor limit the scope of useful technologies which can be used to provide a solution. The small size is necessary because it allows the sensor to be conveniently deployed by soldiers and concealed in various terrains. The limited size of the profiling sensor also restricts the amount of hardware available for any one sensor unit. In the field, battery operation is required, yet the small size reduces the available battery allotment. Battery replacement can be expensive in manpower, when a large number of profiling sensors are deployed. The system, however, is intended to reduce manpower. Concealment may also be compromised if soldiers must repeatedly replace a sensor's batteries. Therefore profiling systems intended for persistent and covert surveillance should remain operable for long periods of time without battery replacement.

The system must be low power to remain functional for long periods of time with a limited battery allotment. A large data transmission can increase power consumption, reducing the battery life. Transmitting more information requires a larger bandwidth, which in turn, requires more power to operate. A reduction in data transmission, therefore bandwidth, can increase the sensor's operating time without battery replacement. Recall, the main goal of the system is to simply provide the user with knowledge of a potential human threat. To minimize the transmitted information, collected sensor data can be reduced to a simple notification of the potential threat. Algorithms which operate on the collected data and decrease the amount of information transmitted are needed. They must be able to function with limited processing power to conserve battery life, while performing reliably in real time for many applications.

Algorithms must be able to decide if a human threat is present from the gathered sensor data. For discriminating between humans, animals, and vehicles, we as humans take note of an object's shape immediately. Profiling sensors systems which collect information about the shape of an object can be used to perform classifications of these objects. For example, the height and width of humans and animals are typically different. Humans tend to be taller and narrower, while animals, such as horses or cows, are shorter and wider. By capturing these distinctions in shape, classifications can be performed. The goal of the system is to simply gather enough data to classify potential targets as human, animal, or vehicle. The information collected about a target's shape should be detailed enough to acquire the height and width of the target. Additionally detailed information about a target's shape may not be necessary. Because of this fact, high resolution systems may not be required for the problem of broad scale classification, and a sparse sensor detection system can often be used for the profiling sensor solution in many applications.

An optical trip wire profiling sensor can be used to gather information on a target's shape. The trip wire consists of individual detector and transmitter elements, each with its own supporting optics. When a target passes one such individual trip wire detector, an alert is made, notifying the breaking of the trip wire beam. Multiple detectors configured in a vertical array can be used to capture the height of a target. For a target with a known velocity, the width of the target can also be found. The trip wire need not be a linear vertical array (i.e. randomly spaced in the horizontal direction), but must span across the target in the vertical direction. The trip wire sensor may be active or passive. For active sensors, energy is transmitted from an emitter and received with a collector. The transmitted energy forms the trip wire beam used to detect the target. An image of an

active 16 element tripwire profiling system prototype designed by The University of Memphis is shown in Figure 1-1. This prototype design has a series of emitter / collector pairs on one post, and the other post contains a reflective material which reflects the transmitted energy back to the collector. The passive sensor version collects energy emitted or reflected from the target. When a target traverses the sensors field of view (FOV), a significant change from the background energy is detected. The change in detected energy acts as the breaking of the beam and signals target detection.

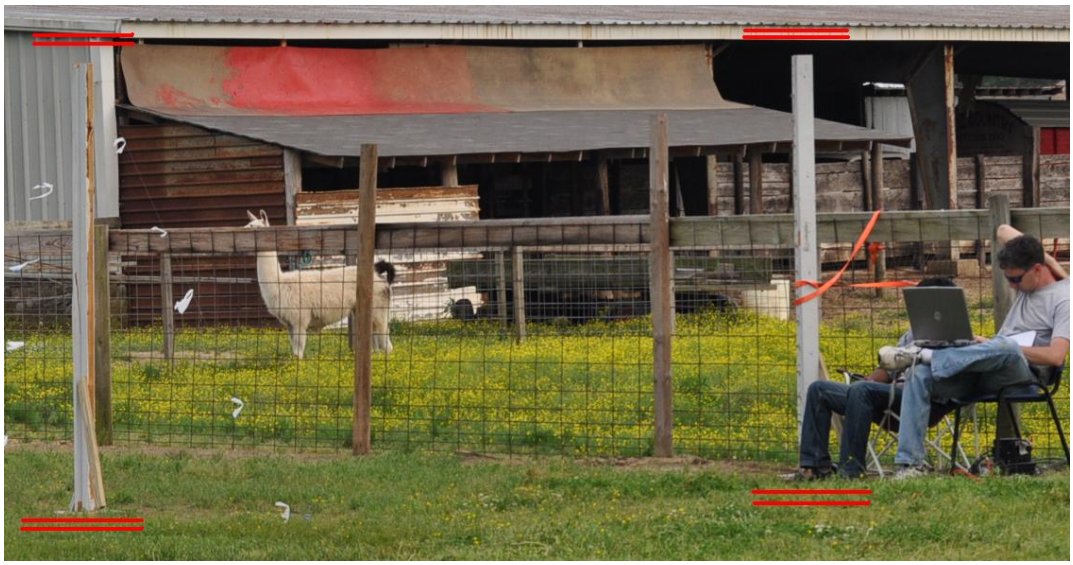


Fig. 1-1. An active tripwire profiling system designed by The University of Memphis.

An alternate approach for the profiling sensor is to place the detector array configuration behind a single optical system. The configuration may include linear arrays, focal plane arrays, or some other design. The passive sensors discussed in this paper have been limited to pyroelectric and microbolometer detectors, though other

detector types can potentially be used. The choice of detector can influence the configuration type. Pyroelectric linear arrays with enough elements to provide high resolution across the target are available on the market. A prototype 128 element pyroelectric linear array, designed by Night Vision and Electronic Sensors Directorate (NVESD), department of the Army, is shown in Figure 1-2. Pyroelectric 2-D arrays, however, typically contain a small number of elements. Microbolometer focal plane arrays with a large number of elements are more common, and, therefore, are used for 2-D array configurations.

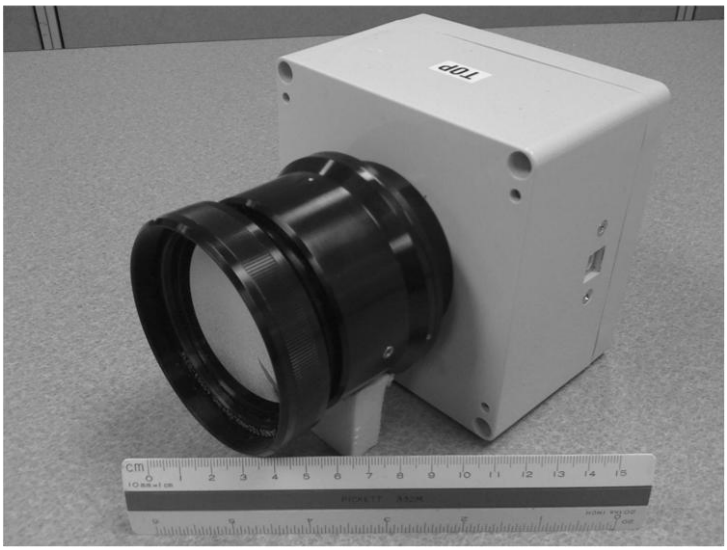


Fig. 1-2. A prototype 128 element pyroelectric array designed by NVESD, dept. of the Army.

There are significant differences in the choice of the profiling sensors described here. Both the trip wire linear array and the single optics linear array require knowledge of a target's velocity to normalize generated profiles along the horizontal axis. Profiles

generated from a single optical system will be scaled according to the target distance from the sensor. The trip wire sensor does not suffer from this distance scaling. The trip wire sensor must also extend in the vertical direction to match the height of the proposed target. This aspect of the trip wire design must be taken into account when covert deployment is desired.

Another important characteristic is the generation of profiles. Active sensors respond to a break in the transmitted energy from the emitter to the detector. For such a system, a single threshold value can be used to determine if the beam has been broken. The output of the sensor is therefore a binary value used to produce the binary profile. For passive sensors, the energy is collected from the scene. A microbolometer detector produces an output proportional to the irradiance impinging on the detector; whereas a pyroelectric detector produces an output proportional to the change in irradiance on the detector. For both the microbolometer and pyroelectric detector, the target signal must be separated from the background signal. For application in various dynamic terrains, a single threshold value cannot be used. An estimate of the background must be determined to segment the target signal to be used in building profiles. Thus there is an additional layer of processing when using passive sensors to create binary profiles from the output of the sensor. Given that the features are extracted from the binary profiles for classification, accurate binary profiles are needed. The signal conditioning process must therefore be able to deliver accurate binary profiles from the sensor output.

1.2 Review of Prior Efforts

Sartain presented the concept of the profiling sensor [1]. Sartain outlined potential designs such as the trip wire system and the single optical system for profiling sensors. The optical and radiometric performance of passive sparse sensor systems was modeled by Klett *et al.* [2]. Robinson *et al.* [3] modeled the system level performance of a passive sparse detector profiling sensor.

Russomano et al [4] constructed a prototype of the active trip wire profiling sensor (see Figure 1-1). Russomano et al [5] reported high classification rates for discriminating humans from non-humans. A library of profile samples was collected [6] and each sensor output was used as a feature in the classification algorithms. The algorithms used were the Naïve Bayesian Classifier (NB), Naïve Bayesian with Linear Discriminant Analysis (LDA+NB), K-Nearest Neighbor Classifier (KNN), Soft Linear Vector Quantization (SLVQ), and Support Vector Machine (SVM). All of these methods yielded classification results of 95% or better. The classification results were generated offline, and all of the classifiers presented may not be applicable for real time processing in a remote profiling sensor. The analysis, however, demonstrates the potential of the profiling sensor and gives a baseline for comparisons.

The single optics system was analyzed by Chari *et al.* [7]. The array configuration used was that of a linear array, simulated from data recorded using a long wave infrared camera (LWIR). The sensor output was differentiated over time and then thresholded to construct a binary profile. The height to width ratio (HWR) of the profile was used as a feature for classification. Once again the classifiers NB, LDA+NB, KNN, and SVM were

used and the classification results for the two class problem of humans and animals ranged between 97% and 98.5%.

The need to establish velocity estimates for accurate classifications was demonstrated by Brown *et al* [8] using simulated data from a LWIR camera. Brown proposed techniques for acquiring velocity estimates using linear arrays. Chari *et al.* [9] analyzed the results of the classifiers NB, KNN, and SVM when the profile HWR, normalized by the velocity estimate, was used as a feature. The classification results, generated from simulated data, improved to 98%, 100%, and 100% for the classifiers NB, KNN, and SVM, respectively.

The classification results of the passive single optics profiling system were reported for simulated data until White *et al.* [10] reported results from a prototype 128 element pyroelectric linear array. The results showed a 93% detection rate and a 92% classification rate, given a correct detection. White *et al.* verified that a low power sparse detector could indeed be used to detect and classify targets. Brown *et al.* [11] further argued that the techniques developed to be used for low power linear arrays, could be used for 2-D focal plane arrays. The benefit was that existing systems using 2-D focal plane arrays could classify targets using a minimal amount of data. Brown *et al.* also demonstrated a 360 degree sparse target detection system, where a focal plane array images a conical mirror, providing for continuous coverage in the horizontal direction. A modified linear array technique which instead applied concentric ring arrays was used to implement the sparse detector system.

There have been significant contributions in the field of catadioptric systems; systems which use lenses and mirrors. Baker and Nayer [12] performed an analysis of

different catadioptric systems to determine which mirrors offer single view points and evaluate defocus blur, caused by mirror curvature, for these systems. Yagi [13] also investigates catadioptric systems which provide omnidirectional or 360 degree viewing. Yagi developed a 360 degree viewing system, which incorporates a conical mirror, and accompanying edge line detection algorithms for use with autonomous robots. Boult *et al*. [14] provide a review of techniques for image thresholding and target tracking when using a 360 degree viewing system for surveillance.

1.3 Preview of Thesis

This thesis discusses techniques for signal conditioning and feature estimation. The signal conditioning techniques are described for passive sensors as more processing is required to detect and produce profiles for passive sensors. The feature estimation techniques described for passive linear arrays may be used for active trip wire sensor systems as well. The following body of this thesis is broken into two parts. The first part describes signal conditioning and feature estimation approaches for linear array systems. The second part depicts a way in which targets can be classified in a 360 degree system using a focal plane array and a conical mirror. Signal conditioning and feature estimation methods are outlined for this special case of the single optics passive profiling system. Results of the described processes are given in both part 1 and part 2. Following parts 1 and 2 is a conclusion and plans for future work.

1.4 Thesis Statement

Statistical modeling based on existing data sets can improve feature estimation for linear array and 360 degree profiling sensors.

2 Linear Array

2.1 Preface

The following section, 2.2 Linear Array, presents an article, *Assessment of a linear pyroelectric array for profile classification* [8], which was published in the proceedings of SPIE 7693 (2010) by Brown *et al*. In this paper, procedures for producing velocity estimates when using a pyroelectric linear array were outlined. The article discusses why velocity estimates are needed for accurate classifications when using a linear array. Two techniques for estimating target velocities are described. One approach involves using two vertical linear arrays separated by a known distance. The time required for the target to cross the known distance is measured and a velocity estimate is calculated. The alternate approach involves only a single linear array tilted at an angle. The slope of the generated profile, induced by the tilt of the sensor, gives information about the target velocity. The section, Detailed Methodology, has been added for clarification. This section contains signal conditioning techniques for developing a binary profile of the target from the sensor output, where a process for detecting the target and extracting the HWR feature of the profile is included. A detailed explanation of the velocity estimation procedure is also given. The pyroelectric linear array was not used; instead, simulated data was generated from a LWIR camera. Results from simulated data

with and without the velocity estimates are provided. The results verify that velocity estimates are needed for accurate classifications.

The following section is relevant in that it argues a need for using velocity estimates and also provides practical methods for producing the velocity estimate. Without the velocity estimate, the profile width may have an inaccurate proportion to the actual target width. Previous papers reported results without the inclusion of velocity estimates. For linear array systems which are already in use, the paper outlines a method in which velocity estimates can be generated without the need for additional hardware. At the same time, however, the methods are viable for focal plane array systems as well. It is also argued that the method used for velocity estimation could even be incorporated in a random active vertical trip wire array, where each sensor has a varying horizontal displacement. Such a system could be a solution to camouflaging the traditional vertical trip wire linear array. It should be noted that random passive vertical trip wire arrays are not recommended due to the complexity of a robust signal conditioning algorithm for extremely sparse data.

2.2 Linear Array Paper

2.2.1 Simulation of Pyroelectric Arrays

The simulation is of a pyroelectric sensor system and simulates, as the name implies, data that would have been the output of such a system. In general, a pyroelectric sensor detects temporal changes in thermal energy and coverts the measured change into an analog signal or current. This analog signal, which can be converted to a voltage, is

measured and given a digital value. The digital value is a measure of the change in temperature within the field of view of the sensor during the sampling period. For multiple detectors, such as in a two dimensional array, the output of each detector forms a pixel in a two dimensional image. For every sampling period or sample, an image frame is created. A sequence of frames would therefore imply a time sequence of two dimensional images, similar to a movie. A linear array of detectors, similarly, produces a one dimensional pixel image frame for each sample. A two dimensional image is formed from a time sequence of one dimensional frames. To scan across a larger field of view, the linear array can be rotated. Often, however, the linear array is left stationary and the movement of the target across the field of view provides the scanning action. It is important to note that when observing a two dimensional image formed by a linear array, one axis represents the outputs of the different sensors and the other axis represents the relative time when the outputs were sampled. The time sequenced frames are then processed to create a binary image(s). The binary image data is used to detect and classify targets.

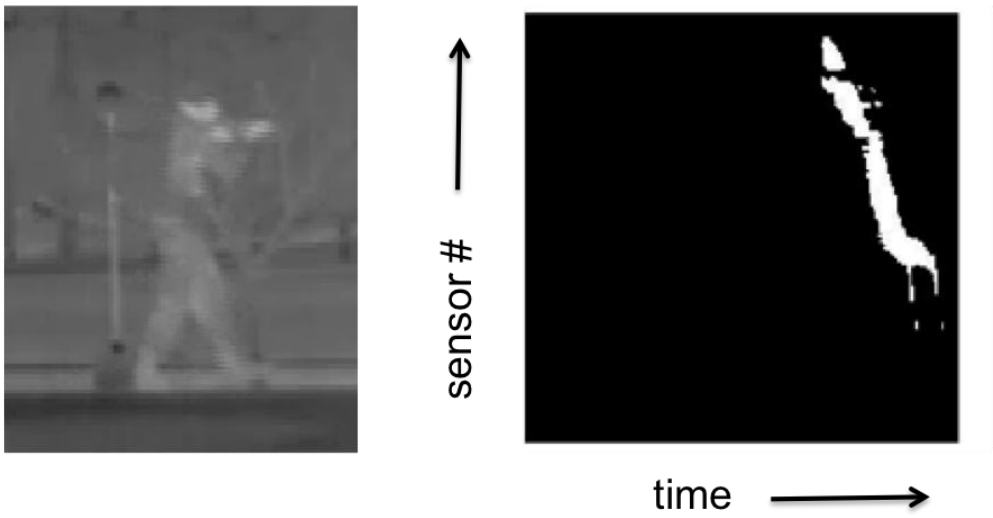


Fig. 2-1. (left) Infrared image of a man with a weapon. (right) Binary profile of human target. This particular profile is tilted as a result of the tilted array simulation.

The simulation replicates the pyroelectric sensor system described. A long wave infrared camera (FLIR A40M) is used to collect infrared video. The infrared video is calibrated using a four panel blackbody, where each panel is a different temperature. Measurements of the camera and blackbody were taken and a linear relation was found between pixel value and temperature across the range of the camera. Given the 8-14 μm spectral band and the calibrated temperature value, the spectral exitance across the scene can be calculated. The power impinging on the detector can be calculated by convolving the input with the system impulse response. The system impulse response is found by taking the inverse Fourier transform of the system transfer function. The transfer function is the product of the atmospheric, optical, and detector transfer functions [15]. For short distances, the atmospheric loss can be considered negligible. The optical transfer function modulates or degrades the input due to diffraction, aberrations, and construction defects.

We will consider the input to be the scene; however, the data collected from the infrared camera has already been modulated due to the optics and detectors of the camera. In general, we would have to account for these effects and add additional degradations to account for the linear array being simulated. For the purposes of investigating the configurations for velocity estimation, we assume an equivalent F# and detector size for the pyroelectric detector system as for the infrared camera. Therefore, the optical transfer function and detector spatial transfer function can be ignored.

The output voltage can be obtained as a product of the power incident on the detector and the responsivity. The responsivity is based upon the induced current responsivity and the electrical responsivity from the preamplifier circuit. For a pyroelectric device, the incident power causes the sensitive element to change temperature with respect to time based upon detector element properties and thermal insulation to a heat sink. The change in temperature produces a proportional charge on the surface of the element, which is known as the pyroelectric effect. Electrodes attached to the sensitive element allow the induced current to flow through a preamplifier, where the frequency dependent current is converted to an output voltage, for voltage mode. Typically, the output voltage of a pyroelectric device will display a rapid rise and slow decay to a sudden change in the incident power. The output voltage for each sensor is then converted to a binary level based on the statistics of the particular sensor and operating environment.

The classification methodology used in this paper operates on binary images. A general understanding of the generation of those images is helpful. It is assumed that "reasonably complete" binary images have been generated. The subjective term

"reasonably complete" simply means that if a human was detected and a binary image formed, the image should contain components of the human such as head, torso, and legs. A binary image of a human target that reflects only the head and torso would not be considered reasonably complete and therefore would not have been used in the subsequent analysis. Reasonably incomplete and complete profiles are shown in Figure 2-2 and 2-3 respectively. The temporal characteristics of a pyroelectric device produce an impulse response with both positive and negative components. The result can be a slight halo effect near edges of the target. In an effort to produce reasonably complete images, these temporal effects were ignored in the simulation. In a best-case scenario actual images produced from a pyroelectric detector will be of the same quality as images produced from the simulation.

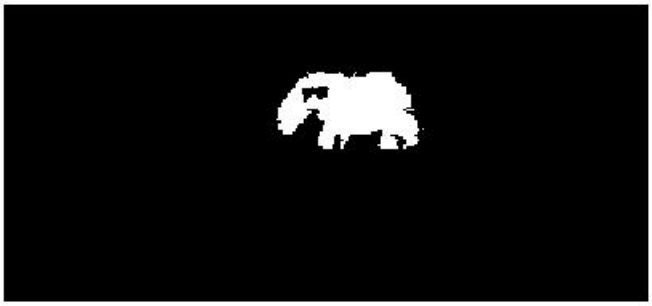


Fig. 2-2. A "reasonably incomplete" binary profile of a horse. The legs of the horse are not captured in the profile.

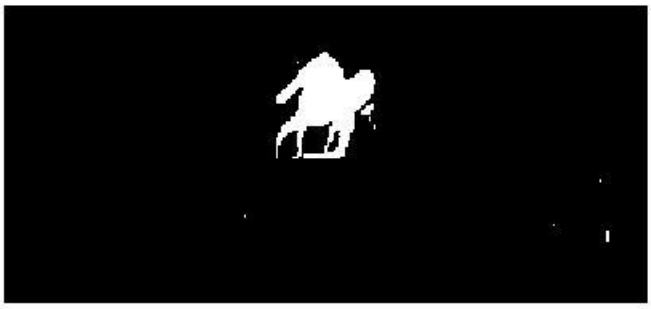


Fig. 2-3. A “reasonably complete” binary profile of a horse.

2.2.2 **Velocity Estimation**

It has been shown that the height to width ratio of an object is a useful feature for discriminating between humans and animals. Therefore an accurate measurement of the height to width ratio is essential to accurate classification.

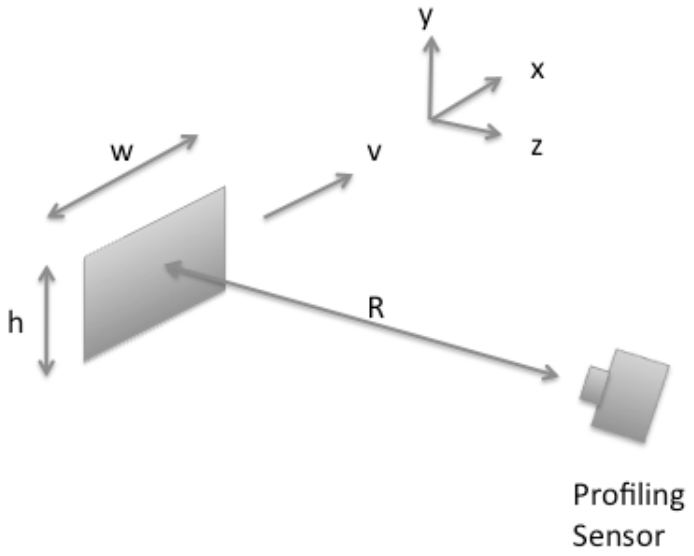


Fig. 2-4. Geometry for profiling sensor.

Consider the moving block illustrated in Figure 2-4. The block has an actual height to width ratio (HWR) given by

$$\rho = \frac{h}{w} \tag{2.1}$$

Ignoring spatial and temporal blurring effects, the measured HWR in the binary output of the sensor can be shown to be

$$\rho_m = \rho \frac{\vartheta \Delta t}{\theta_v} \tag{2.2}$$

where $\vartheta = \frac{v}{R}$ is the angular velocity of the target as viewed from the sensor, Δt is the time between samples from the detector, and θ_V is the vertical instantaneous field of view (IFOV) of the detector. Given that the sampling time and detector IFOV are usually known quantities, it is clear that, in general, the HWR will be subject to error depending on how accurately the angular velocity of the object is known.

2.2.3 Two Columns

The two column approach measures angular velocity using two pyroelectric vertical linear arrays separated by an angular distance *d*. The number of frames between the target passing column 1 and the target passing column 2 designates time *t*. With a fixed distance *d* and measured time *t*, the velocity can be calculated. The accuracy of time *t* determines the accuracy of the velocity. This relationship suggests that an accurate method of determining exactly when a target passes a column produces an accurate velocity. A very simple approach is to record the time when the target first enters a column's field of view. As any method, this method can be subject to error. For one,

noise that remains in the binary profile can be mistaken as target by an algorithm that searches for the first instance of signal. To reduce the impact of these possible errors, the time of the last instance of signal can also be recorded. This, in effect, gives the time it takes for a target to enter and leave columns 1 and 2. Taking the average of the difference of the enter times and the difference of the leave times can give an estimate of the time *t* with low computational complexity. A good cropping algorithm that can isolate the target body can be used to ignore some noise elements. This method, however, can also be affected by elements of the target which may be moving, such as swinging arms and legs for a human target. Two runners may not cross the finish line at the same time due to one runner extending an arm, yet they were running with equal speeds. This error in the time calculation will have a larger impact for smaller values of distance d.

An alternate, but similar method was used, which was less susceptible to the previously mentioned errors. The second method also uses a cropping algorithm which ignores any signal outside of the identified target area. The target is then analyzed to find a center line along the vertical axis through the target. The "center" line gives slope and y-intercept information. Using the slope, y-intercept, and a reference point such as top (head), bottom (feet), or midpoint, a consistent marker can be used to more accurately determine when a target passes a column. Due to the initial cropping some noise can be eliminated, but by finding the center line, remaining noise is averaged with signal to reduce its effects. This second method is also less susceptible to the swinging arms problem as it attempts to track the target body as a whole. This approach is more complex than the previous approach but is more reliable. This approach was used in the results shown in this paper.

2.2.4 Tilted Array

The tilted array approach simply tilts or rotates a linear array by a given angle. When tilted at a given angle, a fixed horizontal distance between sensors in a linear array is introduced. By knowing the distance and finding the time between signal detection for two adjacent sensors, one can calculate an approximate velocity. This concept is similar to the two column approach, except that the horizontal distance is now a function of the tilt angle. Any tilt angle can be used to produce a horizontal distance between sensors and be used to produce a velocity estimate. Therefore, two samples, one collected with tilt angle a1 and the other collected with tilt angle a2, can be analyzed against each other. But the choice of tilt angle is not altogether arbitrary. We shall assume an angle of 90 degrees to represent a vertical array. For angles close to 90 degrees, the resulting horizontal distance d between sensors will be small. Assuming a constant velocity v, as d approaches 0, for d=v*t, t must become smaller. The limit to how small t can be depends on the sensor's sample rate. For a sensor of 30 Hz, t cannot be smaller than 1/30 sec. If we assume that t has a minimal value of 1/sample rate, and d is fixed, then velocities higher than v will not be detectable. In order to detect higher velocities, assuming the sample rate is not increased for a sensor system, the distance d must be increased. By reducing the tilt angle, the horizontal spacing of sensors increases, along with the highest detectable velocity. But there is also a limit as to how much this angle can be reduced. Practically, the angle must be large enough so that the target can be completely detected across its vertical length. Keeping this in mind, a 45-degree angle was used as it maximizes both vertical and horizontal distance. The vertical distance reduces at a greater

rate with respect to angle, for angles smaller than 45 degrees. The choice of 45 degrees also creates some computational simplifications, as we will see later. An image of the 128 element prototype pyroelectric linear array being tilted by 45 degrees during a data collection conducted by The University of Memphis is shown in Figure 2-5.

Fig. 2-5. 128 element prototype pyroelectric linear array being tilted by 45 degrees during a data collection conducted by The University of Memphis

The binary profile from a tilted array will be tilted by some angle dependent on target velocity and sensor tilt. For a binary profile generated by a linear array, the vertical axis is the output of each sensor in the array and the horizontal axis is the output for each frame. Therefore, the horizontal distance in the binary profile (frame distance) is the change in time, where each sample is (1/sample rate) seconds in time. The vertical

distance between two pixels is equivalent to the vertical distance between sensors. The vertical distance is related to the horizontal distance between two sensors by the trigonometric relationship

$$d_h = \frac{d_v}{\tan(\alpha)} \tag{2.3}$$

here d_h is the horizontal distance, d_v is the vertical distance and α is the sensor tilt angle. When the sensor angle is 45 degrees, the horizontal distance is equal to the vertical distance so that

$$v = \frac{d_h}{\Delta t} = \frac{d_v}{\Delta t} \tag{2.4}$$

whereas before, Δt is the sample time. Thus velocity is a measure of the slope of the binary profile. Slower targets will have a greater slope while faster targets will approach a vertical orientation.

Measuring the slope of a binary profile, when using the tilt approach, can provide an estimate of the velocity. For the ideal case of a rectangular target, the estimate is highly accurate, as might be expected. Real targets are not perfectly rectangular, however. Human targets typically have a generally rectangular shape. Irregularities may arise due to body type and more particularly, body motion. The problem of swinging arms does reduce the accuracy. An overall slope of a human target can easily be determined, using both the front and back sides of the profile. Animals, such as cows or horses, will also provide a rectangular profile. There will be irregularities near regions of protrusion such as the head and neck. This error can be reduced by analyzing the

derivative of the slope and finding large spikes greater than two standard deviations. These spikes will designate regions of protrusion and can be ignored when calculating slope. It is important to remember that the slope caused by tilting the sensor is the value we are seeking. Slope caused by the inherent shape of the target can affect slope produced velocity readings. Targets such as vehicles often have slopes along the front and sometimes rear windshields, depending upon the vehicle. These slopes are inherent to the shape of the target and cannot be completely corrected for. Once again, however, averaging the front and rear slopes of the profile provide for a good estimate of velocity.

A drawback of the tilted array is that it can produce inaccurate results for targets moving at an angle towards or away from the detector. The horizontal distance that each sensor sees is a function of the IFOV and the distance from the target and the sensor along the z-axis. If the z distance from the target (see Figure 2-4) and detectors changes as the target moves across the field of view (FOV), then the horizontal distance seen by each sensor also changes. By knowing the target z distance and angle of target approach, the inaccuracies may be corrected for. This, however, is unlikely in practical situations, and therefore the target is limited to an approximately constant z distance as the target is scanned.

2.2.5 Detailed Methodology

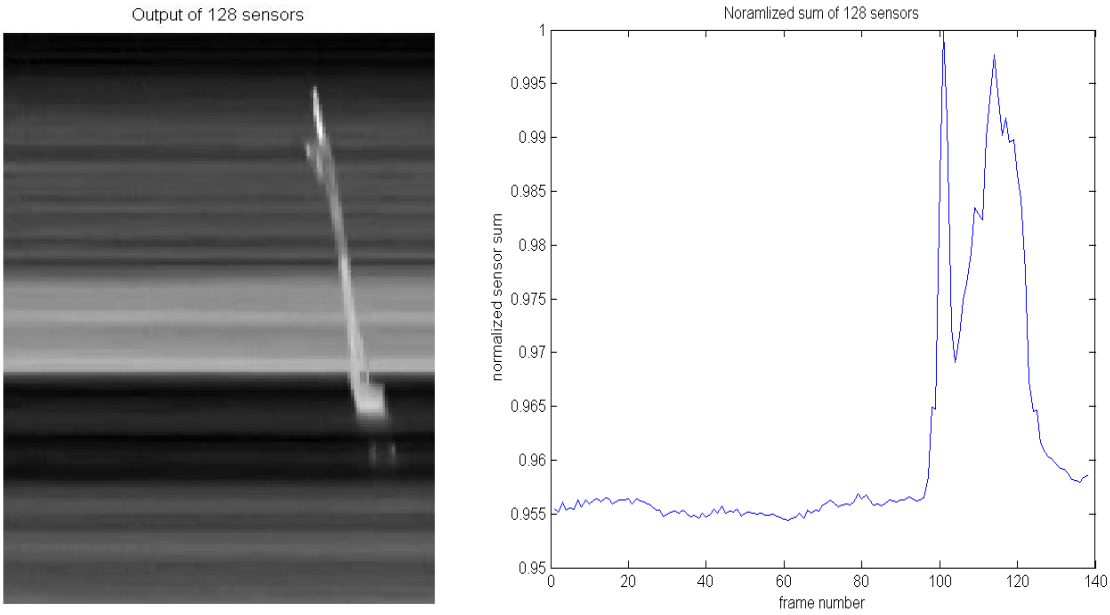
The overall process for extracting features for classification has been outlined for both velocity estimation methods, but results may vary according to implementation. The extracted feature is dependent on the generated profile shape of the target. A better estimate of the generated profile shape provides a better estimate of the feature, which

will be used to estimate the class of object. Inexact signal conditioning processes and excessive noise can lead to poorly generated profiles and inaccurate feature estimations. The process used for conditioning the output signal and extracting features in both the two column array and tilted array approach is presented here.

Background subtraction and target detection are initial steps in evaluating profiles. Due to the nature of the data, these two steps have been combined. Recall, for simulated data, a video was captured in which a known target crossed the sensor systems field of view. The data is extracted from each video at pixel locations corresponding to the desired sensor configuration. The output of M sensors for N frames produces an M by N array. The known target must be detected from this 2-D array sensor output. To detect the location of the target, the N columns of the output array can be summed, producing a 1 by N vector. This summation vector at each index contains the sum of all M sensors for a given frame. Assuming the background, as a whole, does not experience sharp fluctuations, then the summation of the M sensors will also not experience sharp fluctuations. Therefore, any sharp fluctuations experienced by the sensor summation are due to moving targets in scene. Figure 2-6 depicts the sensor output and the corresponding sensor summation for a human target. In this example, the frames in which the target is present are clearly identifiable.

To extract the target frames, the mean and standard deviation are calculated for the summation vector. The mean is subtracted from the summation to remove the bias and the remaining signal is compared against the standard deviation. Signal which is greater than a standard deviation is considered target where signal less than a standard deviation is considered intermediate background. The term intermediate background is

used because the result is not completely background. The decision process between target and background consists of modeling the background and separating signal which is not background. In modeling the background, the statistics (mean and variance) of the background are found and modeled



(a) (b)

Fig. 2-6. (a) The output of 128 sensors extracted from video. (b) The normalized sum of the 128 sensors for each frame.

as a Gaussian distribution. The majority (68%) of the energy contained in a Gaussian distribution can be found within one standard deviations of the mean. The amount of energy improves to 95% and 99.7% when the distribution is evaluated to two and three standard deviations. Signal outside of three standard deviations is unlikely to have been caused by the distribution modeled. The background is intended to be modeled, but that

was indeed not the case. In this case, statistics are generated for both the target and the background. A decision based upon this target/background model can be inaccurate. Therefore, the term intermediate background is used to denote the result. The statistics of the intermediated background are then found and the process of target and background segmentation is repeated. Using a decision value less than 1 standard deviation improves the segmentation process. The result of the second segmentation process produces a closer approximation of background and target signal. The benefit of the segmentation process is that the frames in which the target is present are identified. Frames in which there is not a target present can be ignored.

The issue of pixel values gradually changing due to changing temperature in the scene must also be addressed. The changes in scene temperature may be such that the algorithm detects two regions of supposed target. One region is the real target, where as the other region consists of the frames in which the background has changed temperature considerably. In a small window of frames, it is assumed that the real target region will be larger than the region caused by changing temperature. Thus, the larger region can be evaluated as being due to a real target. It may be necessary to allow for minor gaps when evaluating these large regions of consecutive frames.

This process was implemented for simulation purposes but theoretically can be used for real time evaluation as well. For real time systems, a window or history of previous frames is stored in memory. The history window is then evaluated in the above mentioned fashion. This procedure works well when there is both target and background contained in the window. When there is only background contained in the signal, then the standard deviation calculated for the background only signal is significantly less than the

background and target combination signal. Thus the initially calculated standard deviation for the sum of columns in the history window can be used to determine the presence of a target within the signal. If it is determined there is no target, then the signal is not evaluated further. If a target is determined to be present, the signal is evaluated for a region of consecutive frames containing target. This region, however, should not lie on the boarder of the window, as that would imply only part of the target has been detected. Imposing this restriction implies that the window is large enough to contain a profile of an expected target moving at an expected speed and that the target does not remain in the field of view for an unreasonable amount of time. The window size should be balanced to accommodate for the presence of a full target profile, but also small enough that the temporally localized estimate of the background is applicable. Here it is also assumed there is significant contrast between the target and background. Under these assumptions, the method holds true even when the background is changing rapidly. Another approach to the problem of target detection is frame to frame differencing [10] to find changes that would imply a target is present. For this method, a window must also be stored in memory to capture profiles. Similarly, it assumed that there exists sufficient contrast between target and background.

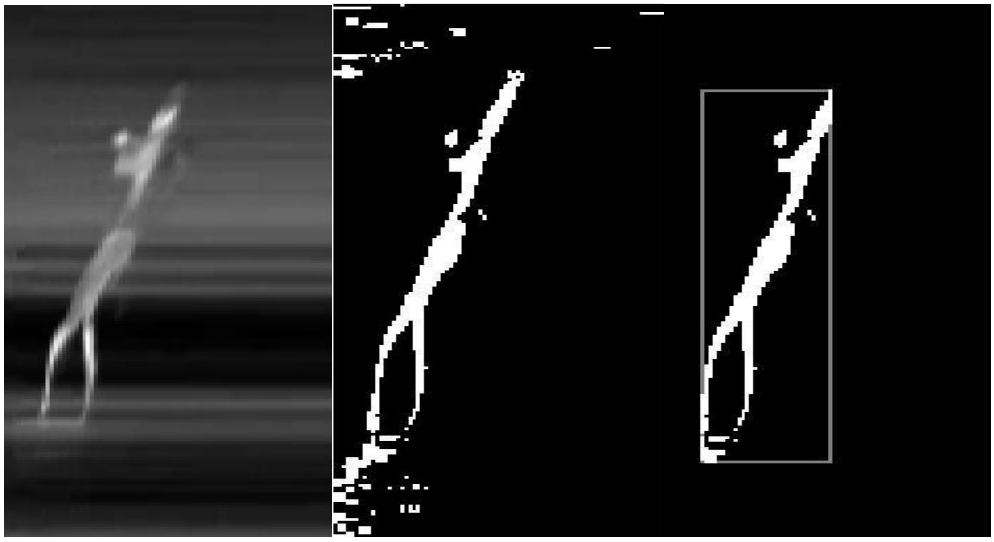
The 2-D image generated by the output of the sensors over multiple frames can be binarized. By determining the frames which consist only of background, a model of the background can be determined. The mean and standard deviation of each sensor output for the background frames can be calculated. Once again the technique of background subtraction and thresholding can be applied. In this case, however, the technique is applied to each pixel, such that

$$pixel_{ij} = \begin{cases} 1, & when\ pixel_i > \mu_{ib} + 3\sigma_{ib} \\ 1, & when\ pixel_i < \mu_{ib} - 3\sigma_{ib} \\ 0, & otherwise \end{cases} \quad (2.5)$$

where μ_{ib} is the mean value and σ_{ib} is the standard deviation of pixel *i* for the frames corresponding to background signal. A decision threshold of three standard deviations from the background should remove a majority of the noise, but once again, requires a level of contrast between target and background. Because the target has already been detected within a given region of frames, background frames can immediately be discarded from target analysis. The result is that unwanted noise in regions where it is known there is not a target is not evaluated. The vertical extent of the target in the binary profile can be determined by taking the sum of the M rows of the binary image creating an M by 1 vector. The row sum vector is evaluated for regions of consecutive indices with values greater than zero. The largest of these regions implies the vertical extent of the target. As with determining the frames occupied by the target, small gaps may be included within these large regions. Binary profiles of targets may not exhibit full connectivity, and by allowing small gaps, the chance of generating partial profiles is reduced. This gap allowance also increases the chance of noise being accepted, which gives more importance to previous noise removal techniques. It should be noted that the row summation must be performed after the binarization. The row sums before the binarization are uncorrelated and the target signal can be difficult to locate.

At this point, a significant amount of information has been extracted. The target has been located across a given number of pixels for a given number of frames. A profile

of the target has been created for visual display and human interpretation. Figure 2-7 illustrates the results of the target detection process. The vertical extent of the profile is synonymous to the target height, as is evident in Figure 2-7(c). The width of the profile can be found by taking the max value of the row sum. Since the M by 1 row sum vector is formed by summing the binary pixels of each row, a noisy image may produce inaccurate values of the profile width. The unmodified height to width ratio has been shown to produce high classification results. The steps described in generating the detected target and HWR are valid for vertical linear array or tilted array, as well as pyroelectric sensors or microbolometers. For more accurate classifications, it has been shown that velocity estimates of targets are needed.



(a) (b) (c)

Fig. 2-7. (a) The sensor output for a 128 element tilted linear array. (b) The binarization of *a*. (c) The detected target profile located within *b*.

The velocity estimate can be acquired in different fashions depending mainly on the hardware in use. For a 2-D array, the two columns or the tilted array approaches discussed previously may be used. For a single linear array, however, the tilted array approach is applicable. The methods for estimating velocity for a two column array have been sufficiently discussed and here we give further discussion to methods in estimating velocity for the tilted array. Let us assume that our target is a rectangular body with constant velocity. The array configuration is a 1 by M tilted array, where the distance between each sensor pair is known. The velocity, $v_{i,j}$, calculated between $pixel_i$ and $pixel_j$ is equal for all values of i and j, where i,j = 1, 2, …, M and $i \neq j$. Let us then expand our definition of a target from a strict rectangular shape, to a shape which is approximately a rectangle. For such a target, $v_{i,j}$ is no longer a constant for all values of i and j. Determining the values of $v_{i,j}$ for all i and j, where there are $\frac{M(M-1)}{2}$ distinct combinations and taking the mean of those values can produce a reasonable estimate for the target velocity. The mean can be expressed as

$$mean\ velocity = \frac{2}{M(M-1)} \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} v_{i,j} \tag{2.6}$$

The mean velocity is valid for objects which are generally rectangular. The approximation is susceptible to error when targets do not have a simple rectangular shape. Human targets can be approximated as rectangular targets but others cannot. A pickup truck is one such object which cannot be approximated as a simple rectangle. Figure 2-8 illustrates the pickup example. Examining the left side or the back of the

vehicle, it is clear that an overall slope is due to the tilted array. But there is a protrusion from the cab to the bed of the truck due to the natural shape of the vehicle. This protrusion will produce inaccurate velocity estimates. To mitigate this error, a small sliding window of length ΔL can be evaluated for $v_{i,j}$ where $|i-j| \leq \Delta L$. As ΔL is made smaller, the error resulting from the protrusion is reduced, but at the cost of limiting the maximum detectable velocity. This pair wise pixel estimation of velocity can be calculated for both the front and back slopes of the target. The average of the front and back slope provides an estimate of the target velocity.

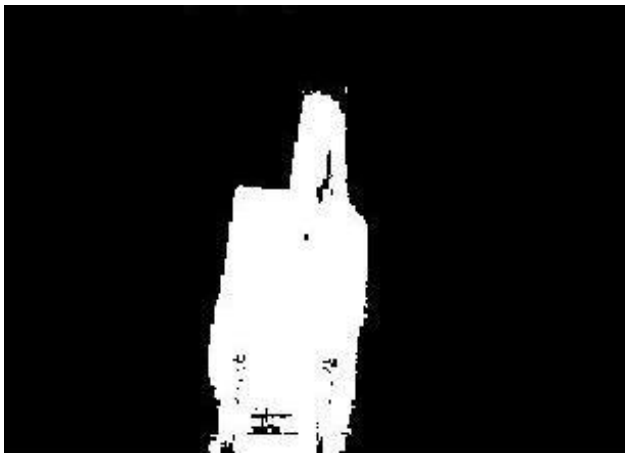


Fig. 2-8 Binary profile of a truck produced by a tilted linear array

An alternate approach to calculating the velocity is by calculating the slope. The midpoint of the target can be found for each row, reducing the 2-D profile of the target to a single 1-D function. A best fit line can be attained from the 1-D function, where the slope of the line is representative of the target velocity. This method also suffers from the effects of irregular shapes and protrusions. To reduce these negative effects, the

derivative of the 1-D function of mid points is found. The derivative gives the change between f(x) and f(x+1). Regions where the slope or change in f(x) is relatively constant will produce a relatively constant velocity. The locations where dramatic protrusions can be found will have large derivative values. By locating these large derivative values, the profile can be divided into regions. The best fit line of the midpoints can be determined for each region and a weighted average of all the slopes can be calculated. The weights are determined according to the length of the region with respect to the sum of the lengths of all regions.

The pair wise pixel estimation was used for the results presented in this paper. Both methods are subject to inaccuracies from targets which exhibit a natural inherent slope, such as a vehicle with a slanted windshield. The pair wise pixel estimation must calculate a varying number of pixel pair velocities due to the height of the target. After all of the distinct combinations have been calculated, then it is a simply an evenly weighted average. The midpoint method must first calculate the midpoint line. The derivative is produced for determining regions, and for each region, a best fit line is determined. The results are then averaged using weights. Depending on the hardware being used, the pair wise pixel estimation can be a simpler process to implement. Another advantage of the pair wise pixel estimation is that it can be used for other sensor configurations. The method was actually developed for use with a randomly scattered sensor configuration. Here the sensor configuration can additionally be applied to active trip wire sensor systems. In some cases, vertical linear arrays may not be concealable, thus eliminating the effectiveness of the sensor. Randomly scattered detectors, may be more effective with respect to concealment but requires reliable methods to process the data for classification.

The potential versatility of the pair wise pixel method led to the decision of using the method for these experiments.

2.2.6 Classification Techniques

The Naïve Bayesian classifier was used for classification [7]. The technique used assumes Gaussian distributions. Each class's mean and standard deviation is calculated for each feature using the training data. The distance of each sample from each class is calculated. The training data was composed from the data set using the leave one out cross validation method.

The data set included 50 humans, 52 animals (horses and llamas), and 90 vehicles (truck, SUV, mini-van, and sedan). Two different problems were analyzed. For the first problem the classes were human and non-human. The animal and vehicle data was combined to form the class non-human for the two class problem. Roussomanno *et.al.* [5] suggested that results can be improved if animals and vehicles were separated statistically and treated as separate classes. According to Roussomanno *et. al.*, after the classifications were accomplished the results of animal and vehicle classifications would be combined to form non-human. We shall call the first approach two class problem A and the second, two-class problem B. Both approaches were tested. The feature used in the two-class problem is a ratio of height and width (HWR). Jacobs *et al* [16] show classification results of 95% and above using the HWR feature and Bayesian classifier when discriminating humans, animals, and vehicles.

The second problem classifies samples as human, animal, or vehicle. For the three-class problem, HWR is not sufficient as a single feature, when compared to the

two-class problem. The standard deviation for animal and vehicle are too large for accurately distinguishing between the two. In Jacobs et al, the targets were segmented into six regions and the percentage of detected target in each region used as features for the two-class problem stated above. This approach was modified as follows. The target was segmented into six square regions (2x3). The lower left region (2,1) was analyzed for the percentage of target and used as a feature. This region was used because animals such horse, cows, and llamas tend to give smaller signatures in their lower regions, whereas vehicles have strong signatures from the engine and exhaust in lower regions. HWR was also used as a feature since it has been shown to accurately discriminate humans. The problem was divided into two stages such that humans were first identified using the HWR for stage one. Stage two consisted of discriminating non-human targets as either animal or vehicle using regional percentages. For both the two and three-class problem the HWR feature was replaced with a product of HWR and velocity.

2.2.7 Results

The data was classified using the techniques described in the previous section. The two-class problem was analyzed first. In Table 2-1, the HWR feature alone gives the expected results of 95% or better with some improvement in the two-class B approach. Significant improvement is shown when velocity information from the tilted array and the two array columns is used.

Table 2-1. Two class (human and non-human) classification results using height to width ratio modified by the tilted array velocity, and two columns velocity. For two class A, animal and vehicle data were combined before classification. For two class B, results of animal and vehicle classifications were combined.

	Two Class A	Two Class B
Unmodified h/w ratio	95.48%	96.19%
H/W using tilted array velocity	99.65%	99.65%
H/W using two columns velocity	98.59%	98.65%

In Table 2-2, the results of the three-class problem are shown. Here we see the same trend of increase in the results of the tilted array and two columns array. One observation of analyzing velocities is that the results of the unmodified h/w approach may be optimistic. If the data is not modified by velocity, then apparent widths of profiles are a function of velocity and actual target width. In data collections, it can be impractical to have horses or various animals run at various speeds in a controlled environment. Also, vehicles typically move faster than a walking human or animal, even when driven slowly. Fast moving vehicles and slow animals can cause unnatural distinctions between classes if unmodified profile widths are used in the creation of features.

Table 1-2. Three class (human, animal, and vehicle) classification results using height to width ratio modified by the tilted array velocity and two columns velocity.

	Three Class
Unmodified h/w ratio	94.35%
H/W using tilted array velocity	97.34%
H/W using two columns velocity	96.67%

2.3 Conclusion

The results confirm the need for velocity estimation. By incorporating effective signal conditioning and feature estimation techniques, the additional velocity information can be acquired without the need for additional hardware. Thus, the same hardware could be used to acquire an increased dimension in data. Without the tilted array approach, the user is restricted to adding additional sensing elements or using a focal plane array to acquire velocity estimates. One of the main benefits of pyroelectric linear arrays is their power consumption. Dias [17] advertises the 128 element 128LTI and the 256 element 256LTI as consuming 40mW of power. In comparison, SCD offers the 25μ Bird 384 (384x288) focal plane array which uses 220mW of power [18]. Even Sofradir's 17μ Pico (640x480) [19], which advertizes power consumption as low 150 mW, still consumes nearly four times as much of the power as the pyroelectric linear array. It may be, however, that some applications are not as restricted in respect to power consumption or may incorporate a focal plane array in an existing system. In such cases, the presented signal conditioning and feature estimation processes are still valid and allow for broad

scale classification while using minimal data and processing power. Overall, these techniques give the user more flexibility in choice of hardware which allows for versatility of design and in many cases, cost savings.

3 360 Profiling Sensor

3.1 Introduction

The profiling sensor has many applications in security. Homeland security is interested in securing miles of unmanned borders. The military is seeking solutions to persistent surveillance, intelligence, and reconnaissance in many scenarios. There could even be commercial uses, such as farmers monitoring animals, provided the sensors were affordable. The trip wire and single optics linear array are able to securely monitor paths, roads, washes, and other similar points of interests. In some applications, it is preferable to have 360 degree surveillance coverage. By combining a LWIR camera and a conical mirror, 360 degree coverage can be achieved. This particular sensor setup can be considered a special case of the single optics profiling sensor with a focal plane array detector configuration.

The following half of this paper describes a system which enables 360 degree vision and pre-processing techniques which allow broad scale classifications. The general system design is presented and analyzed with respect to classification parameters. The analysis is necessary because previous profiling sensors do not contain a mirror component. Though it may be intuitive to determine potential features for classification, we show the reasoning behind our particular choice of features. The methods for image

segmentation, target detection, and feature extraction follow the system analysis. Classification results are then presented, followed by a conclusion.

3.2 System Analysis

To achieve 360 degree vision, a LWIR camera is focused onto a conical mirror. The cone provides continuous coverage in the horizontal direction and limited coverage in the vertical direction. Given the purpose of the profiling sensor is to classify humans, animals, and vehicles, full vertical coverage is not necessary for the task. The layout of the system can be viewed in Figure 3-1. A prototype 360 profiling system, designed and constructed by Jeremy Brown, Robert Jordan, and Eddie Jacobs, of The University of Memphis, is shown in Figure 3-2. This prototype system is designed to be used with a Flir A40 LWIR camera. The system is made to be invertible, where mirrors can be interchanged for various fields of view. In this analysis, it is assumed that the mirror is centrally aligned with the camera and that the mirror subtends minimum field of view, FOV_m, of the camera as shown in Figure 3-3.

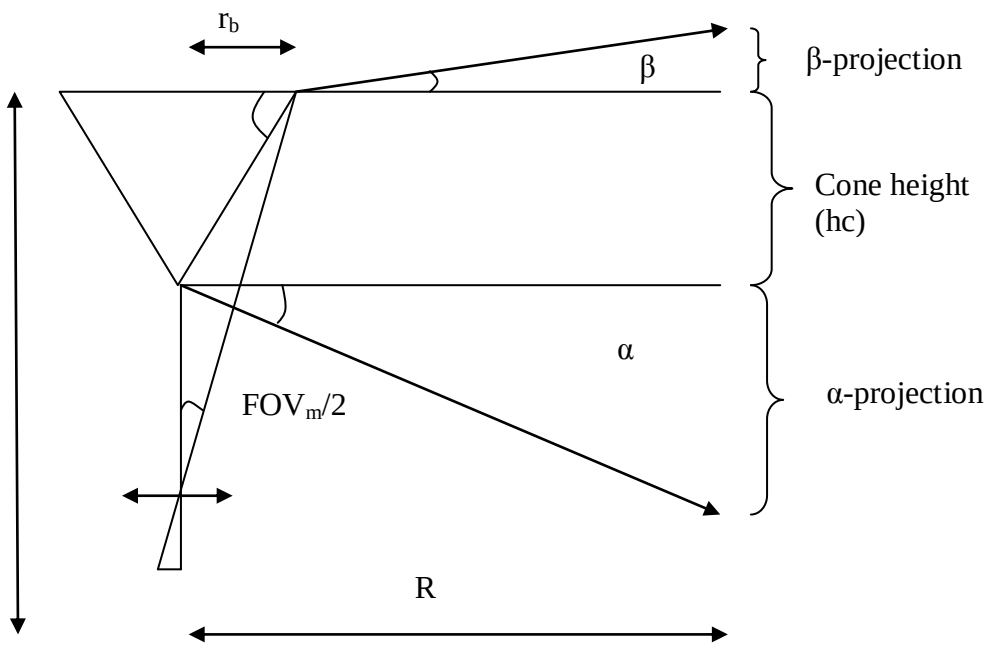


Fig. 3-1. Layout of the 360 profiling sensor.

Fig. 3-2. A prototype 360 profiling system, designed and constructed by Jeremy Brown, Robert Jordan, and Eddie Jacobs, of The University of Memphis

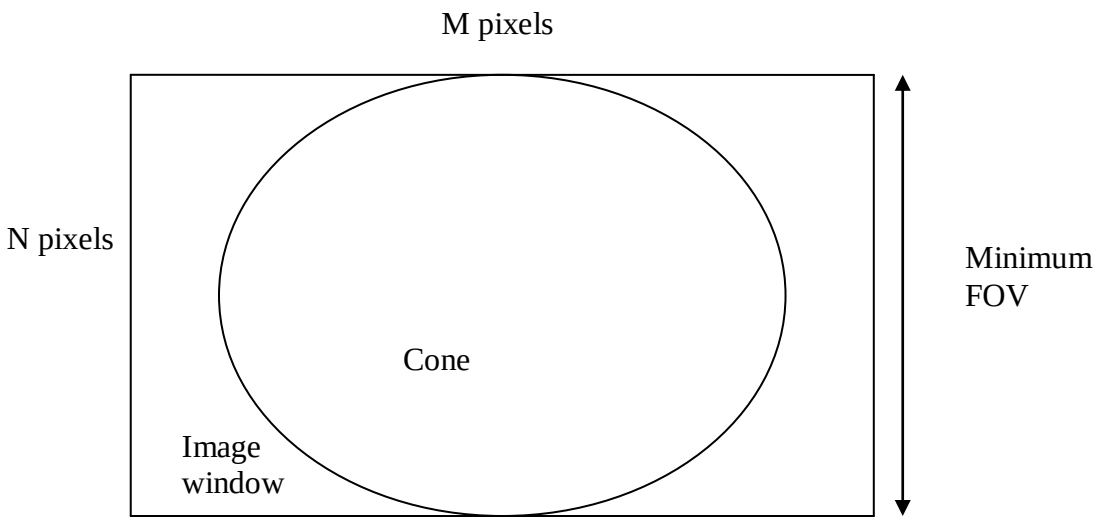


Fig. 3-3. The cone subtends the minimum field of view of the camera.

The vertical and horizontal components of the mirror can be analyzed separately to provide an idea of the imaging characteristics. Along the vertical component, the cone acts as a flat mirror. The field of view of the camera is simply redirected based on the angle of the mirror. The angles α and β can be determined by the following equations

$$\alpha = 90 - 2\phi \tag{3.1}$$

$$\beta = 2\phi - 90 + \frac{FOV_m}{2} \tag{3.2}$$

$$\phi = tan^{-1}(\frac{h_c}{r_b}) \tag{3.3}$$

where h_c is the height of the cone, r_b is the radius at the base of the cone. Along the horizontal component, the cone operates as a spherical mirror with focal point F. Figure 3-4 illustrates the location of the focal point for a spherical mirror.

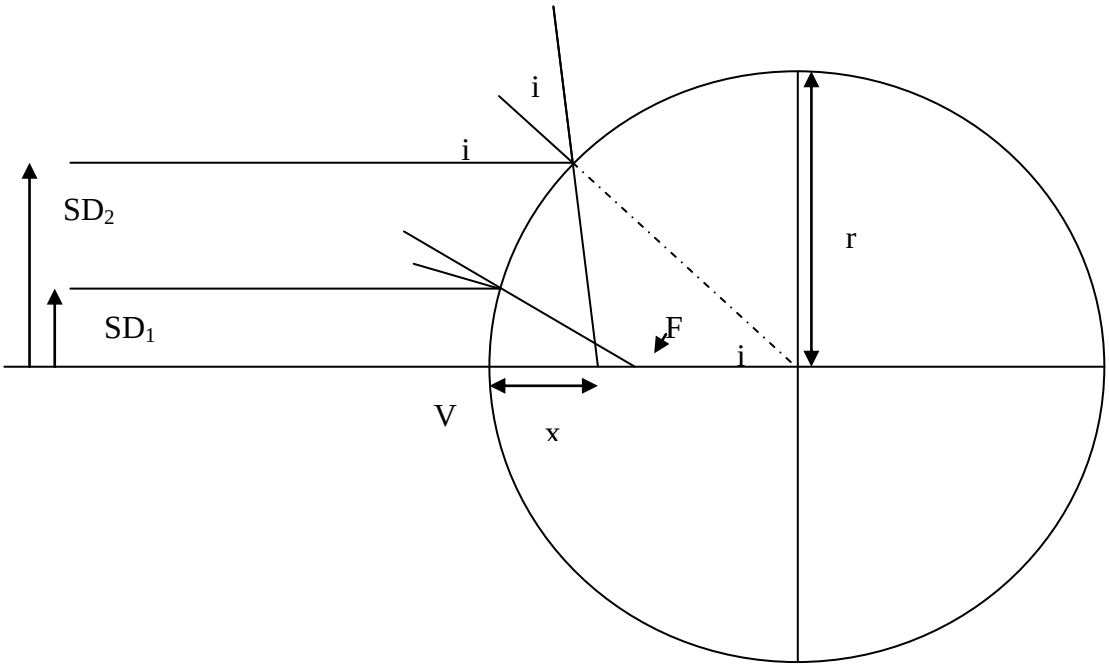


Fig. 3-4. Focal point of a spherical mirror.

The focus can be found [20] using

$$x = \frac{r}{2}\left(2 - \frac{1}{\cos i}\right) \tag{3.4}$$

$$\sin i = \frac{SD}{r} \tag{3.5}$$

When the source distance, SD, from the imaging axis is sufficiently small, the focal point F will be located at r/2. At different heights along the vertical extent of the cone, the radius will have different values. Therefore, imaging at one point along the vertical extent of the cone, may leave other portions of the cone out of focus. The spherical mirror follows the lens law [21] such that

$$\frac{1}{f} = \frac{1}{z_i} + \frac{1}{z_o} \tag{3.6}$$

where z_i and z_o are the image and object distance, respectively. The lateral magnification [20], m, is then found to be

$$m = \frac{y_i}{y_z} = -\frac{z_i}{z_o} \tag{3.7}$$

where y_i and y_z are the image and object lateral distances, respectively. The conical mirror therefore induces a magnification of images in the radial component, but not in the vertical component.

The practical images generated using a conical mirror are blurry, due to out of focus energy, and magnified, due to the curvature of the mirror. The width of a target projected onto the mirror, w', is inversely proportional to the range of the target, R. Using similar triangles, as shown in Figure 3-5, it is clear that

$$w' = \frac{r'W}{R} \tag{3.8}$$

where W is the original width of the target and r' is the radius of the cone at a given height.

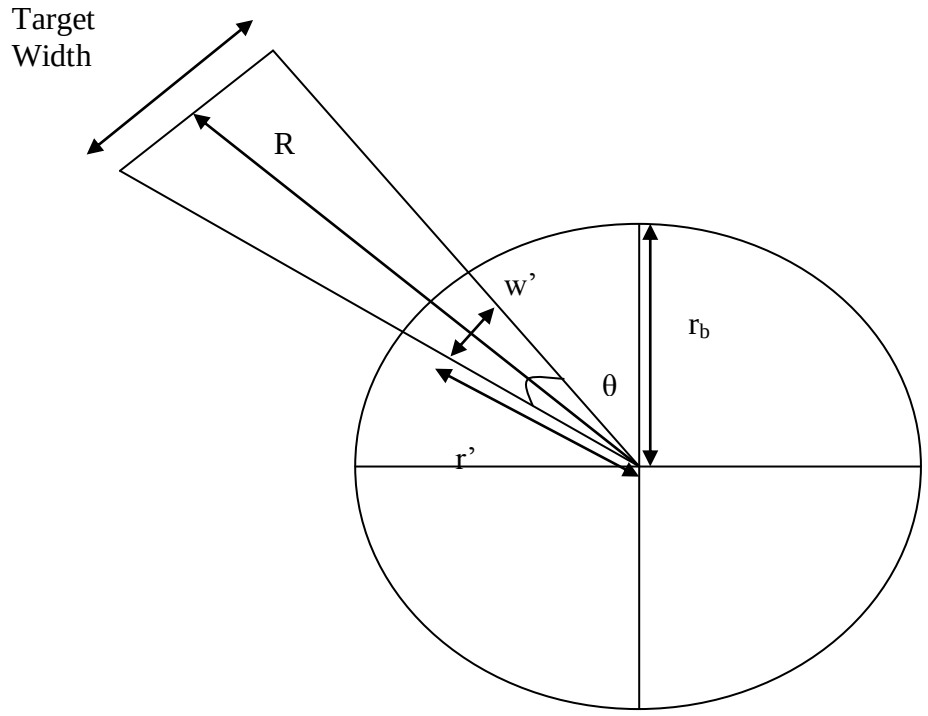


Fig. 3-5. Projection of the target width onto the cone.

To reemphasize this fact, consider Figure 3-5. The projection of a target onto the cone is determined by the angle the target subtends. The angle θ can be determined by

$$\theta = 2tan^{-1}(\frac{W/2}{R}) \tag{3.9}$$

For any angle 0<θ<90, the corresponding projection, w', onto the cone at radius r' is given by

$$w' = 2r'\tan\left(\frac{\theta}{2}\right) \tag{3.10}$$

combining equations (3.9) and (3.10) and simplifying produces equation (3.8). The vertical projection of the target onto the cone, h', is given by

$$h' = \frac{H * h_c}{R * 2\tan\left(\frac{FOV}{2}\right)} \tag{3.11}$$

where H is the actual height of the target. Here, it is assumed that the camera to mirror distance is negligible compared to the target range, R. It is evident that the projected height is also inversely proportional to R.

The short analysis of the 360 profiling system points out some key aspects. These key points will shape the feature extraction and classification process. The first point to note is that images of targets will be blurred, and, in the horizontal direction, images will be demagnified due to the mirror. An image of the raw sensor output for a 360 profiling system is shown in Figure 3-6. This means that detailed images of targets will not be generated and the shape of overall shape of targets will be altered. It is however noted that all targets will be subject to the same magnification and thus the relative width of target will be preserved. The relative height of the target is also preserved in the image plane. Though target image height and width are scaled by different constants, it was

found that both are inversely proportional to the target range. Therefore the HWR of a target remains constant at various ranges and may potentially be used as a feature for classification. This result, however, is based on observing a target at given radius, r', or vertical location, on the mirror. Targets which traverse along the vertical axis may produce a varying HWR. For example, observing the same target traversing a steep hill may produce a varying HWR. Limiting the system's vertical field of view to a small range for an expected target distance, limits the variation of the target's maximum HWR.

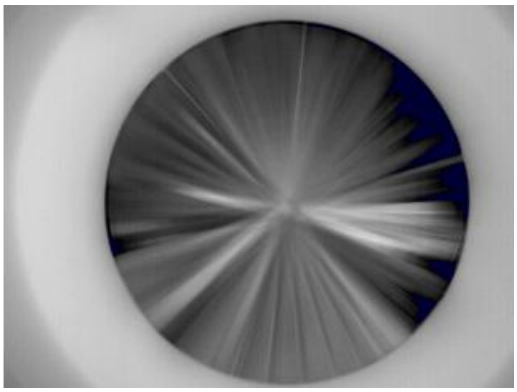


Fig. 3-6. An image of the sensor output for a 360 profiling system

3.3 Image Segmentation

The feature selection used in determining classifications drives the signal conditioning process. Because the HWR is to be used as a feature for classification, binary profiles of targets are sufficient and also convenient. The data is collected with a LWIR camera. LWIR cameras typically output 8 bit data, but many allow the user to output 14 or 16 bit data. The data must be reduced from the gray scale imagery to thresholded binary imagery. A typical method for background segmentation is to use

statistics (mean and standard deviation) of the imagery to determine foreground or target from background. Determining these statistics accurately requires a number of samples. In this case, each $frame_k$, provides a sample for $pixel_{i,j}$ and the mean and standard deviation must be determined for each pixel imaging the cone. Keep in mind that these algorithms are to be run on a small, low power system. Typical camera focal plane array dimensions are 320x240 or 640x480. Storing 30 frames of 640x480 data at 16 bits can be burdensome for a low power processor. Processing excessively large amounts of data may require more processing time and may consume more power. The intent of the profiling sensor, however, is to be low power, and in many cases, real time.

The background segmentation algorithm should be robust enough to handle changing background but still meet system requirements. The memory capability of the system is not known, but it is assumed that the memory storage resource is limited. For this reason an online mean and standard deviation or "running statistics" were evaluated. Rather than storing a window of N frames of data to generate a mean and standard deviation at the N+1 frame, running statistics allows the N+1 mean and standard deviation to be calculated from the current frame, the previously calculated mean and standard deviation and the value N. For example, if the user wishes to generate statistics using the last 100 frames, then a 100 frame window of data must be stored in memory. Running statistics requires only 3 frames and the value N to calculate the next mean and standard deviation at frame 101. The mean of a pixel of the current can be found by

$$mean = \frac{N-1}{N} mean_{prev} + \frac{1}{N} value_{current} \tag{3.12}$$

where $mean_{prev}$ is the mean calculated for the previous frame, $value_{current}$ is the current value of the pixel, and N is the number of frames in the window. To determine the standard deviation of a pixel in the current frame, the following equation can be used:

$$standard\ deviation = [\frac{N-1}{N} std_{prev}^2 + \frac{1}{N-1}(value_{current} - mean)^2]^{1/2} \quad (3.13)$$

Here, std_{prev} is the standard deviation calculated at the previous frame. It should be noted that this method does not use a sliding window. The sliding window allows the algorithm to handle dynamic backgrounds. The running statistics window is ever increasing from the starting location, which causes the statistics to be less susceptible to change as time passes. To make the running statistics method more accommodating for changing backgrounds, two staggered running statistics are simultaneously generated for a limited maximum window and repeated. To simply repeat a single running statistic for a given maximum window length would not be sufficient. The running statistics needs to evaluate a number of frames before the statistics are representative of the true distribution. By alternating, one running statistic, rs1, can be used for evaluation, while the other, rs2, builds a history. When the maximum window length for rs1 is reached, rs1 starts a new window. At the same time, rs2 is then used for evaluation, while rs1 is now building history. The improved alternating method would then require 5 frames of history and the values N_1 and N_2 for the respective windows.

By generating statistics, a model of the background can be formed. The random background was approximated as following a Gaussian distribution, such that a measured signal, greater than 3 standard deviations, can be regarded as foreground or target signal.

But a direct application of the methods described will produce poor results. This is because the measured data may include both target and background signal, while the previous analysis assumes that the statistics model the distribution of the background. Again, it is therefore necessary to remove target signal from the background model. To do this, the dual running statistics were processed at each frame. Values of signal which were above the threshold were replaced by Gaussian random numbers with the statistical properties of the background. These replacements were for statistical calculations of the mean and variance only. By replacing these extreme values, the algorithm loses some of its versatility and becomes more resistant to change. As a result, target or "on" pixels produced from a quickly changing background, will always be replaced statistically, and therefore never become background or turn "off." A fix to this situation is to keep track of the number of consecutive frames a pixel has been "on." When the number of consecutive "on" frames for a pixel reaches a given "consecutive on" threshold, the value is no longer replaced statistically. For every frame after, the pixel value will be considered in the statistics until the statistics reflect the pixel value. At this point the pixel value will be part of the background model, and the pixel will turn "off" provided the updated values have leveled at some consistent mean and standard deviation.

It is also necessary provide a catch in the event of a non-uniformity correction or some unknown event in which the output values of the majority of the pixels suddenly change by an extreme value. In such an event, the majority of the pixels will be "on." In this case, a threshold can be set for the number of "on" pixels, such as 1/8 of the cone imaging window. Recall, "on" pixels should represent targets. In our scenario, it was not expected to have a large number of targets which filled 1/8 of the imaging window, so

this value was more than appropriate. If more than 1/8 or the set threshold of the pixels are "on," then the running statistics should reset and reinitialize. At the time of reset, if a target is moving in the scene, a ghost image will be created at the location the target occupied when the reset occurred. This ghost image will last until the "on" pixels reach a "consecutive on" count greater than the set threshold, and will then be assimilated into the background. If the application simply calls for detecting targets of interest and providing an alert, then this event may not be an issue. If, however, the ghosting effect is to be avoided, a period of frames must be analyzed statistically before statistical replacement is to be applied. This can be thought of as an initialization stage. Consequently, if the target remains stationary during the entire initialization stage and then proceeds to move, ghosting will still occur. These are of course special scenarios, but they must be taken into account when considering potential applications.

3.4 Target Detection and Feature Extraction

The process of background subtraction conditions the digital sensor output into a binary image. The binary image is then analyzed for potential targets. The binary image contains potential targets, if they are present in the scene, and also noise. The targets must be distinguished from the noise in order to extract useful features for classification. Because of the initial steps taken to determine an appropriate background model, it is assumed that noise which passes into the binary image will be sparse in nature. By using a morphological filter and grouping algorithm, isolated pixels can be removed while groups of connected pixels can be recognized as objects. These objects can further be filtered by considering only objects above a size threshold in pixels. A large portion of

the noise can be removed by these simple techniques. The remaining objects are investigated as potential targets. The HWR ratio feature requires the extraction of object's height and width. For each object, each pixel of the object is analyzed for its radial distance and angle from the center of the cone. The pixels with the maximum and minimum radial distance are determined, and the height is measured as

$$height = radial\ distance_{max} - radial\ distance_{min} \tag{3.14}$$

Similarly, the pixels of an object which contain the maximum and minimum angle are found. Let the angle φ be the difference between the maximum and the minimum angle. The midpoint of the target is the minimum radial distance plus the height/2. The width of the object is then measured as the arc across φ at the midpoint such that

$$width = midpoint * \varphi \tag{3.15}$$

$$midpoint = radial\ distance_{minimum} + \frac{height}{2} \tag{3.16}$$

$$\varphi = angle_{maximum} - angle_{minimum} \tag{3.17}$$

In this fashion, the height and width can be measured for each object to produce a HWR.

For accurate classifications, accurate feature extractions must be performed. Objects greater than the size threshold are investigated for the HWR feature. These objects, however, still may not be actual targets. The detection process should remove

non target data from the classification process. In order to improve detection rates, targets are tracked at each frame. The reasoning for tracking objects is that it is expected that noise objects are random and will not persist in the same location for multiple frames or traverse a path in angle. The midpoint of the object in angle, $midpoint_{\varphi}$, is referenced to the recorded $midpoints_{\varphi}$ for all objects in the previous frame. An object within a given threshold angle of an object in the previous frame is considered to be the same object. The history associated with the matched previous object is transferred to the matched current object. Objects which persist or have a history for a persistence threshold number of frames are considered to be targets. Two targets being tracked are shown in Figure 3-7.

Fig. 3-7. Two objects, a human leading a horse, are shown during the tracking procedure. The radial lines have been added to highlight the tracking information.

Target signal may fluctuate from frame to frame. These fluctuations may cause the measured HWR for a target to also vary from frame to frame. Over multiple frames,

the HWR for a target will converge to an average value. For this reason, the HWR of a target is averaged for a given number of frames determined by a residual threshold. The average HWR is used as a feature for classification. The feature cannot be extracted unless a target persists for a residual threshold number of frames, which reinforces detection as being the event an object persists for a residual number of frames. In regions of low contrast or when targets pass behind obstructions, the target signal can fluctuate to the point it is no longer considered an object of interest. At such a point, the consecutive frame tracking would cease and begin anew when the target signal became visible again. For this reason the definition of detection can be expanded to tracking a target for a residual threshold number of frames within an S frame window. The S frame window reflects the system requirements that a target is to be classified within a given time frame. For real time systems, threats need to be identified quickly, so that further action can be taken. If the threat is detected and classified at any point within the S frame window, an alert can be provided. Multiple alerts for the same threatening target would not be reasonable, but a single alert within a limited time frame is sufficient.

3.5 Classification

In the event a detection is made, the averaged HWR of the target is used as a single feature for classification. The classification algorithm choice may vary. The Mahalanobis distance can be used to discriminate between classes with low computational complexity. The Mahalanobis distance can be defined as

$$Mahalanobis\ distance(\boldsymbol{x}) = \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^T Cov^{-1}(\boldsymbol{x}-\boldsymbol{\mu}} \qquad (3.18)$$

where **x** is the measured feature vector, **μ** is the class mean vector, and Cov is the class covariance matrix. For a single feature, the Mahalanobis distance reduces to

$$Mahalanobis\ Distance\ (x) = \sqrt{\frac{(x-\mu)^2}{\sigma}} \tag{3.19}$$

When the class distribution is assumed Gaussian, and class priors are equally likely, the Mahalanobis distance is similar to the Naïve Bayesian classifier for a single feature.

3.6 Results

The results presented here are for a 360 profiling system which uses a Flir photon 640 (640x512) LWIR camera, 25mm lens, and a conical mirror with a cone angle of 95.7 degrees and a base diameter of 16.65cm. The LWIR video was collected at 30 frames per second and consisted of targets walking along arcs around the sensor at 30 meters. Each target set consisted of a human leading a horse. There were a total of 5 target sets used in this analysis. The window length of the running statistics algorithm was 150 frames. Two alternating windows were used and target signal was distinguished as being greater than 2.5 standard deviations from the background mean. In the binary image, grouped objects were only considered valid if they were above a size threshold of 20 pixels. A valid object was allowed ± 3 degrees of movement in consecutive frames in order to be considered the same object during the tracking process. Targets which persisted for 5 or more frames were classified using the average HWR for the last 5 frames. For now, we will consider the detection event as a target persisting for 5 frames. The parameters are listed in Table 3-1.

Table 3-1. Parameter values used to generate results

Parameters	***Values***
Camera	Flir Photon 640
Lens	25mm
Frame Rate	30 fps
Mirror	Conical
Cone Angle	95.7 degrees
Base Diameter	16.65cm
Statistical Window	150 frames
Binary Threshold	2.5 standard deviations
Size Threshold	20 pixels
Tracking Threshold	±3 degrees
Persistence Threshold	5 frames

The classification results are presented for objects which were correctly detected. Using the definition of conditional probabilities, the intersection of correct classification and correct detection is given by the following equation:

$$P(C \cap D) = P(C|D)P(D) \tag{3.20}$$

where C and D are the events of correct classification and correct detection , respectively. The probability of detection is given by

$$P(D) = \frac{Number\ of\ Correct\ Detections}{Total\ Number\ of\ Detections} = \frac{4560}{4718} = 96.65\% \tag{3.21}$$

The confusion matrix for correct classifications given correct detections is shown in Table 3-2. The overall probability of correct classification given correct detection is

84.21%, providing for the probability of correct classification and correct detection to be 81.39%.

Table 3-2. Confusion matrix of correct classifications given correct detections

		Correctly Detected Objects	
		Human	**Animal**
(C|D)	**Human**	2266 (91.15%)	500 (24.11%)
	Animal	220 (8.85%)	1574 (75.89%)
	Total # of Correctly Detected Objects	2486	2074

3.7 Discussion

The overall probability of correct classification and correct detection result of 81.39% is low from a system analysis perspective. The low result can be attributed to the collected data used in the analysis. Because humans were leading horses in the data collection, it often occurred that the humans and animals overlapped. The nature of the mirror also reduces the apparent distances between targets as the target range increases. For the classification results shown, at a given frame, objects which overlapped were ignored and only objects which could be distinguished either human only or animal only were evaluated. Objects, however, were tracked over multiple frames and the object's history was used to calculate average HWR feature. The history often contained instances when the object was overlapped with another object. In these cases, the classification results may not be completely accurate. A remedy for this scenario is to create an

additional class, *other*, which represents targets other than human only or animal only. Because detected objects could have been human, animal, or a combination of both, three distinct classes are more appropriate. All 2-D imaging systems which use a single HWR feature will suffer from this problem of target overlap. The conical mirror, however, increases the effect at longer ranges. A classification algorithm, which uses each pixel as a feature, for an increased feature space dimension may perform better, but at the cost of computational complexity.

In spite of the low overall performance, it should be noted that the probability of correctly classifying humans given a correct detection was reasonably high. This implies that humans were almost always correctly classified with few misses. The system suffered from false alarms caused by misclassified animals. In an environment where humans are more likely to occur than wild animals such as horses or cows, this level of performance might be acceptable. It was also found that when a detection was considered as a target persisting for 5 frames within a 45 frame window, the human and animal target were always detected.

3.8 Conclusion

The 360 profiling system, accompanied by the signal condition and feature extraction techniques presented in this paper, show promise. Though the results leave room for improvement, it should be noted that the data presented a complex scenario for an initial system analysis. Data which contained only human or only animal data would have been more appropriate. Field collections, however, are difficult and expensive to perform. More complex tracking and classifying algorithms can be used to improve

results, but may introduce increased power consumption and processing time. Results can also be improved by expanding the two class problem to a three class problem to reflect the complexity of the data. Though the techniques presented here can certainly be modified, a road map for system construction, feature extraction, and result analysis of a 360 profiling system have been presented here.

4 Conclusion and Future Work

4.1. Conclusion

This thesis has discussed the signal conditioning and feature extraction methods that can be used for a variety of profiling sensors. The methods combine a balance of efficiency and effectiveness. Modifications to these algorithms and optimization of parameters can lead to improved performance. This paper however discusses a framework of methods which can be combined to produce valid results which have been demonstrated successfully. These techniques collect the output produced by the sensor system hardware and provide meaningful and reliable information to the user. The tilted array velocity estimate technique presented in this paper ultimately provides the user with more reliable information without the need for an increased amount of hardware. The two column array technique, once again, allows for reliable user information. When applied to focal plane arrays, the two column approach furnishes a way to significantly reduce the amount of information needed for processing, yet produce accurate broad scale classification. The 360 profiling system algorithm yields a framework of routines which have been shown to validate the system as a whole, which in turn, leads the way for

future development of such systems. In short, this thesis has delivered a way in which multiple profiling sensors can be made useful and effective.

4.2. Future Work

Future work relating to profile sensors would include demonstrating classifications with the pyroelectric array in real time and optimizing the 360 profiling system. The optimization includes the collection of more data, testing, and modification of the algorithm. Improving the tracking methods to more intelligently keep track of targets could potentially provide some improvement. Also, introducing the class for multiple targets overlapped or perhaps vehicles and performing classifications may be an avenue for exploration. The addition of new classes may require the addition of new features. Another modification to the algorithm process is to use the concentric ring approach presented by Brown *et al* [11]. The concentric ring method may allow a reduction in the pixels needed to perform classifications.

By having an optimized algorithm, hardware reduction can be evaluated. It may be found that the same level of performance can be achieved when the pixel count or the frame rates of the camera are reduced. These reductions could reduce power consumption and increase cost savings, making the 360 profiling sensor a more viable option.

References

1. R. B. Sartain, “Profiling sensor for ISR applications,” Proc. SPIE 6963, (2008).

2. K.K. Klett, R. Sartain, T. Alexander, and K. Aliberti, “Optical and radiometry analysis for a passive infrared sparse sensor detection system,” Proc. SPIE 6941, 694101, (2008).

3. A. L. Robinson, C. E. Halford, E. Perry, and T. Wyatt, “Sparse detector sensor model,” Proc. SPIE 6963, 69630L, (2008).

4. D. J. Russomanno, M. Yeasin, E. L. Jacobs, M. Smith, and M. S. Sorrower, “Sparse detector sensor: profiling experiments for broad scale classification,” Proc. SPIE 6941 6963, pp. 69630M-69630M-11, (2008).

5. D. J. Russomanno, S. K. Chari, and C. E. Halford, “Sparse detector imaging sensor with two-class silhouette classification,” *Sensors*, 8(12), 7996-8015, (2008).

6. K. Emmanuel, D. J. Russomanno, E. L. Jacobs, S. K. Chari, and J. B. Brown, “Silhouette Data Acquisition for a Sparse Detector Sensor,” in *Proceedings of Military Sensing Symposia Passive Sensors*, SENSIAC, (2009).

7. S. K. Chari, C. E. Halford, E. L. Jacobs, F. A. Smith, J. B. Brown, and D. J. Russomanno, “Classification of humans and animals using an infrared profiling sensor,” Proc. SPIE 7333, (2009).

8. J. B. Brown, S. K. Chari, and C. E. Halford, “Assessment of a linear pyroelectric array sensor for profile classification,” Proc. SPIE 7693, (2010).

9. S. K. Chari, F. A. Smith, C. E. Halford, E. L. Jacobs, and J. M. Brooks, “Range and velocity independent classification of humans and animals using a profiling sensor,” Proc. SPIE 7694, (2010).

10. E. White, J. B. Brown, S. K. Chari, and E. L. Jacobs, "Real-time assessment of a linear pyroelectric array for sensor classification," Proc. SPIE 7834, (2010).

11. J. B. Brown, S. K. Chari, and E. L. Jacobs, “Signature profiling: a signal processing technique for broad scale classification,” in *Proceedings of Military Sensing Symposia Battlespace Acoustic & Seismic Sensing*, SENSIAC, BP14, (2010).

12. S. Baker and S. K. Nayer, "Single Viewpoint Catadioptric Cameras," in *Panoramic Imaging: Sensors, Theory, and Applications,"* R. Benosman and S. B. Kang, eds. (Springer-Verlag, 2001), pp. 39-70.

13. Y. Yagi, "Omnidirectional sensing and its applications," IEICE Trans. Inf. & Syst. E82-D, 3 (1999).

14. T. E. Boult, X. Gao, R. Michaels, and M. Eckmann, "Omni-directional visual surveillance," Imaging and Vision Computing 22, 515-534 (2004).

15. R. Driggers, P. Cox, and T. Edwards, *Introduction to Infrared and Electro-Optical Systems* (Artech House, 1999).

16. E. L. Jacobs, S. K. Chari, C. E. Halford, and H. McClellan, "Pyroelectric sensors and classification algorithms for border/ perimeter security," Proc. SPIE 7481, 74810P, (2009).

17. http://www.dias-infrared.com/infrared-detector.php .

18. https://www.scd.co.il/dynamic-web.aspx?page_id=38&parent_id=15&pgnm=%C2%BB%20LWIR%20%E2%80%93%20Uncooled.

19. http://www.sofradir-ec.com/products-uncooled.asp.

20. J. B. Calvert, "Spherical mirrors," http://mysite.du.edu/~jcalvert/optics/mirror.htm.

21. F. A. Jenkins and H. E. White, *Fundamentals of Optics* (McGraw-Hill Book Company, 1950).

Appendix

A.1. Tilted Array Matlab Code

```
[Mvid,Nvid,Color,FrameTotal]=size(vid);
n=200;%number of sensors
startHorLoc=430;%starting location of linear array
startVerLoc=260;%
HorInc=1;%increment distance of each sensor in pixels
VerInc=1;
for i=1:1:n
sLoc(i,1)=(startVerLoc-VerInc)+((VerInc*i));  %vertical
sLoc(i,2)=(startHorLoc-HorInc)+(slant*(HorInc*i));  %horizontal
end
%Creates distance matrix. Top triangle is row(vertical distance.
%Bottom triangle is col(horizontal distance). diagonal is 0. (no distance b/n
%same sensor)
[MsLoc,NsLoc]=size(sLoc);
for i=1:1:MsLoc
for j=1:1:MsLoc
if i<j%vertical dist. top half
disMat(i,j)=abs(sLoc(i,1)-sLoc(j,1));
elseif i>j%horizontal dist. bottom half
disMat(i,j)=abs(sLoc(i,2)-sLoc(j,2));
else
```

```
disMat(i,j)=0;%i==j
end
end
end
for i=1:1:FrameTotal/scale
frame=ceil(i*scale);
for j=1:1:n
vert=sLoc(j,1);
hor=sLoc(j,2);
out(j,count(j))=sum(sum(sensor(:,:).*(double(rgb2gray(vid(vert-(4):vert+(4),hor-
(4):hor+(4),:,frame))))));
outUncensored(j,count(j))=double(rgb2gray(vid(vert,hor,:,frame)));
count(j)=count(j)+1;
end
end
%%%%%%%   CROPPING   %%%%%%%%%%%%%%%%%%%%
signalHor=sum(out);
meanSigHor=mean(signalHor);
stdevSigHor=std(signalHor);
numdev11=1;%number of standard deviations on first pass
numdev12=.3;%number of standard deviations on second pass
[sigHorLowInd]=find((signalHor<(meanSigHor+(stdevSigHor*numdev11))));
meanSigHorLow=mean(signalHor(sigHorLowInd));
```

```
stdevSigHorLow=std(signalHor(sigHorLowInd));
[sigHorHighInd]=find((signalHor>(meanSigHorLow+(stdevSigHorLow*numdev12))));
%this portion finds regions of high signal that have breaks less than or equal to the size of gap. It outputs these regions in cropdata, where row1 are start locations and row2 are stop locations of segments. Then find the largests segment and treat that as the signal.
gap=10;
region=1;
cropdata(1,region)=sigHorHighInd(1);
for i=2:1:length(sigHorHighInd)
if sigHorHighInd(i)-sigHorHighInd(i-1)>gap
cropdata(2,region)=sigHorHighInd(i-1);
region=region+1;
cropdata(1,region)=sigHorHighInd(i);
end
end
cropdata(2,region)=sigHorHighInd(length(sigHorHighInd));
cropdata(3,:)=cropdata(2,:)-cropdata(1,:);
indices= cropdata(3,:)==max(cropdata(3,:));
startHor=cropdata(1,indices);
stopHor=cropdata(2,indices);
[Mout,Nout]=size(out);
%%%%%%%%MakeBinary%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:Mout
```

```
numdev21=5;
numdev22=5;
last=floor((FrameTotal/scale));
if startHor>1    %evaluates region outside of target
if stopHor<Nout
stdOut(i,:)=std([out(i,1:startHor) out(i,stopHor:Nout)]);
meanOut(i,:)=mean([out(i,1:startHor) out(i,stopHor:Nout)]);
else
stdOut(i,:)=std([out(i,1:startHor)]);
meanOut(i,:)=mean([out(i,1:startHor)]);
end
else
stdOut(i,:)=std([out(i,startHor:Nout)]);
meanOut(i,:)=mean([out(i,startHor:Nout)]);
end
outHiOrig=out(i,:)>meanOut(i,:)+(numdev21).*stdOut(i,:);
outLoOrig=out(i,:)<meanOut(i,:)-(numdev21).*stdOut(i,:);
outHiVel=out(i,:)>meanOut(i,:)+(numdev22).*stdOut(i,:);
outLoVel=out(i,:)<meanOut(i,:)-(numdev22).*stdOut(i,:);
outVel(i,Nout)=0;
outOrig(i,Nout)=0;
outOrig(i,outHiOrig)=1;
outOrig(i,outLoOrig)=1;
```

```
outVel(i,outHiVel)=1;
outVel(i,outLoVel)=1;
end
%creates two binary images. One highly filtered (outVel) to use for
%velocity calculations, and one (outOrig)less filtered to use locate target for
%cropping vertically. outOrig is first filtered for excess noise with
%bwmorph.
%%%%%%%%%
outOrigFiltered=  bwmorph(outOrig,'majority',1);
out =  bwmorph(outVel,'majority',3);
Uses previously gained 'width' dimensions to adjust height sample
%space. It only sums b/n 'startHor and stopHor'
%Repeats previous horizontal cropping method in the vertical direction
signalVer2=sum(outOrigFiltered(:,startHor:stopHor),2);
sigVerHighInd2=find(signalVer2);
gap=4;
region=1;
cropdata3(1,region)=sigVerHighInd2(1);
for i=2:1:length(sigVerHighInd2)
if sigVerHighInd2(i)-sigVerHighInd2(i-1)>gap
cropdata3(2,region)=sigVerHighInd2(i-1);
region=region+1;
cropdata3(1,region)=sigVerHighInd2(i);
```

```
end
end
cropdata3(2,region)=sigVerHighInd2(length(sigVerHighInd2));
cropdata3(3,:)=cropdata3(2,:)-cropdata3(1,:);
indices3= cropdata3(3,:)==max(cropdata3(3,:));
startVer2=cropdata3(1,indices3);
stopVer2=cropdata3(2,indices3);
%%%%%%%%%%%%%%%%Visualation & Display%%%%%%%%%%%%%%
Cropped2=zeros(size(outOrig));
Cropped2(startVer2:stopVer2,startHor:stopHor)=outOrig(startVer2:stopVer2,startHor:sto
pHor);
Cropped3=Cropped2;
%%%Creates window to box in Target%%%%%
Cropped3(startVer2,startHor:stopHor)=.5+Cropped3(startVer2,startHor:stopHor)-
.5*Cropped2(startVer2,startHor:stopHor);
Cropped3(stopVer2,startHor:stopHor)=.5+Cropped3(stopVer2,startHor:stopHor)-
.5*Cropped2(stopVer2,startHor:stopHor);
Cropped3(startVer2+1:stopVer2-1,startHor)=.5+Cropped3(startVer2+1:stopVer2-
1,startHor)-.5*Cropped2(startVer2+1:stopVer2-1,startHor);
Cropped3(startVer2+1:stopVer2-1,stopHor)=.5+Cropped3(startVer2+1:stopVer2-
1,stopHor)-.5*Cropped2(startVer2+1:stopVer2-1,stopHor);
%%%%%%%%%%Adjust Speed based on Crop%%%%%%%%%%%%%%%
blankPage=zeros(size(out));
```

```
blankPage(startVer2:stopVer2,startHor:stopHor)=out(startVer2:stopVer2,startHor:stopH
or);
Newout=mat2gray(blankPage);
%%%%%%%%%%%%Tilt Adjust & Speed%%%%%%%%%%%%%%%%%%
speedmax=20;%pixels/frame
[Mout,Nout]=size(Newout);
[Mfirst,Nfirst]=find(Newout,1,'first');
[Mlast,Nlast]=find(Newout,1,'last');
%Time Shifts output to account for sensor spacing in horizontal axis.
%finds first target hit and time taken for each  additional first hit on other
%sensors. then evaluates speeds between first hit sensor and additional
%first hits on additional sensors. The speeds are averaged to find an
%overall target speed. This is used to shift data of each sensor in time to
%simulate an actual linear array.
for i=1:1:MsLoc
firstOneInColForRowi=find(Newout(i,:),1,'first');
lastOneInColForRowi=find(Newout(i,:),1,'last');
if isempty(firstOneInColForRowi)||isempty(lastOneInColForRowi);
firstLastInRowTable(1,i)=0;
firstLastInRowTable(2,i)=0;
else
firstLastInRowTable(1,i)=firstOneInColForRowi;
firstLastInRowTable(2,i)=lastOneInColForRowi;
```

```
end
for j=1:1:MsLoc
%Develops a matrix of speeds from sensor(i) to sensor(j) This gets a better
%average to account for phase differences from one loc to another.
firstOneInColForRowj=find(Newout(j,:),1,'first');
lastOneInColForRowj=find(Newout(j,:),1,'last');
if isempty(firstOneInColForRowj)||isempty(firstOneInColForRowi)
frameTimeCol(i,j)=0;
else
frameTimeCol(i,j)=abs(firstOneInColForRowj-firstOneInColForRowi);
end
if frameTimeCol(i,j)==0
speedCol(i,j)=0;%eliminates dividing by zero error
elseif j>i %keeps in the bottom half of distance matrix
speedCol(i,j)=(disMat(j,i))./frameTimeCol(i,j);
if speedCol(i,j)>speedmax
speedCol(i,j)=0;
end
else
speedCol(i,j)=(disMat(i,j))./(frameTimeCol(i,j));
if speedCol(i,j)>speedmax
speedCol(i,j)=0;
end
```

```
end
%%%%%%%%%%%%%%trailing back edge%%%%%%%%%
if isempty(lastOneInColForRowj)||isempty(lastOneInColForRowi)
frameTimeCol2(i,j)=0;
else
frameTimeCol2(i,j)=abs(lastOneInColForRowj-lastOneInColForRowi);
end
if frameTimeCol2(i,j)==0
speedCol2(i,j)=0;%eliminates dividing by zero error
elseif j>i %keeps in the bottom half of distance matrix
speedCol2(i,j)=(disMat(j,i))./frameTimeCol2(i,j);
if speedCol2(i,j)>speedmax
speedCol2(i,j)=0;
end
else
speedCol2(i,j)=(disMat(i,j))./(frameTimeCol2(i,j));
if speedCol2(i,j)>speedmax
speedCol2(i,j)=0;
end
end
end
end
%Creates window around each point to make velocity comparisons to.
```

```
windowSize=20;
for i=1:1:Mout%MsLoc
if (i-floor(windowSize/2))<1
upperLimit=1;
else
upperLimit=i-floor(windowSize/2);
end
if (i+floor(windowSize/2))>Mout%MsLoc
lowerLimit=Mout;
else
lowerLimit=i+floor(windowSize/2);
end
speedCol12(i,1:(lowerLimit-upperLimit+1))=speedCol(i,upperLimit:lowerLimit);
speedCol22(i,1:(lowerLimit-upperLimit+1))=speedCol2(i,upperLimit:lowerLimit);
end
speedCol=speedCol12;
speedCol2=speedCol22;
[validNonZeroSpeeds dummy] = size(find(speedCol>0));
[validNonZeroSpeeds2 dummy2] = size(find(speedCol2>0));
avgSpeed1=sum(sum(speedCol))./validNonZeroSpeeds; %MsLoc uses all sensor, even
those w/ zero speed.
avgSpeed2=sum(sum(speedCol2))./validNonZeroSpeeds2;
height=stopVer2-startVer2+1;
```

```
Width=max(sum(Newout,2));
ratioHW=height/Width;
avgSpeed=(avgSpeed1+avgSpeed2)./2;
```

A.2. 360 Profiling System Matlab Code

A.2.1. Video Analysis

```
%%%%%%%Set Variables to Control Thresholding%%%%%%%%%%
scale=2.5% # of standard deviations away from mean to use as threshold
window=75%active amount of frames to be displayed using previous window
%amount of frames(not displayed) as starting statistics
%%%%%%%Set Variables for .raw file type%%%%%%%%%%%%%%
frameSize=[512 644]%[256 324] or [512 644]
pixelbit=16 % 8 or 16
%%%%%%%%Set Variable for Statistical Replacement%%%%%%%%
SRthreshhold=100%the number of time a pixel can be consecutively replaced
%with background statistics
%%%%%%%Set Variables for Tracking Options%%%%%%%%%%%%%%%
residualTime=10;%The number of frames a target must remain in the scene to be
considered a threat
xtraAngle=3;%the amount of angle a target can move from frame to frame and
%still be considered the same target
```

```
%%%%%%%Set Variables to determine Classifications%%%%%%%%%%%%%%%
groupingThreshold=20;%Ignores pixel groups below this size
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_frames = getrawframecount(fileAddress, frameSize,pixelbit); %count the number of frames in the video
startFrame=500;
endFrame=1500;%round(num_frames*(1/3));
framesRead=endFrame-startFrame+1;
allframes = readraw(fileAddress,frameSize,[startFrame endFrame],pixelbit); %read as many frames as needed
img1 = allframes(:,:,1); %get reference frame to find mirror location
% figure(10),imshow(img1-allframes(:,:,framesRead),[]);%display reference frame
% title('select center, then select point on circumference');
%
% [x,y] = ginput(2); %click the on two points, first point is the center and the other is any point on the circumference
% centerxcol = x(1);%Ncol
% centeryrow = y(1);%Mrow
% rad_cir = sqrt((x(2) - centerxcol)^2 + (y(2) - centeryrow)^2 );
D = distance1(frameSize(1),frameSize(2),centeryrow,centerxcol);%creates a distance matirx from center of cone
mask = D<rad_cir;%creates logical mask of cone
```

```
fileType2='.avi';
saveVideoFileName=[fileName fileType2];
out=avifile(saveVideoFileName,'compression','Cinepak','fps',20);
% % data = [zeros(64,pixels-1);data];
if(length(out.currentState) == 4)
out = close(out);          %save and close the video
end
out=avifile(saveVideoFileName,'compression','Cinepak','fps',20);
%%%%%%%%%%%%Initialize Variable for Residual Group Tracking%%%%%
ccResidual.NumObjects=0;%initialize variable
%%%%%%%%%%%%%%Initialize Variables for Statistical Thresholding%%%
%variables for first statistics window
count=1;
mu=0;
stdev=0;
sumMinus=0;%the sum of terms 1 to n-1 used to determine nth mean & std
%variables for second statistics window
count2=1;
mu2=0;
stdev2=0;
sumMinus2=0;%the sum of terms 1 to n-1 used to determine nth mean & std
display=1;%determines which window is displayed
SRCollection=ones(frameSize);%initialize the matrix for counting
```

```
% consecutive on pixels. If thes values are higher than a threshold the
% pixel should be considered into the statistics for the background. This
% is so that if a target stops, it will eventually be considered part of
% the background.
SRcount=1;%Keeps count of frames that used Stat BG Replacement Method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for ii=1:1:framesRead
imgnow = double(allframes(:,:,ii));%gets current frame
%%%%%%%%%%%%%%%%%%%%
% This section replaces 'target' pixels with random numbers that resemble
% the background for statistical calculations. This is so the target data
% does not corrupt the background data. If a pixel is considered target consecutively
% for longer than SRthreshhold, than the pixel is used in background
% statistics. Also if the camera NUCs and more than 1/4 of the frame is on,
% the system resets the calculation process.
if SRcount==2*window
meanFirstWindow=mu;%Gets the mean of the first full window
stdFirstWindow=stdev;%Gets the std of the first full window
end
if SRcount==4*window
meanSecWindow=mu;%Gets the mean of the second full window
stdSecWindow=stdev;%Gets the std of the second full window
meanWin=(meanFirstWindow+meanSecWindow)./2;%avgs 1st & 2nd mean
```

```
stdWin=(stdFirstWindow+stdSecWindow)./2;%avgs 1st & 2nd std
end
if (SRcount>4*window)&&(SRcount<=5*window)
replaceStatLoc = (imgnow>meanWin+(scale).*stdWin);
%Uses the mean & std of the first 2 full (4)windows to replace
%target for statistical calculations
SRCollection=SRCollection + replaceStatLoc;
%adds the current frame of replacement pixels to the count of
%previous frames of replacement pixels
SRCollection(logical(1-replaceStatLoc))=1;
%(1-replaceStatLoc) inverts repStatLoc so that 1=leave & 0=replace
%wherever the pixels were to  be left(were considered background)
%then the consecutive 'on' count for that pixel resets b/c this frame
%it is 'off'
replaceStatLoc(SRCollection>SRthreshhold)=0;
%If the count of any pixel in SRCollection(the sum of consecutive
%time the pixel was target) is greater than SRthreshhold, then
%don't replace it statistically. Leave it as the original value.
%This is so moving objects which stop in the frame won't stay
%forever and will become background
randomNums=randn(frameSize);%generates random numbers for a frame
replaceStat=meanWin + stdWin.*randomNums;%create random numbers
%with the statistics of the background
```

```
statImag=imgnow;%Creates an image used for statistical calculations
%from original image
statImag(replaceStatLoc)=0;%removes locations from original image
%(statistical image) that need to be replaced
statImag=statImag+(replaceStatLoc.*replaceStat);
%In the locations where the original image needs to be replaced,
%random numbers with the statistics of the background are used
end
if SRcount>5*window
randomNums=randn(frameSize);
if display==1
replaceStatLoc = (imgnow>mu+(scale).*stdev);
replaceStat=mu + stdev.*randomNums;
else
replaceStatLoc = (imgnow>mu2+(scale).*stdev2);
replaceStat=mu2 + stdev2.*randomNums;
end
SRCollection=SRCollection + replaceStatLoc;
%adds the current frame of replacement pixels to the count of
%previous frames of replacement pixels
SRCollection(logical(1-replaceStatLoc))=1;
%(1-replaceStatLoc) inverts repStatLoc so that 1=leave &    0=replace
%wherever the pixels were to  be left(were considered background)
```

```
%then the consecutive 'on' count for that pixel resets b/c this frame
%it is 'off'
replaceStatLoc(SRCollection>SRthreshhold)=0;
%If the count of any pixel in SRCollection(the sum of consecutive
%time the pixel was target) is greater than SRthreshhold, then
%don't replace it statistically. Leave it as the original value.
%This is so moving objects which stop in the frame won't stay
%forever and will become background
statImag=imgnow;%Creates an image used for statistical calculations
%from original image
statImag(replaceStatLoc)=0;%removes locations from original image
%(statistical image) that need to be replaced
statImag=statImag+(replaceStatLoc.*replaceStat);
%In the locations where the original image needs to be replaced,
%random numbers with the statistics of the background are used
end
if SRcount<5*window
statImag=imgnow;%initializes statImag when there is NO replacement
end
if SRcount>(4*window)
if sum(sum(replaceStatLoc))>((frameSize(1)*frameSize(2))/8)
%If a cloud passes over or if the camera is NUC and a large num of %pixels are outside
of the threshhold than the process is started
```

```
%over if the number of on pixels is greater than 1/8 of the frame
SRcount=1;
count=1;
end
end
SRcount=SRcount+1;
%%%%%%%%%%%%%%%%%%%%%gets statistics for current frame
[mu,stdev,sumMinus,count]=runStat(stdev,sumMinus,count,statImag);
count=count+1;%keeps track of # of frames used in statistical window
%%%%%Starts the second tier of statistics gathering%%%%%%%%%%%%%
if count>window||count2>window
%gets statistics for current frame
[mu2,stdev2,sumMinus2,count2]=runStat(stdev2,sumMinus2,count2,statImag);
count2=count2+1;%keeps track of # of frames used in statistical window
end
%%%Resets variable to start process over as the two tiers alternate
if count==(window*2)+1
count=1;
mu=0;
stdev=0;
sumMinus=0;
display=2;%alternates b/n dislpaying filter 1 and 2
end
```

```
if count2==(window*2)+1
count2=1;
mu2=0;
stdev2=0;
sumMinus2=0;
display=1;%alternates b/n dislpaying filter 1 and 2
end
%%%%%%Alternates between displaying the tiers%%%%%%%%%%%%%%
if display==2
dyn_img=((imgnow>mu2+scale.*stdev2)|(imgnow<mu2-scale.*stdev2))&mask;
else
dyn_img=((imgnow>mu+scale.*stdev)|(imgnow<mu-scale.*stdev))&mask;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dyn_img=bwmorph(dyn_img,'majority');%removes isolated pixels
%%%Finds groups, determins size and angle, makes classification%%%%%
 [GL
dyn_img]=getGroupPos4(dyn_img,centerxcol,centeryrow,rad_cir,groupingThreshold);
ccResidual=getGroupResidual(GL,ccResidual,xtraAngle);
% next line records all possible targets at each frame%%
answer.List(ii).Group=GL;
answer.List(ii).GroupResidual=ccResidual;
```

```
imshow(mat2gray(dyn_img));%displays image for frame capture to save in movie
if trackTarget==1 %Adds tracked target info (# of humans, angles of humans) to
displayed frame
actualFrame=startFrame+ii-1;%The actual frame in the original video
title(['Frame: ',num2str(actualFrame),' ,Humans : ',num2str(GL.human),', Angles :
',num2str(GL.angleTarget)]);
end
%     F = getframe(gca);          % captures current frame being displayed (just the
window)
F = getframe(gcf);          % captures current figure being displayed (titles,axis,etc)
out=addframe(out,F);        % appends frame to video
end
if trackTarget==1
answer.BaseInfo.filename=fileName;
answer.BaseInfo.numFrames=framesRead;
answer.BaseInfo.startStopFrame=[startFrame endFrame];
answer.BaseInfo.frameSizeRC=frameSize;
answer.BaseInfo.centerRow=centeryrow;
answer.BaseInfo.centerCol=centerxcol;
answer.BaseInfo.circleRad=rad_cir;
end
save(fileName, 'answer');
```

```
out = close(out);          %save and close the video
```

A.2.2. getGroupPos4

```
function [GL
outbin]=getGroupPos4(bin,centerxcol,centeryrow,rad_cir,groupingThreshold,humanHWr
atioThreshhold)
%groups connected pixels in binary img 'bin'
cc = bwconncomp(bin);%Finds groups of pixels
[M,N]=size(bin);
[NumGroup]=cc.NumObjects;
% [NumGroup]=length(cc.PixelIdxList);
%number of groups created %
for i=1:1:NumGroup%Produces pixel locations in row,col format in PixelIdxList2
celli=cc.PixelIdxList(1,i);
%group cell
cellVal=celli{1,1};
%group cell values
cc.GroupLength(i)=length(cellVal);
%number of pixels in a group
%converts pixel location to row,col coordinates. for a 3x3 matrix,
%pixel (1,3) corresponds to pixel location 7.
quotient = idivide(int32(cellVal),int32(M),'floor');
colPos=quotient+1;
```

```
rowPos=cellVal-(double(quotient).*M);
cc.PixelIdxList2(1,i).cell(:,1)=rowPos;
cc.PixelIdxList2(1,i).cell(:,2)=colPos;
end
human=0;%intializes human target count
angleTarget=-1;%intializes human target angle(default negative)
validGroups=0;%initialize the number of valid pixel groups(groups>groupingThreshold)
for i=1:1:NumGroup %goes through each group
distMax=0;%initialize variable
distMin=N+1;%initialize variable
thetaMax=-1;%initialize variable
thetaMin=361;%initialize variable
if cc.GroupLength(i)>groupingThreshold %selects groups larger than grouping threshold
validGroups=validGroups+1;
for j=1:1:cc.GroupLength(i)
%finds distance and angle of each pixel in valid group
dist=sqrt((centeryrow-cc.PixelIdxList2(1,i).cell(j,1))^2 + (centerxcol-
cc.PixelIdxList2(1,i).cell(j,2))^2);
thetaThisPixel=getAngleFromCenter(cc.PixelIdxList2(1,i).cell(j,1),cc.PixelIdxList2(1,i).
cell(j,2),centeryrow,centerxcol);
if thetaThisPixel>thetaMax
thetaMax=thetaThisPixel;%finds max angle in valid group
end
```

```
if thetaThisPixel<thetaMin
thetaMin=thetaThisPixel;%finds min angle in valid group
end
if dist<distMin
distMin=dist;%finds min distance in valid group
end
if dist>distMax
distMax=dist;%finds max distance in valid group
end
end
%       finds the average angle of the target
theta=getAverageAngle(thetaMax,thetaMin);
%Calulate height, width and ratio
height=distMax-distMin;
aTSR=(thetaMax-thetaMin)*(2*pi/(360));%angleTargetSubtendsRadians
width=(distMin+((distMax-distMin)/2))*aTSR;%S=r*theta(radians)
%Records information for each valid group
GL.GroupList(validGroups).size=cc.GroupLength(i);
GL.GroupList(validGroups).Angle=[thetaMax thetaMin];
GL.GroupList(validGroups).Radius=[distMax distMin];
GL.GroupList(validGroups).HW=[height width];
GL.GroupList(validGroups).Ratio=height/width;
GL.GroupList(validGroups).PixelIdxList2=cc.PixelIdxList2(1,i).cell(:,:);
```

```
end
end
%Initializes values to return incase there were not any valid groups (AVOID
%ERROR)
if (validGroups==0)
GL.GroupList(1).size=0;
GL.GroupList(1).Angle=[0 0];
GL.GroupList(1).Radius=[0 0];
GL.GroupList(1).HW=[0 0];
GL.GroupList(1).Ratio=0;
GL.GroupList(1).PixelIdxList2=[0 0];
end
```

A.2.3. getGroupResidual

```
function ccNewResidual=getGroupResidual(cc,ccResidual,xtraAngle)
ccNewResidual.NumObjects=0;%initialize variable
count=1;
setflag=0;
if (ccResidual.NumObjects>0)&&(cc.NumObjects>0)%if the residual and new structs
are not empty
for groupNumcc=1:1:cc.NumObjects%cycle through all the new group
%The following three lines sets up cc.GroupList to be transfered to
%ccNewResidual regardless of whether matches were found or not. If
```

```
%no matches were found, then current grups will be saved as the
%next residuals. If matches are found, then current groups are
%added to previous groups.
cc.GroupList(groupNumcc).residual=1;%initializes variable so cc.GroupList is the same
size as ccResidual.GroupList
cc.GroupList(groupNumcc).RatioList=cc.GroupList(groupNumcc).Ratio;%initializes
variable so cc.GroupList is the same size as ccResidual.GroupList
temp.GroupList=cc.GroupList(groupNumcc);%Transfers current group to temp
for groupNumccRes=1:1:ccResidual.NumObjects%cycle through all the residual group
groupAngle=getAverageAngle(cc.GroupList(groupNumcc).Angle(1),cc.GroupList(group
Numcc).Angle(2));
groupResidualAngle=getAverageAngle(ccResidual.GroupList(groupNumccRes).Angle(1
),ccResidual.GroupList(groupNumccRes).Angle(2));
if (groupAngle>=groupResidualAngle-
xtraAngle)&&(groupAngle<=groupResidualAngle+xtraAngle)
setflag=1;%Sets flag if the scenario of a match between residual and current group has
been found
temp.GroupList.residual=ccResidual.GroupList(groupNumccRes).residual+1;%Increase
the residual duration count in the residual group
end
ccNewResidual.GroupList(count)=temp.GroupList; %transfers temp to NewResidual
group
if setflag==1%the scenario of a match between residual and current group has been found
```

```
temp.GroupList.RatioList =ccResidual.GroupList(groupNumccRes).RatioList;%Records
the previous residual recorded ratio values
temp.GroupList.RatioList(1,length(temp.GroupList.RatioList)+1)=cc.GroupList(groupN
umcc).Ratio;%Adds the current ratio value to the residual list
%Ratio values are maintained for averaging purposes.
ccNewResidual.GroupList(count).RatioList=temp.GroupList.RatioList;%transfers
ratioList to output
end
setflag=0;%resets flag for next iteration
end
ccNewResidual.NumObjects=ccNewResidual.NumObjects+1;%increase the number of
objects in new residual
count=count+1;%Counts the number of residual groups created. May not be necessary as
a residual group is created for every current
%group. Either the groups are matched and the residual is saved
%into the output with the new info added or no match is found and
%the current group is saved as a 1st time residual group for the
%next frame. Perhaps usful if the same current group gets matched
%to multiple residual grups near the same angle.
end
else%There were no reisudal groups and no current groups
ccNewResidual=cc;%assigns the current structure to the output(residual)
for groupNumcc=1:1:cc.NumObjects
```

```
ccNewResidual.GroupList(groupNumcc).residual=1;%initializes variable so the structure
is the same size
ccNewResidual.GroupList(groupNumcc).RatioList=ccNewResidual.GroupList(groupNu
mcc).Ratio; %initialize variable so structure components are the same size when
transfering
end
end
```

A.2.4. getGroundTruth

```
function truth=getGroundTruth(answer)
startFrame=301;
stopFrame = 1001;%answer.BaseInfo.numFrames;
% if nargin==1;
%     startFrame=1;
%     endFrame = answer.BaseInfo.numFrames;
% end
% startFrame=startFrame-answer.BaseInfo.startStopFrame(1,1)+1;
% endFrame=endFrame-answer.BaseInfo.startStopFrame(1,1)+1;
iteration=1;
for frame=startFrame:1:stopFrame
img = zeros(answer.BaseInfo.frameSizeRC);
for groupi=1:1:answer.List(1,frame).Group.NumObjects
for pixelList=1:1:answer.List(1,frame).Group.GroupList(1,groupi).size
```

```
rowPixel=answer.List(1,frame).Group.GroupList(1,groupi).PixelIdxList2(pixelList,1);
colPixel=answer.List(1,frame).Group.GroupList(1,groupi).PixelIdxList2(pixelList,2);
img(rowPixel,colPixel)=1;
end
end
actualFrame=frame+answer.BaseInfo.startStopFrame(1,1)-1;
img(30,1:30)=1;
img(1:30,30)=1;
imshow(mat2gray(img));
title(['Select All Humans in Frame#: ',num2str(actualFrame),' if main human not there,
check box']);
[x,y] = getpts(gcf);
truth.List(1,iteration).human(:,1)=y;%row
truth.List(1,iteration).human(:,2)=x;%col
close gcf;
img(1:30,1:30)=1;
imshow(mat2gray(img));
title(['Select Animals in Frame#: ',num2str(actualFrame)]);
[x,y] = getpts(gcf);
truth.List(1,iteration).horse(:,1)=y;
truth.List(1,iteration).horse(:,2)=x;
close gcf;
truth.List(1,iteration).frame=actualFrame;
```

```
iteration=iteration+1;
end
truth.BaseInfo=answer.BaseInfo;
truth.truthInfo.numFrames=stopFrame-startFrame+1;
truth.truthInfo.startFrame=startFrame+answer.BaseInfo.startStopFrame(1,1)-1;
truth.truthInfo.stopFrame=stopFrame+answer.BaseInfo.startStopFrame(1,1)-1;
truth=reEvaluateTruth(truth);%adds angle and radius info
filename=answer.BaseInfo.filename;
saveFilename=[filename 'Truth'];
save(saveFilename, 'truth');
```

A.2.5. reEvaluateTruth

```
function truth=reEvaluateTruth(truth)
%revevaluates the truth struct and put results in terms of angle and radius
centeryrow=truth.BaseInfo.centerRow;
centerxcol=truth.BaseInfo.centerCol;
for frame=1:1:truth.truthInfo.numFrames
humanCount=0;
horseCount=0;
[numHumansThisFrame unused]=size(truth.List(1,frame).human);
[numHorsesThisFrame unused]=size(truth.List(1,frame).horse);
for human=1:1:numHumansThisFrame
pixelyrow=truth.List(1,frame).human(human,1);
```

```
pixelxcol=truth.List(1,frame).human(human,2);
if (pixelyrow<=30)||(pixelxcol<=30)||isempty(pixelyrow)
truth.List(1,frame).humanAngRad(human,1:2)=[0,0];
else
truth.List(1,frame).humanAngRad(human,1)=getAngleFromCenter(pixelyrow,pixelxcol,c
enteryrow,centerxcol);
truth.List(1,frame).humanAngRad(human,2)=sqrt( ((pixelyrow-centeryrow).^2) +
((pixelxcol-centerxcol).^2) );
humanCount=humanCount+1;
end
end

for horse=1:1:numHorsesThisFrame
pixelyrow=truth.List(1,frame).horse(horse,1);
pixelxcol=truth.List(1,frame).horse(horse,2);
if (pixelyrow<=30)||(pixelxcol<=30)||isempty(pixelyrow)
truth.List(1,frame).horseAngRad(horse,1:2)=[0,0];
else
truth.List(1,frame).horseAngRad(horse,1)=getAngleFromCenter(pixelyrow,pixelxcol,cen
teryrow,centerxcol);
truth.List(1,frame).horseAngRad(horse,2)=sqrt( ((pixelyrow-centeryrow).^2) +
((pixelxcol-centerxcol).^2) );
horseCount=horseCount+1;
```

```
end
end
truth.List(1,frame).numHuman = humanCount;
truth.List(1,frame).numHorse = horseCount;
end
```

A.2.6. compareAnswerTruth

```
function answer=compareAnswerTruth(answer,truth,xtraAngle)
%truth should be modified by truth=reEvaluateTruth(truth)
% xtraAngle=2;%degrees, the amount angle that supposed object can be from the
%truth object and still be considered the same object.
startFrameTruth=truth.truthInfo.startFrame;
stopFrameTruth=truth.truthInfo.stopFrame;
numFrames=truth.truthInfo.numFrames;
startFrameAnswer=answer.BaseInfo.startStopFrame(1,1);
for truthListPos=1:1:numFrames
answerListPos=startFrameTruth+truthListPos-startFrameAnswer;
if answer.List(1,answerListPos).Group.GroupList(1,1).size==0%no tartgets detected
continue;
end
for object=1:1:length(answer.List(1,answerListPos).Group.GroupList)
thetaMax=answer.List(1,answerListPos).Group.GroupList(1,object).Angle(1,1);
thetaMin = answer.List(1,answerListPos).Group.GroupList(1,object).Angle(1,2);
```

```
%finds the average angle of the target
objectAngle=getAverageAngle(thetaMax,thetaMin);
answer.List(1,answerListPos).Group.GroupList(1,object).AngleAvg=objectAngle;
answer.List(1,answerListPos).Group.GroupList(1,object).ValidTarget=0;
answer.List(1,answerListPos).Group.GroupList(1,object).humanTarget=0;
answer.List(1,answerListPos).Group.GroupList(1,object).horseTarget=0;
for checkHuman=1:1:truth.List(1,truthListPos).numHuman
realAngle=truth.List(1,truthListPos).humanAngRad(checkHuman,1);
sameAngle=sameAngleCheck(objectAngle,realAngle,xtraAngle);
if sameAngle==1
answer.List(1,answerListPos).Group.GroupList(1,object).ValidTarget=1;
answer.List(1,answerListPos).Group.GroupList(1,object).humanTarget=1;
end
end
for checkHorse=1:1:truth.List(1,truthListPos).numHorse
realAngle=truth.List(1,truthListPos).horseAngRad(checkHorse,1);
sameAngle=sameAngleCheck(objectAngle,realAngle,xtraAngle);
if sameAngle==1
answer.List(1,answerListPos).Group.GroupList(1,object).ValidTarget=1;
answer.List(1,answerListPos).Group.GroupList(1,object).horseTarget=1;
end
end
end
```

```
end
```

A.2.7. showResults

```
function
result=showResults(answer,truth,startFrame,stopFrame,startAngle,stopAngle,residualThr
eshold,detWindow)
[halfAngle,halfAngleRange]=getAverageAngle2(startAngle,stopAngle);
%Finds the range in angle at which targets are valid given start/stop angle
numFrames=stopFrame-startFrame+1;
%start at 300, stop at 305, will produce 6 frames
startFrameAnswer = answer.BaseInfo.startStopFrame(1,1);
startFrameTruth = truth.truthInfo.startFrame;
firstHumanFlag=0;
firstHumanFrame=-1;
for index=1:1:numFrames
%finds the index for the corresponding to original raw frame video
indexAnswer = startFrame - startFrameAnswer + index;
indexTruth = startFrame - startFrameTruth + index;
realHumanCount=0;
realHorseCount=0;
%gets the number of real humans and horses in the frame within the
%given angle
%     result.List(1,index).realHuman = truth.List(1,indexTruth).numHuman;
```

```
%     result.List(1,index).realHorse = truth.List(1,indexTruth).numHorse;
for humansInFrame=1:1:truth.List(1,indexTruth).numHuman
humanAngle=truth.List(1,indexTruth).humanAngRad(humansInFrame,1);
sameAngle=sameAngleCheck(halfAngle,humanAngle,halfAngleRange);
realHumanCount=realHumanCount+sameAngle;
end
for HorsesInFrame=1:1:truth.List(1,indexTruth).numHorse
horseAngle=truth.List(1,indexTruth).horseAngRad(HorsesInFrame,1);
sameAngle=sameAngleCheck(halfAngle,horseAngle,halfAngleRange);
realHorseCount=realHorseCount+sameAngle;
end
%If there were no detections this frame
%     if answer.List(1,indexAnswer).Group.GroupList(1,1).size==0
if answer.List(1,indexAnswer).Group.NumObjects==0
result.List(1,index).realHuman=0;
result.List(1,index).realHorse=0;
result.List(1,index).detectedHuman=0;
result.List(1,index).detectedHorse=0;
result.List(1,index).persistentHuman=0;
result.List(1,index).persistentHorse=0;
result.List(1,index).classifiedHuman=0;
result.List(1,index).classifiedHorse=0;
result.List(1,index).classifiedHumanCorrectly=0;
```

```
result.List(1,index).classifiedHorseCorrectly=0;
result.List(1,index).classifiedHumanIncorrectly=0;
result.List(1,index).classifiedHorseIncorrectly=0;
else%there were detections this frame
%       validObjects=0;
%       validObjectsCount=0;
detectedHumanCount=0;
detectedHorseCount=0;
persistentHumanCount=0;
persistentHorseCount=0;
classifiedHumanCount=0;
classifiedHorseCount=0;
classifiedHumanCorrectlyCount=0;
classifiedHorseCorrectlyCount=0;
classifiedHumanIncorrectlyCount=0;
classifiedHorseIncorrectlyCount=0;
%cycle through valid objects
for object=1:1:answer.List(1,indexAnswer).Group.NumObjects;
%
validObjectsCount=validObjectsCount+answer.List(1,indexAnswer).Group.validTarget;
%           validObjects(1,validObjectsCount)=object;%records the object position of
correct detections
%         Checks whether object fall within given angle range
```

```
objectAngle=answer.List(1,indexAnswer).Group.GroupList(1,object).AngleAvg;
sameAngle=sameAngleCheck(halfAngle,objectAngle,halfAngleRange);
if sameAngle==1
%          sums detected humans and horses
% **********Note: if horse and human both fell under single detection
% (they were close in angle or overlapped), this
% procedure will show as detecting the horse and detecting the human
detectedHumanCount = detectedHumanCount +
answer.List(1,indexAnswer).Group.GroupList(1,object).humanTarget;
detectedHorseCount = detectedHorseCount +
answer.List(1,indexAnswer).Group.GroupList(1,object).horseTarget;
end
end
for object=1:1:answer.List(1,indexAnswer).GroupResidual.NumObjects
%check whther object is within start/stop angle boundaries
grpResAngleMax=answer.List(1,indexAnswer).GroupResidual.GroupList(1,object).Angl
e(1,1);
grpResAngleMin=answer.List(1,indexAnswer).GroupResidual.GroupList(1,object).Angl
e(1,2);
grpResAngle=getAverageAngle(grpResAngleMax,grpResAngleMin);%finds objects
average angle
sameAngle=sameAngleCheck(halfAngle,grpResAngle,halfAngleRange);%is angle
within start/stop
```

```
if sameAngle==1
if
answer.List(1,indexAnswer).GroupResidual.GroupList(1,object).residual>=residualThres
hold
%Checks whether this residual object was a valid (real human or horse)
%target on this frame
for ObjectValidity=1:1:answer.List(1,indexAnswer).Group.NumObjects
grpAngle=answer.List(1,indexAnswer).Group.GroupList(1,ObjectValidity).AngleAvg;
if grpResAngle==grpAngle
validObject=answer.List(1,indexAnswer).Group.GroupList(1,ObjectValidity).ValidTarge
t;
humanObject=answer.List(1,indexAnswer).Group.GroupList(1,ObjectValidity).humanTa
rget;
horseObject=answer.List(1,indexAnswer).Group.GroupList(1,ObjectValidity).horseTarg
et;
end
end
%%%%%%%%%%%%%%%%%%%%%classifies residual object
numResiduals=answer.List(1,indexAnswer).GroupResidual.GroupList(1,object).residual;
%Averages the last x recorded HW ratios for a target. Here
%x  is the residualThreshold
feature=sum(answer.List(1,indexAnswer).GroupResidual.GroupList(1,object).RatioList((
numResiduals-residualThreshold+1):numResiduals))/residualThreshold;
```

```
class=mahalonobis1feature(feature);%Returns 1 if class is human,0 otherwise
%records what the classifcation was and if it was correct
if (humanObject==1)&&(horseObject==0)
%only count objects which can be human only
persistentHumanCount=persistentHumanCount+1;
if firstHumanFlag==0
%Record when the first correct human classification occurred
firstHumanFrame=indexAnswer+startFrameAnswer-1;
firstHumanFlag=1;
end
end
if (horseObject==1)&&(humanObject==0)
%only count objects which can be horse only
persistentHorseCount=persistentHorseCount+1;
end
if class==1%Human
classifiedHumanCount=classifiedHumanCount+1;
if (humanObject==1)&&(horseObject==0)
%Correct Classification Human%%%%%%%%%%%%%%%%%%
%only count objects which can be human only and
%were correctly classified
classifiedHumanCorrectlyCount=classifiedHumanCorrectlyCount+1;
end
```

```
if (humanObject==0)&&(horseObject==1)%FalseAlarm%%%%%%%
%only count objects which can be human only and
%were correctly classified
classifiedHumanIncorrectlyCount=classifiedHumanIncorrectlyCount+1;
end
else %class==0 Horse
classifiedHorseCount=classifiedHorseCount+1;
if (horseObject==1)&&(humanObject==0)%Correct Classification Animal or Correct
Rejection%%%%%%
%only count objects which can be horse only and
%were correctly classified
classifiedHorseCorrectlyCount=classifiedHorseCorrectlyCount+1;
end
if
(horseObject==0)&&(humanObject==1)%Miss%%%%%%%%%%%%%%%%%%%
%only count objects which can be horse only and
%were correctly classified
classifiedHorseIncorrectlyCount=classifiedHorseIncorrectlyCount+1;
end
end
end%    if
answer.List(1,indexAnswer).GroupResidual.GroupList(1,object).residual>=residualThres
hold
```

```
end%    if sameAngle==1
end%    for object=1:1:answer.List(1,indexAnswer).GroupResidual.NumObjects
result.List(1,index).realHuman=realHumanCount;
result.List(1,index).realHorse=realHorseCount;
result.List(1,index).detectedHuman=detectedHumanCount;
result.List(1,index).detectedHorse=detectedHorseCount;
result.List(1,index).persistentHuman=persistentHumanCount;
result.List(1,index).persistentHorse=persistentHorseCount;
result.List(1,index).classifiedHuman=classifiedHumanCount;
result.List(1,index).classifiedHorse=classifiedHorseCount;
result.List(1,index).classifiedHumanCorrectly=classifiedHumanCorrectlyCount;
result.List(1,index).classifiedHorseCorrectly=classifiedHorseCorrectlyCount;
result.List(1,index).classifiedHumanIncorrectly=classifiedHumanIncorrectlyCount;
result.List(1,index).classifiedHorseIncorrectly=classifiedHorseIncorrectlyCount;
end%   if answer.List(1,indexAnswer).Group.GroupList(1,1).size==0
end%   for index=1:1:numFrames
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Finds the detection event that a target w/ history of residual number of
%frames was detected within a window (detWindow) # of frames. ie. A target
%with a that persisted for 20 frames has been detected within the last 45
%frames.
%Note these results only take into account target which are
%non-overlapping, which means they are either human or animal, but not both
```

```
%initializes results to zero for the region before a detection event can
%take place
for index=1:1:detWindow-1
result.List(1,index).detectionEventHumanwWindow=0;
result.List(1,index).detectionEventHorsewWindow=0;
result.List(1,index).possibleDetectionEventsHuman=0;
result.List(1,index).possibleDetectionEventsHorse=0;
end
%In the region where a detection event is valid, checks whether a persitent
%target has occured.
for index=detWindow:1:numFrames
%initialize variable at each frame
result.List(1,index).possibleDetectionEventsHuman=0;
result.List(1,index).possibleDetectionEventsHorse=0;
result.List(1,index).detectionEventHumanwWindow=0;
result.List(1,index).detectionEventHorsewWindow=0;
for detIndex=index-detWindow+1:1:index
%Finds the total number of possible detections.
if result.List(1,detIndex).realHuman>0
result.List(1,index).possibleDetectionEventsHuman=1;
end
if result.List(1,detIndex).realHorse>0
```

```
result.List(1,index).possibleDetectionEventsHorse=1;
end
%Finds the actual detection
if result.List(1,detIndex).persistentHuman==1
result.List(1,index).detectionEventHumanwWindow=1;
end
if result.List(1,detIndex).persistentHorse==1
result.List(1,index).detectionEventHorsewWindow=1;
end
end
end
%%%%%%% Produces Totals
realHumanCountTot=0;
realHorseCountTot=0;
detectedHumanCountTot=0;
detectedHorseCountTot=0;
persistentHumanCountTot=0;
persistentHorseCountTot=0;
classifiedHumanCountTot=0;
classifiedHorseCountTot=0;
classifiedHumanCorrectlyCountTot=0;
classifiedHorseCorrectlyCountTot=0;
classifiedHumanIncorrectlyCountTot=0;
```

```
classifiedHorseIncorrectlyCountTot=0;
possibleDetectionEventsHumanTot=0;
possibleDetectionEventsHorseTot=0;
detectionEventHumanwWindowTot=0;
detectionEventHorsewWindowTot=0;
for index=1:1:numFrames
realHumanCountTot = realHumanCountTot + result.List(1,index).realHuman;
realHorseCountTot = realHorseCountTot + result.List(1,index).realHorse;
detectedHumanCountTot = detectedHumanCountTot +
result.List(1,index).detectedHuman;
detectedHorseCountTot = detectedHorseCountTot + result.List(1,index).detectedHorse;
persistentHumanCountTot = persistentHumanCountTot +
result.List(1,index).persistentHuman;
persistentHorseCountTot = persistentHorseCountTot +
result.List(1,index).persistentHorse;
classifiedHumanCountTot = classifiedHumanCountTot +
result.List(1,index).classifiedHuman;
classifiedHorseCountTot = classifiedHorseCountTot +
result.List(1,index).classifiedHorse;
classifiedHumanCorrectlyCountTot = classifiedHumanCorrectlyCountTot +
result.List(1,index).classifiedHumanCorrectly;
classifiedHorseCorrectlyCountTot = classifiedHorseCorrectlyCountTot +
result.List(1,index).classifiedHorseCorrectly;
```

```
classifiedHumanIncorrectlyCountTot = classifiedHumanIncorrectlyCountTot +
result.List(1,index).classifiedHumanIncorrectly;
classifiedHorseIncorrectlyCountTot = classifiedHorseIncorrectlyCountTot +
result.List(1,index).classifiedHorseIncorrectly;
possibleDetectionEventsHumanTot = possibleDetectionEventsHumanTot +
result.List(1,index).possibleDetectionEventsHuman;
possibleDetectionEventsHorseTot = possibleDetectionEventsHorseTot +
result.List(1,index).possibleDetectionEventsHorse;
detectionEventHumanwWindowTot = detectionEventHumanwWindowTot +
result.List(1,index).detectionEventHumanwWindow;
detectionEventHorsewWindowTot = detectionEventHorsewWindowTot +
result.List(1,index).detectionEventHorsewWindow;
end
result.Total.realHuman=realHumanCountTot;
result.Total.realHorse=realHorseCountTot;
result.Total.detectedHuman=detectedHumanCountTot;
result.Total.detectedHorse=detectedHorseCountTot;
result.Total.persistentAssumedRealHuman=persistentHumanCountTot;
result.Total.persistentAssumedRealHorse=persistentHorseCountTot;
result.Total.classifiedHuman=classifiedHumanCountTot;
result.Total.classifiedHorse=classifiedHorseCountTot;
result.Total.classifiedHumanCorrectly=classifiedHumanCorrectlyCountTot;
result.Total.classifiedHorseCorrectly=classifiedHorseCorrectlyCountTot;
```

```
result.Total.classifiedHumanIncorrectly=classifiedHumanIncorrectlyCountTot;
result.Total.classifiedHorseIncorrectly=classifiedHorseIncorrectlyCountTot;
result.Total.possibleDetectionEventsHuman=possibleDetectionEventsHumanTot;
result.Total.possibleDetectionEventsHorse=possibleDetectionEventsHorseTot;
result.Total.detectionEventHumanwWindow=detectionEventHumanwWindowTot;
result.Total.detectionEventHorsewWindow=detectionEventHorsewWindowTot;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%the probHumanClass is the number of correct human classifications over the
%number of humans able to be classified. This does not take into account
%the probability of detection. This only says how often are the
%classifications correct
result.percentages.probHumanClass=(result.Total.classifiedHumanCorrectly)/result.Total
.persistentAssumedRealHuman;
result.percentages.probHorseClass=result.Total.classifiedHorseCorrectly/result.Total.pers
istentAssumedRealHorse;
result.percentages.probHumanDet=result.Total.detectionEventHumanwWindow/result.T
otal.possibleDetectionEventsHuman;
result.percentages.probHorseDet=result.Total.detectionEventHorsewWindow/result.Total
.possibleDetectionEventsHorse;
result.percentages.probClass=(result.Total.classifiedHumanCorrectly+result.Total.classifi
edHorseCorrectly)/(result.Total.persistentAssumedRealHuman+result.Total.persistentAss
umedRealHorse);
```

```
result.percentages.probDet=(result.Total.detectionEventHumanwWindow+result.Total.de
tectionEventHorsewWindow)/(result.Total.possibleDetectionEventsHuman+result.Total.
possibleDetectionEventsHorse);
result.percentages.probCandD=result.percentages.probClass*result.percentages.probDet;
result.firstHumanCrrectClassificationFrame=firstHumanFrame;
result.resultInfo.startFrame=startFrame;
result.resultInfo.stopFrame=stopFrame;
result.resultInfo.startAngle=startAngle;
result.resultInfo.stopAngle=stopAngle;
result.resultInfo.residualThreshold=residualThreshold;
result.resultInfo.BaseInfo=answer.BaseInfo;
result.resultInfo.truthInfo=truth.truthInfo;
% The result(structure) has a section called "Total," which has a summation
% of the results generated at each frame. Under "Total" we have:
%
% I. Total
%     a)realHuman
%     b)realHorse
%     c)detectedHuman
%     d)detectedHorse
%     e)persistentAssumedRealHuman
%     f)persistentAssumedRealHorse
%     g)classifiedHuman
```

```
%       h)classifiedHorse
%       i)classifiedHumanCorrectly
%       j)classifiedHorseCorrectly
%
% a & b) realHuman/realHorse
% These values correspond to ground truth only. These are the objects that
% the human observer considered to be human or animal at each frame. This
% information is only as good as the human observer or the ground truth
% generating algorithm(interpolation).
%
% c & d) detectedHuman/Horse
% These are objects that were detected(any object over a threshold number
% of pixels in size) and matched against a human or horse in the ground
% truth. This value tells whether you detected signal from the human or
% animal. It does not classify. The human/animal in the title implies that
% the nameless object that was detected either matched against a ground
% truth human or a ground truth animal.
%
% e & f)persistentAssumedRealHuman
% These are objects which 1) have a history of a "residual threshold"
% number of consecutive frames and 2)have signal coming from only a human
% or only from a horse. It does not classify. The object at the current
% frame is compared against the ground truth to determine if the signal
```

```
% came from a human or animal target.  Also the history associated with
% this object is not being checked by the ground truth, only the current
% frame. This means that if a human and horse were overlapped and then
% split apart, they would both contain history from when they were
% overlapped. And even though the signal is identified as coming from
% a human at the current frame, the object would have history that
% incorporated the horse. You can think of this as a detection,
% where a single detection refers to an object which persists for a
% given number of frames.
%
% g & h)classified human/horse
% These are the results of classification of any and all classifications
% performed. It does not state whether the classification was correct or
% whether the object was generated from noise. It only gives the output of
% the classifier at every time the classifier was run.
%
% i & j)classifiedHumanCorrectly
% These are objects which were classified correctly against "only" a human
% or "only" a horse. When I say "only," I am saying that the object could
% only be matched to human in the ground truth or only a horse in the
% ground truth. When human and animals are to close, the object could
% potentially be matched to either the human or the animal. Because of this
% issue, I have ignored the cases where this is possible. I only report
```

```
% cases where the human can be separated from the animal.
%
% Also there is a percentage "probHumanClass"
% This is simply classifiedHumanCorrectly/persistentAssumedRealHuman.
% This value simply gives a number that states,"Well, when you detected a
% human and had the opportunity to classify, did you classify it correctly?"
```

A.2.8. runStat

```
function [mu,stdev,sumMinus,count]=runStat(stdPrev,sumMinus,count,frame2)
% %Returns the mean,standard deviation of the frames in question using the
% last std dev, and sum of values up to (but not including)the current
% value, the current value, and the number of values(frames) in question
% sumMinus : the sum of values up to (but not including)the current value
% frame2 : the current value(frame)
% stdPrev : the previous standard deviation
% count : the number of frames used to generate the statistics
%mu : mean
if count==1
mu=frame2;
stdev=zeros(size(frame2));
sumMinus=frame2;
end
if count==2
```

```
muPrev=sumMinus;
frame1=sumMinus;
mu=(frame1+frame2)./2;
stdev=sqrt( ( ((frame1-muPrev).^2) + ((frame2-muPrev).^2)) ./count );
sumMinus=frame1+frame2;
end
if count>2
muPrev=sumMinus./(count-1);
mu = ((count-1) .* muPrev + frame2)./count;
stdev=sqrt(((count-1).*((stdPrev).^2) + 2.*muPrev.*(sumMinus) -(count-1).*(muPrev.^2)
- 2.*mu.*(sumMinus) - (2.*mu.*frame2)  + (frame2.^2) + count.*((mu).^2)  )./count);
sumMinus=sumMinus+frame2;
end
```

A.2.9. sameAngleCheck

```
function sameAngle=sameAngleCheck(T1Angle,T2Angle,xtraAngle)
%returns 1 if T1 and T2 are seperated in angle by 'xtraAngle' or less
sameAngle=0;
if (T1Angle-xtraAngle)<=0%T1 first quad,lowerbound fourth quad
if (T2Angle)>270%T2 fourth quad
T2Angle=T2Angle-360;
if T2Angle>=(T1Angle-xtraAngle)
sameAngle=1;%T2 is within lowerbound and 0 degrees
```

```
end
elseif (T2Angle)<90%T2 first quad
if T2Angle<=(T1Angle+xtraAngle)
sameAngle=1;%T2 is within upper bound and 0 degrees
end
end
elseif (T1Angle+xtraAngle)>=360%T1 is in fourth quad,upper bound first quad
if (T2Angle)<90%T2 first quad
T2Angle=T2Angle+360;
if T2Angle<=(T1Angle+xtraAngle)
sameAngle=1;%T2 is within upper bound and zero degrees
end
elseif (T2Angle)>270%T2 fourth quad
if (T2Angle)>=(T1Angle-xtraAngle)
sameAngle=1;%T2 is within lower bound and zero degrees
end
end
else
if (T2Angle<=T1Angle+xtraAngle)&&(T2Angle>=T1Angle-xtraAngle)
sameAngle=1;%T1 is near T2
end
end
```

A.2.10.getAngleFromCenter

```
function theta=getAngleFromCenter(pixelyrow,pixelxcol,centeryrow,centerxcol)
% Given a circle with center and point on circle, returns the corresponding
% angle in degrees
%finds angle of first pixel in group
theta=atand((centeryrow-pixelyrow) ./ (pixelxcol-centerxcol) );
if (pixelxcol-centerxcol)<0%correcting -pi to pi into 0 to 2pi,2&3Quad
theta=theta+180;
end
if theta<0
theta=360+theta;%correcting -pi to pi into 0 to 2pi,4thQuad
end
%Just to insure infinity is not returned from possible tangent calculations
if theta==90
theta=89.99;
end
if theta==180
theta=179.99;
end
if theta==270
theta=269.99;
end
if theta==360
```

```
theta=259.99;
end
```

A.2.11.getAverageAngle

```
function theta=getAverageAngle(thetaMax,thetaMin)
if (thetaMax>270)&&(thetaMin<90)%checks for 0 and 360 degree boarder
theta=(((360-thetaMax)+thetaMin)/2)+thetaMax;
if theta>=360
theta=theta-360;
end
else
theta=(thetaMax+thetaMin)/2;
end
```

A.2.12.getAverageAngle2

```
function [theta,range]=getAverageAngle2(startAngle,stopAngle)
%given a start angle and a stop angle, finds the midpoint(angle) between
%and the angle between the midpoint and the border(start or stop)
if startAngle>stopAngle
thetaMax=startAngle;
thetaMin=stopAngle;
theta=(((360-thetaMax)+thetaMin)/2)+thetaMax;
if theta>=360
```

```
theta=theta-360;
end
range=(((360-thetaMax)+thetaMin)/2);
else
thetaMax=stopAngle;
thetaMin=startAngle;
theta=(thetaMax+thetaMin)/2;
range=thetaMax-theta;
end
```

A.2.13.regroupSizeAnswer

```
function [NewAnswer]=RegroupSizeAnswer(answer,groupingThreshold,xtraAngle)
%%%%%%%%%%%%Initialize Variable for Residual Group Tracking%%%%%
ccResidual.NumObjects=0;%initialize variable
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
startFrame=1;
stopFrame=answer.BaseInfo.numFrames;
for frame=startFrame:1:stopFrame%cycle thru all frames
countGroup=0;
for groupi=1:1:answer.List(1,frame).Group.NumObjects%cycle thru groups in this frame
if answer.List(1,frame).Group.GroupList(1,groupi).size>=groupingThreshold%
%checks if group is larger or equal than grouping threshold
countGroup=countGroup+1;%counts valid groups
```

```
Group.GroupList(1,countGroup)=answer.List(1,frame).Group.GroupList(1,groupi);
%Saves larger groups in new list
end
end
if countGroup==0
Group.GroupList(1,1).size=0;
Group.GroupList(1,1).Angle=zeros(1,2);
Group.GroupList(1,1).Radius=zeros(1,2);
Group.GroupList(1,1).HW=zeros(1,2);
Group.GroupList(1,1).Ratio=0;
Group.GroupList(1,1).PixelIdxList2=zeros(1,2);
end
Group.NumObjects=countGroup;%saves the number of valid groups
ccResidual=getGroupResidual(Group,ccResidual,xtraAngle);%finds residual groups
NewAnswer.List(1,frame).Group=Group;
NewAnswer.List(1,frame).GroupResidual=ccResidual;
clear Group;
end
```