2014

Fragment Association Matching Enhancement (FAME) on a Video Tracker

Andrew Johnson
Wright State University

Follow this and additional works at: http://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons

Repository Citation

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu.
FRAGMENT ASSOCIATION MATCHING ENHANCEMENT (FAME) ON A VIDEO TRACKER

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

By

ANDREW JOHNSON
B.S., Wright State University, 2012

2014
Wright State University
April 24, 2014

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Andrew Hawn Johnson ENTITLED Fragment Association Matching Enhancement (FAME) on a Video Tracker BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Computer Engineering

Dr. Wischgoll, Thesis Director

Mateen Rizki,
Ph.D. Chair, Department
Computer Science & Engineering

Committee on Final Examination

Juan Vasquez, Ph.D.

Thomas Wischgoll, Ph.D.

Arthur Goshtasby, Ph.D.

Robert E.W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School
ABSTRACT


In the field of surveillance, algorithms are developed to extract meaningful information out of a video feed captured via a camera. One type of algorithm used in the field of surveillance is a tracking algorithm. A tracking algorithm allows a user to watch the movement of an object in the camera's field of view. The tracker used in this thesis research is a feature aided tracker (FAT). The FAT uses both features and kinematics to generate tracks. However, camera movement will affect the tracker's ability to accurately track an object which poses a problem to the tracker. Specifically, the camera will introduce the multi-fragmentation problem to the tracker.

Multi-fragmentation occurs when an object is marked with two tracks instead of a single track. By marking the object with two tracks, the tracker's performance and accuracy will decrease. This thesis research proposes the idea of matching features of small foreground objects (fragments) to create larger foreground objects. A pair of fragments will have their features calculated into a score. If the fragment pair's score is below a specific threshold, they will be matched to create a larger fragment. Many of the concepts used to design this tracking algorithm (FAME) stem from the fields of computer vision, pattern recognition, and tracking.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Terminology Used</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Problem.</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Goal.</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Layout of the Thesis.</td>
<td>5</td>
</tr>
<tr>
<td>2.0 BACKGROUND</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Median Filtering</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Frame Differencing vs. Background Modeling</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Frame Differencing</td>
<td>7</td>
</tr>
<tr>
<td>2.2.2 Frame Differencing over Background Modeling</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Adaptive Thresholding</td>
<td>10</td>
</tr>
<tr>
<td>2.4 Euclidean Distance</td>
<td>11</td>
</tr>
<tr>
<td>2.5 Eccentricity</td>
<td>13</td>
</tr>
<tr>
<td>2.6 Hu Moments</td>
<td>13</td>
</tr>
<tr>
<td>2.7 Bhattacharyya Distance</td>
<td>15</td>
</tr>
<tr>
<td>2.8 Sobel Edge Detection</td>
<td>16</td>
</tr>
<tr>
<td>2.9 Gray Level Co-Occurrence Matrix (GLCM)</td>
<td>17</td>
</tr>
<tr>
<td>2.9.1 GLCM features</td>
<td>18</td>
</tr>
<tr>
<td>2.10 Image Registration</td>
<td>21</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Continued)

3.0 FAME ................................................................. 23

3.1 FAME Requirements .................................................. 23

3.2 Checking Eligibility of Fragment Pairs ............................ 24
   3.2.1 Finding Contours ..................................................... 24
   3.2.2 Calculating Distance Between Fragments ....................... 25

3.3 Scoring Eligible Fragment Pairs ...................................... 26
   3.3.1 In-between Fragment Calculation using Arc Tangent Function .... 26
   3.3.2 Scoring Fragment Pair ............................................. 30

3.4 Match Pairs that are Above Threshold ............................. 31
   3.4.1 Fragment Match Check ......................................... 32

3.5 Region Calculation Function ......................................... 34

4.0 RESULTS ............................................................. 38

4.1 Experiment ........................................................... 40

4.2 Evaluation ........................................................... 48

5.0 CONCLUSION ......................................................... 51

REFERENCES ............................................................ 53
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. FAT tracker flowchart</td>
<td>2</td>
</tr>
<tr>
<td>2. Good tracking and multi-fragmentation problem</td>
<td>4</td>
</tr>
<tr>
<td>3. Flowchart of proposed FAT with the FAME algorithm</td>
<td>5</td>
</tr>
<tr>
<td>4. Shows a zoomed in fragmented car</td>
<td>8</td>
</tr>
<tr>
<td>5. Frame differenced image with adaptive thresholding</td>
<td>9</td>
</tr>
<tr>
<td>6. Original image and gradient image</td>
<td>17</td>
</tr>
<tr>
<td>7. GLCM example</td>
<td>18</td>
</tr>
<tr>
<td>8. Vehicle and detection image with a fragment (f2) inside of a fragment (f1)</td>
<td>25</td>
</tr>
<tr>
<td>9. Vertical fragments, Horizontal fragments, Diagonal fragments</td>
<td>28</td>
</tr>
<tr>
<td>10. The final result from the in-between fragment calculation</td>
<td>30</td>
</tr>
<tr>
<td>11. Pieces of gray scale, gradient, and detection images at the same location</td>
<td>32</td>
</tr>
<tr>
<td>12. Fragment pair that passes the post-processing check</td>
<td>33</td>
</tr>
<tr>
<td>13. Fragment Pair that will not pass the post-processing check</td>
<td>34</td>
</tr>
<tr>
<td>14. Graph to show area between a fragment pair calculation</td>
<td>36</td>
</tr>
<tr>
<td>15. Frame from scenario 1 video</td>
<td>43</td>
</tr>
<tr>
<td>16. Comparison of FAME implementations used in scenario 1</td>
<td>43</td>
</tr>
<tr>
<td>17. Frame from scenario 2 video</td>
<td>44</td>
</tr>
<tr>
<td>18. Comparison of FAME implementations used in scenario 2</td>
<td>44</td>
</tr>
<tr>
<td>19. Frame from scenario 3 video</td>
<td>45</td>
</tr>
</tbody>
</table>
LIST OF FIGURES (Continued)

20. Comparison of FAME implementations used in scenario 3. ...................... 45

21. Frame from scenario 4 video. .............................................................. 46

22. Comparison of FAME implementations used in scenario 4. ................... 46

23. Frame from scenario 5 video. .............................................................. 47

23. Comparison of FAME implementations used in scenario 5. ................... 47
<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Describes the four points to represent the area between the fragments</td>
<td>36</td>
</tr>
<tr>
<td>2. Results from scenario 1</td>
<td>43</td>
</tr>
<tr>
<td>3. Results from scenario 2</td>
<td>44</td>
</tr>
<tr>
<td>4. Results from scenario 3</td>
<td>45</td>
</tr>
<tr>
<td>5. Results from scenario 4</td>
<td>46</td>
</tr>
<tr>
<td>6. Results from scenario 5</td>
<td>47</td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

In the field of surveillance, algorithms must be developed to extract information from a camera's field of view. One of the algorithms used in the field of surveillance is a tracking algorithm. Tracking is the process of following the path of a moving object for the purpose of studying the object's behavior. While tracking is very useful, the design of the tracker can be very complicated. There have been many proposed designs for trackers. For example the histogram of gradients, or HOG, tracker uses features from a distribution of intensity gradients, or edge directions, to form detections to use in a tracker [15]. Another type of tracker is a kinematic tracker that uses motion of an object to determine how the object will be tracked. The tracker used in this research is a feature aided tracker, or FAT. A FAT uses both features and kinematic metrics to track an object. The FAT used for this research is made up of many different components.

The components of the FAT used in this research are the video input device, detector, feature noise filter, and associator. The video input device is the component that is responsible for reading in the video that is going to be tracked. For this research, gray scale video is being read in by the video input device. Once the gray scale video is read in by the video input device, the video stream is converted into images that can be passed to the detector. The detector takes the images from the video and forms detections. Detections used in this research are formed by using frame differencing to extract the foreground objects from the background. Other methods attempt to model the background using multiple prior images versus simple differencing. The justification for
using frame differencing over background modeling will be provided later in this thesis. The detections that are formed by the detector are then passed along to the feature noise filter. The feature noise filter will help decide which detections are likely to be a moving object and which detections are likely to be noise. The feature noise filter will extract features to judge if the detections are valid or not. The feature noise filter will then send the detections that belong to moving objects to the associator. The associator will then match the moving object detections with the set of current tracks. If there are no current tracks to associate the detections with, the associator will process a few frames before creating a new track from correlated detections. Also, if no track has a matching detection over a set number of frames, the associator will terminate the track. Figure 1 depicts the tracking process used by the FAT.

![FAT tracker flowchart](image)

**Figure 1:** FAT tracker flowchart

1.1 Terminology Used

The purpose of this section is to familiarize the reader with the terminology used in this thesis. The thesis contains a lot of terms found in the field of video tracking, computer vision, and pattern recognition to describe the creation of the Fragment Association Matching Enhancement (FAME) algorithm. The last word in the list was coined for this
research to describe what pieces of the image are being matched.

**Background:** is any object in the camera's field of view that does not move. An example of a background object would be a tree or a building.

**Foreground:** is any moving object in the camera's field of view. Foreground objects could be a vehicle or person.

**Frame:** is a single image extracted from a video sequence.

**False alarm:** is when the tracker determines a background object is actually a foreground object. In computer vision, this is called a false positive [25].

**Fragment:** is any partial foreground object that is being tested for matching to create a larger foreground object. For example, a partial foreground object would be half of a car extracted by the detector.

1.2 Problem

Typically, trackers are very sensitive devices. Any change in camera movement, or changes in lighting, may reduce the accuracy of the tracker. A moving camera will introduce more noise since some background objects will appear to be at a different location in the two frame differenced images. Changes in lighting can change the appearance of objects in an image by casting shadows across the image. Shadows being casted by a building could make tracking a vehicle more difficult since the shadows make part of the image dark. The darkness from the shadows would cause the difference of pixels from the two images to be very small. Both the camera movement and lighting will introduce more noise into the tracker. By introducing more noise, the tracker will
have a harder time maintaining a quality track due to the noise in the detections. One of the problems created by the added noise is the multi-fragmentation problem.

The multi-fragmentation problem is when an object is assigned with two tracks instead of one track. Having two tracks on the object instead of one track reduces the accuracy of the tracker by increasing the redundancy. The FAME algorithm developed in this research will work on correcting the multi-fragmentation problem. To correct the multi-fragmentation problem FAME will match smaller fragments with similar smaller fragments to create larger fragments. The obvious goal being that the larger fragment better represents a single true object (e.g., vehicle).

![Figure 2: (L-R) Good tracking and multi-fragmentation problem.](image)

1.3 Goal

The goal of this research is to propose a solution called FAME to solve the multi-fragmentation problem. The FAME algorithm is new solution to the multi-fragmentation problem that uses known feature calculation algorithm to match the pairs of fragments. The proposed FAME algorithm is implemented as part of the feature noise filter component. Once the noise is filtered out from the detector, small fragments are grouped into pairs and then sent to the FAME algorithm. FAME then uses a series of sequential filters to remove any incorrect fragment pairs that do not meet the requirement to be
considered a fragment match. Fragment pairs that pass all the filters in FAME will then be matched together to create larger fragments. These larger fragments can then be passed to the associator for matching with the current set of tracks. Figure 2 shows a flowchart of the FAT with the FAME algorithm developed from this research.

1.4 Layout of the Thesis
This thesis is divided into five different chapters. Chapter 1 gave an introduction to what the thesis will cover. Chapter 2 discusses related background work used to develop FAME. Chapter 3 describes the FAME algorithm used to match the fragments. The results generated by FAME are shown in chapter 4. Lastly, Chapter 5 concludes the research done for this thesis and provides recommendations for future research.
2.0 BACKGROUND

This chapter will be describing the different tracking, computer vision, and pattern recognition techniques used in developing the FAME algorithm. The techniques covered in this section range from image filtering, feature calculation, and distance calculation. Each of the features was tested to see how well they do in describing the similarity of two fragments. Each section will give an overview about the theory behind each technique.

2.1 Median Filtering

The median filter is an image smoothing filter that is used to remove noise from an image [18]. To apply a median filter to an image a filter kernel is passed over the image. To determine the intensity value of a median filtered pixel, the kernel sorts all the gray scale pixels from smallest to largest. Then the filter then chooses the pixel at the median location and sets the value of the filtered image pixel to the median pixel value from the kernel. The authors of [18] suggested using a kernel that would have an odd number of pixels to sort from. By having an odd number of pixels, there will be no confusion for the value of the median pixel value. Typically a kernel of size 3 or 5 would be used, but this size is problem dependent. In the FAME algorithm, the median filter is used to smooth the images that are extracted from the low resolution video. Smoothing the images helps remove any noise that is present in the image. By removing the noise, the FAT is able to extract more meaningful detections.
2.2 Frame Differencing vs. Background Modeling

In the area of tracking there are two main ways a tracker can form detections based on motion. The two main ways detections are formed is by using frame differencing and background modeling. The next two sections will describe frame differencing and background modeling. Also, section 2.2.2 will describe why frame differencing was chosen over background modeling in the FAT used in this thesis.

2.2.1 Frame Differencing

In order to detect the objects viewed by the tracker, the background and foreground must be separated. One of the most common ways to separate the background from the foreground is to do frame differencing. Frame differencing is also known as temporal differencing [4]. The idea behind frame differencing is that foreground objects can be extracted from the background by finding the absolute pixel difference, $DF$, between the current frame, $I(t)$, and the previous frame, $I(t-1)$.

$$DF(t) = |I(t) - I(t-1)|$$

After the frame differencing has been completed, the background objects should have a pixel intensity of zero and appear black. All the foreground objects will have an intensity greater than 0 and appear to be either gray or white. To create a binary foreground image, an adaptive threshold is applied under the assumption that foreground object will have a frame differencing value above the threshold and background objects will have a frame differencing value below the threshold. After the adaptive threshold is applied all the foreground objects are white (255) and the background objects are black (0). The adaptive thresholding attempts to remove small noise that might have been cause from
camera movement or changes in lighting. However, there are some drawbacks to frame differencing.

Any major change between the previous image and current image will reduce the effectiveness of frame differencing in forming detections for the FAT. Two such events that will reduce the effectiveness of frame differencing are changes in lighting and camera movement. When the lighting in the environment changes every pixel's intensity will change. This change in pixel intensity will cause the frame differencing to give bad values and be non-effective in extracting the foreground object. If a camera moves quickly, the frame differencing may fail unless this motion can be accurately estimated and removed. Also, frame differencing has a difficult time differencing two frames that contain a large object, say a bus. More than likely, a large bus will appear as two small cars. The reason this happens is when the previous frame is subtracted from the current frame part of the object will be in the same location in both the previous and current frame. By having part of the object located in the same position in both frames the difference of the two same intensities will be zero. That would then create the false appearance to the feature noise filter that the two small detections are two cars instead of being a bus.

Figure 4: Shows a zoomed in fragmented car.

The FAME algorithm described in this research will fix the problem caused by frame differencing frames with large objects. Even with these flaws, frame differencing was
still chosen over background modeling.

![Frame differenced image with adaptive thresholding.](image)

Figure 5: Frame differenced image with adaptive thresholding.

2.2.2 Frame Differencing over Background Modeling

The main reason why frame differencing was chosen over background modeling was how fast frame differencing can adapt to change compared to background modeling. In background modeling, the assumption is that the camera will remain relatively static and have very little movement [16]. The camera will have to stay focused on one area for a set amount of time to construct a model of the background. If the camera is significantly moved, the background model will have to start all over again learning the background. Alternatively, the moving images could be registered to satisfy the assumption of no movement however this would only apply to regions that overlap in the imagery. The FAT being used in this research has the ability to move freely and zoom in on targets. Since the camera is able move significantly, the background modeling is just not effective for the FAT. Even though camera movement does affect the performance of frame differencing, the time needed for frame differencing to recover to get good detections after camera movement is much less than the time needed by background modeling. The reason why frame differencing is able to adapt to camera movement faster than
background modeling is that there is no learning being done by frame differencing.
Instead only two frames, the previous and current frame, are used to extract the
detections.

2.3 Adaptive Thresholding
Adaptive thresholding is used to make sure all the frame differenced foreground pixels
have the same intensity of 255 and all the background pixels will have an intensity of 0.
The adaptive threshold will attempt to filter out any foreground pixels with small
intensity values that have been caused by camera movement. An adaptive threshold will
attempt to adjust to any change in lighting that may occur in the background. Since
changes in lighting are a major problem in a computer vision system, using adaptive
thresholding was better than just defining a static threshold. In order to calculate the
adaptive threshold the mean and standard deviation from the frame differenced image
needs to be calculated. Shown below is the equation used to apply the adaptive threshold
to the image [4]. The value $DF(i,j)$ is the pixel intensity value at a location $(i,j)$ in the
frame differenced image. The value $x$ is the number of pixels in the horizontal direction
of the image. The value $y$ is the number of pixels in the vertical direction. The $\mu_{DF}$ is the
mean of the frame differenced image. The $\sigma_{DF}$ is the standard deviation of the frame
differenced image. The value $T(t)$ is the adaptive threshold value for the frame
differenced image.

$$
\mu_{DF} = \frac{\sum_{i=0}^{x} \sum_{j=0}^{y} DF(i,j)}{(x \times y)}
$$
\[
\sigma_{DF} = \frac{\sum_{i=0}^{X} \sum_{j=0}^{Y} (DF(i,j) - \mu_{DF})}{(x \ast y)}
\]

\[
T(t) = (w_1 \ast \mu_{DF}) + (w_2 \ast \sigma_{DF})
\]

By added weights \(w_1\) and \(w_2\) to the adaptive threshold, the threshold can be influenced more by the mean or the standard deviation, respectively. Larger weighting of the mean increases the mean used in the adaptive thresholding calculation. Larger weighting of the standard deviation allows the threshold to be farther from the mean.

2.4 Euclidean Distance

In FAME, the distance between the two fragment's centroid is calculated by the Euclidean distance, or L2-norm. The Euclidean distance will describe how many pixels there are between the two centroids. Euclidean distance is represented by [17]. Where the \(x\) is one fragment and the \(y\) is the second fragment.

\[
d(x, y) = \sqrt{\sum_{i=1}^{l} (x_i - y_i)^2}
\]

In order to find the fragment’s centroid, first order moments of a fragment are used to find the average \(X\) and \(Y\) location values of the fragment. Moments are used to provide different levels of numerical information about an image. An image can be reconstructed from all the orders of the moments [17]. A moment is defined by [12]

\[
m_{pq} = \int \int x^p y^q f(x, y) dx dy
\]

The order of the moment is determined by adding the \(p\) and \(q\) variable in the moment \(m_{pq}[12]\). Since only first order moments are required to find the center of the fragment,
moments $m_{00}$, $m_{10}$, and $m_{01}$ need to be calculated.

\[
m_{00} = \int \int dx dy b(x, y) \]

\[
m_{10} = \int \int dx dy x f(x, y) \]

\[
m_{01} = \int \int dx dy y f(x, y) \]

A moment is a first order moment when the sum of the $p$ and $q$ variable equals 1. This also applies when calculating second and third order moments. However, when calculating moments from a digital image the integrals are discretized with summations [17].

\[
m_{pq} = \sum_{i=0}^{X} \sum_{j=0}^{Y} I(i,j)i^p j^q \]

Where $I(i,j)$ is the pixel intensity value at a location $(i, j)$. The averages for $X$ and $Y$ are described by $\bar{X}$ and $\bar{Y}$. The calculation for the $\bar{X}$ and $\bar{Y}$ are shown below.

\[
\bar{X} = m_{10} - m_{00} \]

\[
\bar{Y} = m_{01} - m_{00} \]

Once the centroids of the fragment have been determined the Euclidean distance between the two fragments can be calculated. Say there are two fragments $f_1$ and $f_2$. Each fragment will have a centroid described by $(\bar{X}, \bar{Y})$. The Euclidean distance calculation is then shown below.

\[
f_1 \in (\bar{X}_1, \bar{Y}_1), \ f_2 \in (\bar{X}_2, \bar{Y}_2) \]

\[
d(f_1, f_2) = \sqrt{\sum_{i=1}^{2} (f_{1i} - f_{2i})^2} = \sqrt{(f_1(X_1) - f_2(X_1))^2 + (f_1(Y_2) - f_2(Y_2))^2} \]
2.5 Eccentricity

Eccentricity is used to determine if a fragment is circular or ellipsoidal. Eccentricity ranges between 0 and 1. When the eccentricity is low the fragment will look like a circle. A fragment with a higher eccentricity would be closer to an ellipsoidal shape. One way to calculate the eccentricity is to use second order central moments. All central moments are reduced from spatial moments and refer to the center of gravity [12]. To calculate central moments of the first order or higher the following equation can be used [12].

\[ \mu_{pq} = \frac{m_{pq}}{m_{00}} - \left( \frac{m_{10}}{m_{00}} \right)^p \left( \frac{m_{01}}{m_{00}} \right)^q \]

From the general central moment equation, the central moments needed by the eccentricity can be calculated [12].

\[ \mu_{20} = \frac{m_{20}}{m_{00}} - \left( \frac{m_{10}}{m_{00}} \right)^2 \]
\[ \mu_{02} = \frac{m_{02}}{m_{00}} - \left( \frac{m_{01}}{m_{00}} \right)^2 \]
\[ \mu_{11} = \frac{m_{11}}{m_{00}} - \left( \frac{m_{10}}{m_{00}} \right) \left( \frac{m_{01}}{m_{00}} \right) \]

The eccentricity, \( e \), is calculated by [12]

\[ e = \frac{(\mu_{20} - \mu_{02})^2 - 4\mu_{11}^2}{(\mu_{20} - \mu_{02})^2} \]

2.6 Hu Moments

Hu moments are used in this thesis to compare how similar two fragments are. There are seven different types of Hu moments that are invariant under the actions of translation,
scaling, and rotation [17]. Only the seventh Hu moment is affected by reflection [17].

Hu moments can be used for feature matching in many different pattern recognition problems. If we combine the Hu moments into a set or vector of moments, we can compare how well two sets match. To determine how well two sets of Hu moments match, FAME uses the sum of differences between the two sets. The pair of features with the smallest sum of differences will be the most similar based on Hu moments. The pair of Hu moments with the largest sum will be the least similar. However, in order to calculate the Hu moments the normalized, \( n_{pq} \), second and third moments must be calculated [17].

\[
n_{pq} = \frac{\mu_{pq}}{\mu_{00}^p} = \left( \frac{p + q + 2}{2} \right)
\]

The calculation of the central moments, \( \mu_{pq} \), was described previously in section 2.5.

The seven different Hu moments, \( \theta \), are listed below.

\[
\begin{align*}
\theta_1 &= n_{20} + n_{02} \\
\theta_2 &= (n_{20} - n_{02})^2 + 4n_{11}^2 \\
\theta_3 &= (n_{30} - 3n_{12})^2 + (n_{03} - n_{21})^2 \\
\theta_4 &= (n_{30} + n_{12})^2 + (n_{03} + n_{21})^2 \\
\theta_5 &= (n_{30} - 3n_{12})(n_{30} - n_{12})[(n_{30} - n_{12})^2 - 3(n_{21} - n_{03})^2] \\
&\quad + (n_{03} - 3n_{21})(n_{03} - n_{21})[(n_{03} - n_{21})^2 - 3(n_{12} - n_{30})^2] \\
\theta_6 &= (n_{20} - n_{02})[(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2] + 4n_{11}(n_{30} + n_{12})(n_{03} + n_{21}) \\
\theta_7 &= (3n_{21} - n_{03})(n_{30} - n_{12})[(n_{30} - n_{12})^2 - 3(n_{21} - n_{03})^2] \\
&\quad + (n_{30} - 3n_{12})(n_{21} - n_{03})[(n_{03} - n_{21})^2 - 3(n_{30} - n_{12})^2]
\end{align*}
\]
2.7 Bhattacharyya Distance

The Bhattacharyya distance compares the histograms of two fragments to describe how similar two fragments are. Before the Bhattacharyya distance can be calculated, the Bhattacharyya coefficient must be determined. The Bhattacharyya coefficient shows how much similarity there are between the two fragments. The Bhattacharyya coefficient is shown below by $\rho$ [5]. Where $p_1(x)$ is a histogram from one fragment and $p_2(x)$ is the histogram of the other fragment [7].

$$\rho = \int \sqrt{p_1(x)p_2(x)} \, dx$$

The Bhattacharyya distance, $b$, can then be calculated using the Bhattacharyya coefficient $\rho$.

$$b = \sqrt{1 - \rho} = \sqrt{1 - \int \sqrt{p_1(x)p_2(x)} \, dx}$$

However, when using the Bhattacharyya distance and coefficient on a digital image the integration must be changed to summations. So in a digital image the Bhattacharyya coefficient becomes

$$\rho = \sum_{i=1}^{B} \sqrt{p_1(x)p_2(x)}$$

and the Bhattacharyya distance becomes [7]. The $B$ used in the summations represents the number of bins in the histogram.

$$b = \sqrt{1 - \rho} = \sqrt{1 - \sum_{i=1}^{B} \sqrt{p_1(x)p_2(x)}}$$
2.8 Sobel Edge Detection

The Sobel operator is used to calculate an approximation of a gradient in the horizontal and vertical direction [13]. To calculate the gradient in the horizontal and vertical direction, two separable convolution kernels, $G_x$ and $G_y$, are used. The $G_x$ kernel is applied to the image to get the gradient in the vertical direction. The $G_y$ kernel is applied to the image in to get the gradient in the horizontal direction. For this thesis the Scharr convolution kernel was chosen over the Sobel convolution kernel. The reason why the Scharr convolution kernel was chosen over the Sobel convolution kernel is the Scharr convolution kernel gave more accurate results than the Sobel [26]. Once the convolution kernels have been applied to the image, the absolute gradient magnitude is used to calculate a gradient image that contains the edges from the image. The absolute gradient magnitude is calculated by convolving the two images that were created by applying the Scharr convolution kernels $G_x$ and $G_y$. A high gradient value indicates a major change in the image [13]. These high gradient values correspond to a definite edge between two boundaries in an image. An example of a strong boundary generated by the edge detector would be a white car driving on a black topped road. Since the color of the car is so different than the road, a strong edge will be generated in the edge detection image

\[
G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \quad G_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}
\]

The equation to calculate the absolute gradient magnitude on an image $I$ with convolution kernel $G_x$ and $G_y$ is:
gradient = $\sqrt{(I \ast G_x)^2 + (I \ast G_y)^2}$

Figure 6: (L-R) Original image and gradient image.

2.9 Gray Level Co-Occurrence Matrix (GLCM)

A GLCM is used to determine how different neighboring pixel's intensities are from one another. A neighboring pixel is determined by the offset used to calculate the GLCM. An offset is just the distance of the neighboring pixel to the current pixel. The four main offsets to calculate a GLCM are 0 degrees (0,1), 45 degrees (1, -1), 90 degrees (-1,0), and 135 degrees (-1,-1) [11]. Each offset will give a different GLCM. To calculate the GLCM another matrix $P$ will be calculated for an image $I$ that is of size $N \times N$. The $N \times N$ size matrix is usually the size of the pixel's intensity in the image [11]. To calculate the value of the matrix $P$ the following equation was used [11].

$$P(i, j) = \sum_{x=1}^{N} \sum_{y=1}^{N} \begin{cases} 1, & \text{if } I(x, y) = i \text{ and } I(x + \Delta x, y + \Delta y) = j \\ 0, & \text{otherwise} \end{cases}$$

The first $I$ in the equation is the current location and the second $I$ is the offset location.
The 0 degrees offset will check the current pixel with its closest horizontal neighbor. The 90 degrees offset checks the current pixel with its vertical neighbor. The right diagonal is checked by the 45 degrees offset and the left diagonal is checked by the 135 degrees. Figure 5 shows how to calculate a GLCM using a 0, 45, 90, and 135 degrees offset.

![Input Image](image)

<table>
<thead>
<tr>
<th>Horizontal GLCM 0</th>
<th>Vertical GLCM 90</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Horizontal GLCM Example" /></td>
<td><img src="image" alt="Vertical GLCM Example" /></td>
</tr>
</tbody>
</table>

Figure 7: GLCM example.

### 2.9.1 GLCM features

From the GLCM, 14 different Haralick features can be calculated to numerically describe the pixels intensities in the GLCM. There are a few calculations that must be determined before a Haralick feature can be calculated. For the calculations, \( p(i, j) \) will represent an element in the normalized co-occurrence matrix [22]. The number of dimensions of the matrix will be represented by \( N_g \) [22]. The values \( p_x(i) \) and \( p_y(j) \) will represent the marginal probabilities [22]. Lastly, the mean and variance will need to be calculated before the Haralick features can be calculated.
\[p_x(i) = \sum_{j=1}^{N_g} p(i,j), \quad p_y(j) = \sum_{i=1}^{N_g} p(i,j)\]

\[\mu_x = \sum_{i=1}^{N_g} i p_x(i), \quad \mu_y = \sum_{j=1}^{N_g} j p_y(j),\]

\[\sigma_x = \sqrt{\sum_{i=1}^{N_g} p_x(i)(i - \mu_x)^2}, \quad \sigma_y = \sqrt{\sum_{j=1}^{N_g} p_y(j)(j - \mu_y)^2}\]

\[p_{x+y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j), \quad i + j = k, \quad k = 2,3,\ldots,2N\]

\[p_{x-y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j), \quad i - j = k, \quad k = 0,1,\ldots,N - 1\]

\[H_{XY1} = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log(p_x(i)p_y(j))\]

\[H_{XY2} = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p_x(i)p_y(j) \log(p_x(i)p_y(j))\]

\[H_{XY} = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log(p(i,j))\]

\[H_X = -\sum_{i=1}^{N_g} p_x(i) \log(p_x(i)), \quad H_Y = -\sum_{j=1}^{N_g} p_y(j) \log(p_y(j))\]

\[Q(i,j) = \sum_{k=1}^{N_g} \frac{p(i,k)p(j,k)}{p_x(i)p_y(j)}\]
From these preliminary calculations, the 14 Haralick features can be calculated. All of the Haralick feature equations came from [20].

\[
\text{ASM} = f_1 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)^2
\]

\[
\text{CONTRAST} = f_2 = \sum_{n=0}^{N_g-1} n^2 \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) (i-n) (j-n)
\]

\[
\text{CORRELATION} = f_3 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \frac{(i-\mu_x)(j-\mu_y)p(i,j)}{\sigma_x \sigma_y}
\]

\[
\text{VARIANCE} = f_4 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} (i-\mu)^2 p(i,j)
\]

\[
\text{IDM} = f_5 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \frac{1}{1+(i-j)^2} p(i,j)
\]

\[
\text{SUM AVERAGE} = f_6 = \sum_{k=2}^{2N_g} k \cdot p_{x+y}(k)
\]

\[
\text{SUM VARIANCE} = f_7 = \sum_{k=2}^{2N_g} (k - f_6)^2 p_{x+y}(k)
\]

\[
\text{SUM ENTROPY} = f_8 = - \sum_{k=2}^{2N_g} p_{x+y}(k) \log(p_{x+y}(k))
\]

\[
\text{ENTROPY} = f_9 = - \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log(p(i,j))
\]

\[
\text{DIFFERENCE VARIANCE} = f_{10} = \text{variance of } p_{x-y}
\]

\[
\text{DIFFERENCE ENTROPY} = f_{11} = - \sum_{i=1}^{N_g} p_{x-y}(i) \log(p_{x-y}(i))
\]

\[
\text{INFORMATION MEASURES OF CORRELATION} = f_{12} = \frac{(HXY-HXY1)}{\max(HX, HY)}
\]

\[
\text{INFORMATION MEASURES OF CORRELATION} = f_{13} = \frac{1}{\exp[-2.0(HXY2 - HXY)]}
\]

\[
\text{MAXIMAL CORRELATION COEFFICIENT} = f_{14} =
\]

\[
\sqrt{\text{second largest eigen value of } Q}
\]

The research for this thesis focused on three Haralick features that are used in the feature
calculation for FAME. Since some of the Haralick features would not provide useful information they were left out of the research. The three Haralick features used in this research are: angular second moment (ASM), inverse different moment (IDM), and correlation. ASM measures the image homogeneity [22]. ASM is high when the image has very good homogeneity, or when pixels are very similar. However, if the pixels are very different than the ASM will be really low. IDM is very similar to the ASM in the sense that IDM returns a value of how homogenous the GLCM is [22]. IDM is different than ASM in the since that IDM is a weighted value. The weight, $1 + (i - j)^2$, used by IDM will give smaller homogenous values to pixel farther away and higher homogenous values to pixels that are closer. IDM still give high results for images that are similar and low results for images that are dissimilar. Correlation measures the linear dependency of gray levels of neighboring pixels [22]. In other words, correlation describes how dependent the rows and columns are of the GLCM.

2.10 Image Registration

In the tracker, image registration is used to align two frames taken at different time intervals. Usually, the current and the previous frame are used for image registration. The purpose of image registration is to align the frames before frame differencing. Anytime a camera is able to move, additional noise will be added into the frame differenced detection image. The goal is the image registration is to reduce the noise in the differenced image being used to extract object detections. Once the two images are aligned, the tracker can perform motion detection to how much the images should be transformed. Two image registration techniques that extract features for registration are
scale invariant feature transform (SIFT) and speeded-up robust feature (SURF) [19]. SIFT uses key points from the gradient direction in an image to find the foreground objects [15]. These key points are obtained by transforming the image into a scale-invariant space. Once the images have been transformed, key features in both images are extracted. After the weak features have been removed, the strong features in the two images are matched together. There have been many different versions of SIFT that have been proposed for different types of images [19]. SURF is an improved, and faster, version of SIFT. There are two main parts to the SURF image registration technique [23]. The first part of SURF applies a Laplacian of Gaussian to the integral image to get the key points. The key points are not affected by scale or invariance. In the second phase a square window is placed around each key point. The window creates a feature descriptor around each point using Haar wavelets. The K-nearest neighbor algorithm can be used to match which SURF descriptors are related in two images.
3.0 FAME

The purpose of the FAME algorithm is to pass a fragment pair through multiple filters to determine if the fragment pair should be matched together. Each filter will remove any incorrect fragment pair. These filters are executed sequentially to ensure that all the incorrect fragment pairs are removed. The sequential filters used to create FAME are: check the eligibility of the fragment pairs, score the eligible fragment pairs, and match the fragment pairs. Figure 3, in section 1.3, shows the flow of data through the FAME algorithm. The FAME algorithm described in this section was developed using the C++ programming language on a Linux operating system. The open source library OpenCV was also used to implement some of the computer vision and pattern recognition algorithms [24]. The FAME algorithm does require some information before the fragment pairs can be processed.

3.1 FAME Requirements

Each filter in the FAME algorithm requires a user defined parameter. The “Check Eligibility of the Fragment Pairs” requires an integer value describing what the largest allowable pixel distance is between two fragments. The “Score the Eligible Fragment Pairs” filter requires a float value to describe the largest allowable score for a fragment pair. Lastly, the “Match Fragment Pairs” filter requires an integer value to ensure that the pair of fragments is being correctly matched. Also, FAME expects to receive two images of the same size from the detector. The first image FAME expects to receive from the
detector is a binary detection image containing the foreground and background objects. Second, FAME will expect to receive an image used in the frame differencing. This second image will be used for feature calculations. For this research the frame differenced image being passed to FAME is a gray scale image. By having images of the same size, FAME can easily compare the same region in the binary image to a region in the gray scale image when calculating features.

3.2 Checking Eligibility of Fragment Pairs

This section will begin the description of the FAME algorithm. The “Check Eligibility of Fragment Pairs” filter is responsible for two tasks. The first task is to extract all the foreground objects, which are being called fragments, and store them to be used in the FAME algorithm. Second, all the fragments will be paired with one another. Once a fragment is paired with another fragment, then the Euclidean pixel distance between the two fragments will be calculated. If a fragment pair passes the user defined distance threshold, the fragment pair will be passed to the next filter.

3.2.1 Extract Foreground Objects

To extract the fragments from the binary detection image, FAME will use the OpenCV findContours routine. The findContours is a border following routine that finds, and marks, all the foreground objects in the image. The purpose of the findContours routine in FAME is to find the location of the fragments in the binary detection image. When finding the fragments, findContours stores the locations of the points used to outline the fragment in \((x, y)\) pixel coordinates. The function findCountours has the ability to create
a hierarchy of all the fragments in the image. Any contour inside of the image is on the first level of the hierarchy. It is possible that a contour on the first level can have a hole that contains another contour. This contour inside of the first hierarchy is then placed on the second level of the hierarchy since the contour is inside of another contour. Figure 8 shows two fragments \( f1 \) and \( f2 \). The fragment \( f1 \) is on the first level of the hierarchy since \( f1 \) is closest to the image border. Fragment \( f2 \) is on the second level of the hierarchy since \( f2 \) is surrounded by \( f1 \).

![Figure 8: (L-R) Vehicle and detection image with fragment (f2) inside of fragment (f1).](image)

3.2.2 Calculating Distance Between Fragments

After the fragments have been extracted using findContours, the distance can be calculated between each pair of fragments. FAME uses the centroid of each fragment to calculate the distance between a fragment pair. In order to find the centroids of the fragments, FAME will use first order moments (2.4) to locate the average \( x \) and \( y \) value of all the pixels located inside the fragment. The average \( x \) and \( y \) location will then be marked as the centroid of the fragment. The Euclidean pixel distance between fragments will then be checked against the user defined distance threshold. If the distance is less than the threshold, then the fragment pair can move on to the next filter. However, if the distance is greater than the threshold the fragment pair will no longer be considered an eligible fragment pair for matching.
3.3 Scoring Eligible Fragment Pairs

The scoring section gives each fragment pair a score that represents the similarity of the fragments. The fragment pair score is based on a sum of normalized features with values between 0 (similarity) and 1 (different). For some of the features, the region between the two fragments is used for feature calculation. The region between the two fragments is called the mid-region.

3.3.1 Mid-region Calculation using Arc Tangent Function

In the binary detection image, the mid-region of two fragments is determined to be background. So in order to extract the mid-region of two fragments from an image this function was developed. The goal of the “Mid-region Calculation using Arc Tangent Function” was to use information from the fragment pair to calculate the mid-region. The OpenCV findContours routine returns a rectangle that contains the pixels in and around the fragment. This rectangle contains the upper left point in \((x, y)\) pixel coordinates, the width and the height of the rectangle in pixels. The four points of the rectangle can then be created by using a combination of the upper left point and offsetting the \(x\) coordinate with the height and the \(y\) coordinate with the width. The “Mid-region Calculation using Arc Tangent Function” will use these four points to calculate the mid region.

Say there are two fragments \(f1\) and \(f2\). Each fragment will have four corners, \(c1\), \(c2\), \(c3\), \(c4\), in \((x, y)\) pixel coordinates used to represent the rectangle around the fragment. The “Mid-region Calculation using Arc Tangent Function” will determine which 2 pairs of points will have the shortest Euclidean distance.

\[
c1=[x1, y1], \ c2=[x2, y2], \ c3=[x3, y3], \ c4=[x4, y4];
\]

\[
f1=[c1, c2, c3, c4];
\]
\( c_5 = [x_5, y_5], c_6 = [x_6, y_6], c_7 = [x_7, y_7], c_8 = [x_8, y_8] \);
\( f_2 = [c_5, c_6, c_7, c_8] \);
shortest = 1000;
secondShortest = 1000;
shortestIndex[0, 0];
secondShortestIndex[0, 0];

for(i = 0; i < f1.length; i++)
{
    for(h = 0; h < f2.length; h++)
    {
        d = dist(f1[i], f2[h])
        if(d < shortest)
        {
            // check to see if secondShortest needs updated
            if(shortest < secondShortest)
            {
                secondShortest = shortest;
                secondShortestIndex[0] = shortestIndex[0];
                secondShortestIndex[1] = shortestIndex[1];
            }
            shortest = d;
            shortestIndex[0] = i;
            shortestIndex[1] = h;
        } else if(d < secondShortest)
        {
            secondShortest = d;
            secondShortestIndex[0] = i;
            secondShortestIndex[1] = h
        }
    }
}

The two pairs of shortest distance points will then be used to define the four corners of the mid-region. Next, a rotation angle will be calculated to help rotate the mid-region if the two fragments are diagonal to one another. Figure 9 shows an example of having diagonal, horizontal, and vertical fragments. The yellow rectangles in Figure 9 represent the fragments.
From the 2 pairs of points, the closest and farthest points from the origin of the detection image are calculated. The origin is defined by where the pixel location is \((0, 0)\) in the detection image being passed by the detector. In the OpenCV library, the \((0, 0)\) location, or origin, of the image is located in the upper left corner. The closest point is defined by the point that has the shortest Euclidean pixel distance to the origin. The farthest point is defined by the point largest Euclidean pixel distance away from the origin. Knowing the information calculated from the fragments \(f_1\) and \(f_2\) during the shortest distance calculation, the closest point, \(closestPt\), and farthest point, \(farthestPt\), can be calculated from the points \(p_1, p_2, p_3, p_4\) used in the 2 pairs of closest points.

```c
//copy the shortest and second shortest distance points using the previously calculated indexed
p1 = f1[shortestIndex[0]], p2 = f2[shortestIndex[1]];  
p3 = f1[secondShortestIndex[0]], p4 = f2[secondShortestIndex[1]];  
pts=[p1,p2,p3,p4];
closet=1000;  
farthest=0;  
origin=[0,0];  
closestPt=[0,0];  
farthestPt=[0,0];
```
for(i=0; i < pts.length; i++)
{
    d=dist(pts[i], origin)
    if(d<closestPt)
    {
        closet=d;
        closestPt =pts[i];
    }
    if(d > farthest)
    {
        farthest=d;
        farthestPt=pts[i];
    }
}

Using the closest and farthest point, the rotation angle of the mid-region was calculated using the arc Tangent function. Where \((X_f, Y_f)\) is the location of the farthest point, \(farthestPt\), and \((X_c, Y_c)\) is the location of the closest point, \(closestPt\).

\[
\Delta Y = |Y_f - Y_c|
\]
\[
\Delta X = |X_f - X_c|
\]
\[
\theta = arctan\left(\frac{\Delta Y}{\Delta X}\right) \times \left(\frac{180}{\pi}\right)
\]

The mid-region could then be calculated using the four points from the two pairs of shortest distance points and the rotation angle. To rotate the mid-region the midpoint of the mid-region must be calculated using the four points \(p1, p2, p3,\) and \(p4\) from the 2 pairs of closet points.

\[
midX = \frac{p1.x + p2.x + p3.x + p4.x}{4}
\]
\[
midy = \frac{p1.y + p2.y + p3.y + p4.y}{4}
\]

\(Midpoint = (midX, midY)\)
Then the mid-region uses the midpoint as a pivot to rotate the mid-region by the calculated angle. Using the rotation angle prevented overestimating pixel in the mid-region. Figure 10 shows two fragments and the calculated mid-region. The two red rectangles represent a pair of fragments. The rotated green rectangle represents the mid-region calculated in this function. However, in order for OpenCV to process the pixels in the green rectangle a bounding box must be placed around the rotated mid-region. The bounding box used to process the mid-region is represented in yellow.

![mid-region](image)

> Figure 10: The final result from the in-between fragment calculation.

### 3.3.2 Scoring Fragment Pair

The scoring of the fragment pair is used to represent how likely a fragment pair is to be a match. The equation below describes the formula used to calculate the score.

$$S(B_1, B_2) = \sum_{i=0}^{N} \frac{F(i)}{N}$$

In the score equation, $B_1$ and $B_2$ represent the fragment pair being scored. The $F(i)$
represents the normalized feature in the score and \( N \) represents the number of features in the score. The scoring equation is very flexible since one or more features can be used to determine the similarity of a fragment pair. During the testing of FAME, different combinations of features were used in the score calculation. One such combination of features was: Hu moments (2.6), Haralick features (2.9.1), and eccentricity (2.5). If a score is lower than the user defined score threshold value, the fragment pair is considered “similar” and eligible for matching.

3.4 Match Pairs that are Above Threshold

To match a pair of fragments, a line is drawn to connect the two fragments together. The line used to connect the fragments goes from one fragment’s centroid to the other fragment’s centroid. By matching the fragments together two or more small fragments can turn into one large fragment. For example two small fragments could represent the front and rear of a bus. By matching the two fragments together FAME matches the buses front with the rear to create the whole bus. If a fragment is shared by multiple fragment pairs then the fragment will contain multiple lines connecting to other fragments. By allowing a fragment to connect too many other fragments there is a chance that two close fragments pairs could just become one large fragment. However, if multiple fragments were not connected together a large fragment consisting of three smaller fragments would never be created. The “Fragment Match Check” (3.4.1) was created to insure two close fragments would not be matched together even if their score was below the allowable score value.
3.4.1 Fragment Match Check

To ensure the fragments are a correct match FAME creates an edge image from the gray scale image using the absolute gradient magnitude (2.8). This edge image will be the same size as the gray scale image, which in turn make the gradient image the same size as the binary detection image. Figure 11 shows a piece of the gray scale, gradient, and detection image.

![Figure 11](image)

Figure 11: (L-R) Pieces of gray scale, gradient, and detection images at the same location.

Since all the images are the same size, in Figure 11 the gradient image shows that there is an edge between the two fragments in the detection image. In a gradient image, edges have a high pixel intensity if the edge is very definitive in the gray scale image. An example of a definitive edge is a white car on a black road. The edge image is used to make sure two fragments are a correct match by extracting the previously calculated mid-region from the edge image. A large object should have a very definitive edge. That edge should be present in the pixel location of the two fragments and the mid-region. If there is no edge in the mid-region then the two fragments do not belong to a large object. To determine if an edge is in the mid-region, the average pixel intensity of all the pixels in the mid-region is calculated. If the average pixel intensity is greater than previously defined user allowable gradient value then the fragments will be a matched and passed
along to the associator. If the average pixel intensity is less than the user defined allowable gradient value, the fragments will not be matched. Figure 12 shows a scenario when a fragment pair will be matched. The two black lines represent the mid-region in both the detection image and gradient edge image. Since the mid-region in the edge image has an edge the fragments will be matched and passed along to the associator.

Figure 12: Fragment pair that passes the post-processing check.

Figure 13 shows a situation when the fragments will not be matched. The fragments will not be matched in Figure 13 since there is no edge in the mid-region. During the development of FAME there were some functions that were created but never used in the final version.
Figure 13: Fragment Pair that will not pass the post-processing check.

3.5 Region Calculation Function

The concept describe in this section was developed early in the FAME research. The “Region Calculation Function” was intended to calculate the region between two fragments. This region calculation function was the precursor to the “Mid-region Function using Arc Tangent” described in section 3.3.1. Like the mid-region function (3.3.1), the region calculation function uses the four points from the rectangle around the fragment to calculate the region between the two fragments. The region between the fragments is defined by the pair of points from both fragments in the pair, just like in 3.3.1. The main difference between 3.3.1 and “Region Calculation Function” is how the region between the two fragments is calculated.

The “Region Calculation Function” uses the two pairs of shortest distance points to create a new set of points to define the region between the two fragments. From two pairs of shortest distance points, the $x$ and $y$ closest to zero and the $x$ and $y$ farthest from zero will be copied into a new set of points. A point $P_{cp,f}$ contains a value $(x_{cp,f}, y_{cp,f})$. 

$\text{34}$
The subscript \( cp \) is the closest pair point number and \( f \) is the fragment number. Say the closest point pair, \( C_{p1} \), contains point \( p1,1 \) from fragment \( f1 \) and point \( p1,2 \) from \( f2 \). The second closest pair of points, \( C_{p2} \), contains point \( p2,1 \) from fragment \( f1 \) and point \( p2,2 \) from fragment \( f2 \).

\[
\begin{align*}
P_{1,1} &= [x1,1, y1,1], \ p_{1,2} = [x1,2, y1,2]; \\
C_{p1} &= [p1,1, p1,2]; \\
P_{2,1} &= [x2,1, y2,1],\ p_{2,2} = [x2,2, y2,2]; \\
C_{p2} &= [p2,1, p2,2]; \\
smallest X &= 1000, \ smallest Y = 1000; \\
largest X &= 0, \ largest Y = 0; \\
\end{align*}
\]

//check to see if the smallest and largest x, y are in the first closest pair of point
for (i = 0; i < cp1.length; i++) {
  if (cp1[i].x < smallestX) smallestX = cp1[i].x;
  if (cp1[i].x > largestX) largestX = cp1[i].x;
  if (cp1[i].y < smallestY) smallestY = cp1[i].y;
  if (cp1[i].y > largestX) largestX = cp1[i].y;
}

//using the results from the first for loop, check to see if the smallest and largest x, y are in the \( C_{p2} \)
for (i = 0; i < cp2.length; i++) {
  if (cp2[i].x < smallestX) smallestX = cp2[i].x;
  if (cp2[i].x > largestX) largestX = cp2[i].x;
  if (cp2[i].y < smallestY) smallestY = cp2[i].y;
  if (cp2[i].y > largestX) largestX = cp2[i].y;
}

Once the smallest and largest \( x \) and \( y \) are determined, four points will be created using a combination of the smallest and largest \( x \) and \( y \).
Table 1: Describes the four points to represent the area between the fragments.

<table>
<thead>
<tr>
<th>Point number</th>
<th>Point location ((x, y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>((\text{smallestX, smallestY}))</td>
</tr>
<tr>
<td>p2</td>
<td>((\text{smallestX, largestY}))</td>
</tr>
<tr>
<td>p3</td>
<td>((\text{largestX, smallestY}))</td>
</tr>
<tr>
<td>p4</td>
<td>((\text{largestX, largestY}))</td>
</tr>
</tbody>
</table>

Figure 14 gives a visual representation of the region calculated between the two fragments. The two fragments shown in Figure 11 are diagonal to one another.

![Middle Fragment Calculation](image)

Figure 14: Graph to show area between a fragment pair calculation.

While testing this region calculation function a flaw was discovered. This region calculation function overestimated the region between the two fragments when the fragments are in a diagonal shape. Figure 9 shows the possible positions of a fragment pair. If the fragments were horizontal, or vertical, then this “Region Calculation Function” worked. However if the fragments were diagonal to one another, then there
would be an overestimate of the region between the two fragments. The overestimation is not desirable since the pixels in the overestimated region could reduce the accuracy of the feature calculations and “fragment matching check” filter by introducing un-needed pixels.
4.0 RESULTS

The data chosen to test the FAT with the FAME algorithm was very important. Each data set contained at least two cars of any size or color. These multiple cars had to have the ability to drive close together, change speed, and direction. These multiple cars tested on how well FAME would match the fragments to the correct corresponding car. Also, the data used to test FAME had to come from a moving camera. The data from the moving camera tests FAME's ability to remove noise from the detection image that might be caused from the camera's movement. The program COMPASE Tracker Evaluation Software Suite (CTESS) was used to test how well FAME preformed on the FAT [21]. CTESS provides a robust set of tracking metrics along with visualization of the tracker’s performance results. Testing the results from the FAT using CTESS required two sets of data.

The first set of data was the track points generated by the FAT. These track points were used to represent the location of the object being tracked by the FAT in \((x, y)\) image pixel space. The second set of generated data was the truth points. The truth points are the actual locations of the moving object in \((x, y)\) image pixel space. Once CTESS has those two sets of data, a frame from the track points was compared to a frame from the truth points. Since CTESS compares frame-to-frame from each data set both the track points and the truths points must be taken at the same frame rate. One way CTESS analyzes the data is by checking if the track points are relatively close to the truth points.
based on a pixel distance threshold. If a track point is within the pixel threshold distance of the truth point, the track point will be matched to the truth point. Also, if two track points are close to a single truth point, the two track points will be matched to the truth point. Having the two track points matched to a single truth point is sub-optimal since that will increase the redundancy [21]. If there is no truth point within the pixel threshold of a track point that track point will be marked as spurious [21]. The following terms will be used to describe the performance of all three of the implementations of FAME [21]:

**Completeness:** I the fraction of associated track points and the truth points.

\[
\frac{\text{# of associated track points}}{\text{# of truth points}}
\]

**Purity:** Is the fraction of the dominant track points over all the associated track points.

The dominant track is the track that consists of the most track points.

\[
\frac{\text{# of track points used by dominant track}}{\text{# of associated track points}}
\]

**Redundant:** Is the fraction of redundant track point associations made to the truth points. A redundant point occurs when two track points are matched to a single truth point.

\[
\frac{\text{# of redundant track point associations}}{\text{# of truth points}}
\]

**Spuriousness:** Represents how many false alarms were in the track points. A false alarm
is generated when there are no truth points to be associated with the track point. The spuriousness is calculated by taking the fraction of spurious points over all the truth points.

\[
\frac{\text{# of spuriousness points}}{\text{total # of truth points}}
\]

4.1 Experiment

The purpose of these experiments was to see how much FAME would improve the performance of the FAT using five different test scenarios. The FAT with FAME will be considered improved if the following criteria are met. The first criterion is that the FAT using FAME must have a higher completeness and purity than the FAT not using FAME. Meeting the first criteria would mean that more of the tracks were a match to the truth points using the FAT with FAME instead of the FAT without FAME. The second criterion is that the redundancy and spuriousness must be less on the FAT with FAME compared to the FAT without FAME. By reducing the redundancy and spuriousness, the FAT using FAME will show the ability to have less incorrectly matched tracks than the FAT not using FAME.

The experiment used five different test scenarios to evaluate the improvement FAME adds to the FAT. The first two test scenarios have both a black and a white car in the video. The first scenario has a black car and white car traveling close together and making a left turn. The second scenario has a black car and white car traveling close together, taking a right turn, and then the black car passes the white car. The third test scenario had two white cars traveling in a line and then making a U-turn down the road. The fourth and fifth scenarios test how well the FAT deals with stopped cars. The fourth
scenario has two white cars driving to a stop side by side in the same location. The fifth scenario has two stopped cars and then after some time passes the two stopped cars in scenario five will begin to move apart. All of the test scenarios have a free-moving camera that will zoom in and out on the vehicles. The video stream was read into the FAT at a frame rate of 30 frames per second. CTESS used a pixel distance threshold of 35 pixels when comparing the track points to the truth points. This threshold was arbitrary selected to ensure tracks would be associated with truth objects without incorrectly associating false alarms to a true object. From each of the five scenarios, three different implementations of the FAT using FAME were conducted.

The three implementations used on each scenario were the standalone FAT, FAT using FAME with a global score and gradient value, and FAT using FAME with a scenario tuned score and gradient value. The FAME using global values means that for all the scenarios the same score and gradient threshold were used. For these experiments the global score being used was 0.35 and the global gradient intensity being used was 20. These values were determined by doing multiple testing on all five different scenarios. The goal was to get a low score and a high gradient threshold. The score had to be low enough to match similar fragments together. However, if the score was too high then there is a greater chance incorrect fragments would be matched together. The gradient threshold had to be high enough to filter out any incorrect matched pairs that did not share an edge. The word usage FAME with tuned values means that each test scenario used a different score and gradient threshold. The tuned score and gradient threshold were calculated after conducting multiple tests with different score and gradient value on each scenario. The score and gradient value with the best results were kept for the tuned
scenario experiment. Both the implementations of the global FAME and the tuned FAME used the same normalized features.

For these scenarios, FAME used ASM(2.9.1) and Hu moments(2.6) to calculate the feature score used to match the fragments. The ASM is used to describe a visual feature of each fragment by describing how uniform the pixels are in the fragment. The Hu moments are used to compare the shapes of the fragments in the pair. Hu moments are good at score how similar the shapes of two fragments are since Hu moments are not affected by translation, scaling, and rotation of the fragment [17]. The following tables show the results from the five different test scenarios.
**Scenario 1:** 30 second video of a black car and a white car traveling together then making a left turn. For the tuned FAME in experiment 1 the score was 0.32 and the gradient was 40.

![Figure 15: Frame from scenario 1 video.](image)

<table>
<thead>
<tr>
<th></th>
<th>No FAME</th>
<th>FAME Global</th>
<th>FAME Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>66%</td>
<td>86%</td>
<td>97%</td>
</tr>
<tr>
<td>Purity</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Redundant</td>
<td>27%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>Spuriousness</td>
<td>37%</td>
<td>9%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 2: Results from scenario 1.

![Figure 16: Comparison of FAME implementations used in scenario 1.](image)
Scenario 2: A 51 second video shows a black and white car taking a left turn, with the black car passing the white car. For the tuned FAME in experiment 2, the score was 0.40 and the gradient was 21.

![Frame from scenario 2 video.](image)

Figure 17: Frame from scenario 2 video.

<table>
<thead>
<tr>
<th></th>
<th>No FAME</th>
<th>FAME Global</th>
<th>FAME Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>57%</td>
<td>78%</td>
<td>90%</td>
</tr>
<tr>
<td>Purity</td>
<td>95%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Redundant</td>
<td>18%</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td>Spuriousness</td>
<td>59%</td>
<td>12%</td>
<td>5%</td>
</tr>
</tbody>
</table>

Table 3: Results from scenario 2.

![Comparison of FAME implementations used in scenario 2.](image)

Figure 18: Comparison of FAME implementations used in scenario 2.
**Scenario 3:** 1 minute, two white cars traveling down the road and taking a U-turn. For the tuned FAME in experiment 3 the score was 0.36 and the gradient was 17.

![Frame from scenario 3 video.](image)

Figure 19: Frame from scenario 3 video.

<table>
<thead>
<tr>
<th>metric</th>
<th>No FAME</th>
<th>FAME Global</th>
<th>FAME Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>91%</td>
<td><strong>96%</strong></td>
<td><strong>96%</strong></td>
</tr>
<tr>
<td>Purity</td>
<td><strong>98%</strong></td>
<td>96%</td>
<td>97%</td>
</tr>
<tr>
<td>Redundant</td>
<td>39%</td>
<td>1%</td>
<td>12%</td>
</tr>
<tr>
<td>Spuriousness</td>
<td>19%</td>
<td>4%</td>
<td>5%</td>
</tr>
</tbody>
</table>

Table 4: Results from scenario 3.

![Comparison of FAME implementations used in scenario 3.](image)

Figure 20: Comparison of FAME implementations used in scenario 3.
Scenario 4: 30 second video two white cars coming to a complete stop next to one another. For the tuned FAME in scenario 4 the score was 0.40 and the gradient was 15.

Figure 21: Frame from scenario 4 video.

<table>
<thead>
<tr>
<th></th>
<th>No FAME</th>
<th>FAME Global</th>
<th>FAME Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>33%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Purity</td>
<td>52%</td>
<td>77%</td>
<td>51%</td>
</tr>
<tr>
<td>Redundant</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Spuriousness</td>
<td>21%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 5: Results from scenario 4.

Figure 22: Comparison of FAME implementations used in scenario 4.
**Scenario 5:** 30 second video two parked white cars that start moving after time has elapsed. For the tuned FAME in scenario 5 the score was 0.30 and the gradient was 80.

![Frame from scenario 5 video.](image)

**Figure 23:** Frame from scenario 5 video.

<table>
<thead>
<tr>
<th></th>
<th>No FAME</th>
<th>FAME Global</th>
<th>FAME Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>98%</td>
<td>89%</td>
<td>82%</td>
</tr>
<tr>
<td>Purity</td>
<td>84%</td>
<td>91%</td>
<td>94%</td>
</tr>
<tr>
<td>Redundant</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Spuriousness</td>
<td>5%</td>
<td>2%</td>
<td>5%</td>
</tr>
</tbody>
</table>

**Table 6:** Results from scenario 5.

![Comparison of FAME implementations used in scenario 5.](image)

**Figure 24:** Comparison of FAME implementations used in scenario 5.
4.2 Evaluation

In all five of the test scenarios, the FAT with FAME out preformed the FAT without FAME. The FAT using FAME had a higher completeness and purity than the standalone FAT. Also, the addition of FAME resulted in the lowest redundancy and spuriousness. There are two main reasons why adding FAME improved the performance of the FAT. The first main reason is the median filter applied to the current and previous frame before the frame differencing helped remove some of the added noise from the camera movement. By removing the added noise, there was less track fragments clutter the matching phase of FAME. The second reason was the ability of FAME to match the fragments and form a single valid track.

By matching fragments that share similar features spuriousness and redundancy decreased and the completeness increased. The redundancy and spuriousness decreased due to the two smaller fragments were matched to create a larger fragment that would be passed along to the associator. The larger fragment made the associator’s job easier since there is only one large fragment to match with the previous track. If there were two small fragments, the associator would then have to decide which fragment to associate with the current track and what to do with the second fragment. Even though the FAT using FAME is an improvement, FAME still had a hard time tracking the parked cars in scenario 4 and 5.

Tracking parked cars is one of the hardest objects to track for any tracking algorithm. Since the FAT used in this thesis requires motion to generate tracks, tracking any non-moving object becomes a challenge. Once the cars are parked they appear to be part of the background to the FAT since they were not moving. Since the cars were not
moving their position never changed in the current and previous frame. By having no major changes in the current and previous frame, the frame differencing had a difficult time extracting any meaningful foreground objects. The FAT with FAME was able to track the parked cars in scenario 4 since a track was generated before the cars stopped. Once the cars stopped, the track was able to stay on the parked cars by matching the small detections calculated by the frame differencing due to the cameras movement. However, if the cars were already parked, like in scenario 5, no tracks could be generated until the cars started to move. However, specifically tuning the variables for a certain scenario did slightly help improve FAME's results in some of the scenarios.

Tuning the variables really helped in scenario 5 when FAME had to track cars that started moving. By increasing the gradient threshold in scenario 5, the fragments from the two cars did not join to create one giant car. During testing, FAME had a hard time matching fragments to cars that were really close to one another. Therefore, the gradient check was developed to remove the incorrect fragment match from the close fragments. Also, in scenario 5 the completeness decreased while the purity increased when the FAT used FAME. The reason why the completeness decreased was there were less track points associated to the truth points. However, the tracks that were associated to the truth points were associated to the actual truth point the track belongs to. By having the track points associate to the correct truth points the purity increased. However, tuning the variables did not significantly help for all the experiments.

The reason why specifically tuning the variables to each scenario did not yield significant improvements compared to the global variables was due to the fact that the global variables were very close to the value of the tuned variables. The reason why the
global values were close to the tuned values was the global values were used as benchmark on where to begin tuning the variables for each scenario. If the tuned values were tuned to high, or low, then FAME may not meet the two criterions. So only slightly tuning the variables for each scenario gave acceptable results. Therefore, there was no surprise that the global variables and tuned variables had very similar results. The main reason why tuned variables were compared to global variables was to show the ability of FAME to use the same variables over different data sets and still get better results than the FAT without FAME.
5.0 CONCLUSION

In conclusion, this research showed that image features can be used to match smaller fragments together to create larger fragments. By matching small fragments together, and doing some pre-processing on the frame differenced images, FAME was able to increase completeness and purity while reducing redundancy and spuriousness in the FAT. Even with the moving camera, FAME showed how much of an improvement could be made on a FAT when small fragments are matched together based on a similarity of their features. This research showed that many different concepts from the computer vision, pattern recognition, and tracking field can be combined to correct the multi-fragmentation problem.

In this research, an original filter called FAME was created to fix the multi-fragmentation problem suffered by video tracker. This research showed how FAME can use a combination of features to describe how the similarity of two fragments. Even though FAME is a newly proposed filter, the features used in the fragment feature calculation have been proven by other authors. In addition to the features, the idea of frame differencing is a common technique to extract foreground objects from a video scene. However, the foreground objects were not enough to do all the feature calculations required by FAME. In order to get information about the region between the two fragments to a new mid-region calculation function had to be created. Even with the good results from the FAME algorithm there is still room for future research.

There are two main areas of future research for FAME. Those two main areas of
research are new feature development and adaptive variable calculation. FAME always needs good features to be used to describe how similar two fragments are. In the future, new features will be tested to help improve the performance of FAME. Also, trying to find a way to adaptively choose the user defined variable would be an improvement on the FAME algorithm. By having adaptive variable, the user will no longer be required to set the user defined variables described earlier in the paper. These adaptive variables would make FAME easier to operate for the user.
REFERENCES


23. Khan, Nabeel Younus, Brendan McCane, and Geoff Wyvill. "Sift and surf
performance evaluation against various image deformations on benchmark dataset."
Digital Image Computing Techniques and Applications (DICTA), 2011 International
Conference on. IEEE, 2011.


performance evaluation." International Workshop on Visual Surveillance and