TARGET TRACKING WITH LUCAS-KANADE OPTICAL FLOW AND PARTICLE FILTERS THROUGH AFFINE TRANSFORMS AND OCCLUSION

by

JUSTIN GRAHAM

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2010

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Farhad Kamangar, for his unwavering support and leadership in the background research and mentoring in the writing of this thesis. I am grateful for the involvement of my committee members Dr. Manfred Huber and Dr. David Levine.

I thank George Gazzam from L3 Communications for convincing me to go forward with the thesis option of my Computer Science degree. Thanks to Jason Roberts, Elaine Harmon, Frank Delisle, Ron Cross and Rennie Cross from L3 Communications for making it possible to combine my target tracking research with my work related tasks.

I am grateful to my parents and sisters for their support throughout my life and convincing me to always give my best. Finally, I would like to thank my beloved wife, Dr. Sarabeth Graham for her support throughout my Masters tenure.

November 15, 2010

ABSTRACT

TARGET TRACKING WITH LUCAS-KANADE OPTICAL FLOW
AND PARTICLE FILTERS THROUGH
AFFINE TRANSFORMS
AND OCCLUSION

JUSTIN GRAHAM, M.S.
The University of Texas at Arlington, 2010

Supervising Professor: Farhad Kamangar

This paper investigates a hybrid approach derived from Lucas-Kanade optical flow tracking and particle filters that is capable of tracking objects through occlusion and affine transformations. This approach is inspired by aircraft sensor pod infrared and electro-optical tracking applications. For aircraft based sensors, it is important that a tracking system be able to track through rotations as the aircraft orbits a targeting area. It is also of use to handle cases where the target may be momentarily occluded due to other vehicles or obstacles in the area. The main focus of this investigation is to find a technique that works well in these scenarios for a single tracked target. For tracking performance verification, the implementation of this algorithm is written in Matlab and is not intended to run in realtime, but could be easily extended to do so with minor runtime performance tweaks and native implementations of some of the more performance intensive functions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

There are a wide range of target tracking algorithms [1],[2],[3],[4] that can be applied to a multitude of target tracking issues [5],[9],[6]. The topic of target tracking has evolved tremendously over the years yielding a vast toolset that can be applied to handle a diverse set of tracking scenarios. Each technique has its own advantages and disadvantages necessitating a careful selection process when designing a tracking system. Many of the tools associated with target tracking can be combined in various ways as a means to produce an acceptable tracking solution for a given problem [7]. The focus of this study is to find a tracking solution for tracking both ground and air vehicles from military aircraft. For the purposes of this study, assume that the tracking image resolution is $640x480$.

Given the variety of tools and techniques, it is inherently useful to define the constraints of the generic target tracking problem when designing a target tracking system. Once these constraints are defined the requirements of the proposed tracking system can be examined and a solution designed around those constraints.

The main tracking applications considered for this investigation are intended for use in aircraft based sensor pods. Sensor pods can be used for reconnaissance, target detection, tracking and weapon guidance. The pods also vary with respect to the portion of the electromagnetic spectrum that can be detected. An electro-optical

sensor typically detects visible wavelengths and outputs a greyscale image while a TV-based sensor outputs three bands of colors. A Forward Looking Infrared (FLIR) sensor picks up infrared wavelengths which is useful for tracking targets that contain heat sources such as engines. While night vision based sensors are useful for tracking in low light conditions and pick up more energy in the near infrared spectrum.

Sensor pods can also be attached to moveable mounts so that they can change orientation in realtime. This is especially useful for a pilot since it requires less interaction time from a pilot with a heavy workload. With automated tracking, the pilot can typically click on a target with some form of flight-stick or touchscreen device and the sensor should be able to keep the target in the center of the sensor field of view from that point forward.

1.2 Tracking Classifications

When designing a tracking system the first step is to classify the aspects of target tracking that are being considered for the target tracking application. Some of the potential attributes are enumerated in the following list. Using [1] along with this list would perhaps give some potential algorithms that can be applied to a specific tracking problem.

1. Invariance support
    - Rotational
    - Translational
    - Scale
    - Intensity/Color

2. Tracking system output
    - Target Centroid
    - Bounding Box of Target
    - Elliptical
    - Curve/3-D Representation
3. Common attributes for predicting the physical model of the target
    - Velocity/Acceleration
    - Rotational
    - Translational
    - Scale
    - Intensity/Color
4. Common attributes for predicting the physical model of the sensor
    - Velocity/Acceleration
    - Rotational
    - Translation
    - Scale
    - Intensity/Color
5. Real-time performance needs
6. Support for multiple target tracking
7. The required hardware/software constraints

1.2.1 Invariance Support

One of the first items that should be determined is whether or not the system should track the target through a variety of changes. In other words, the types of invariance should be defined for the specific target tracking application. Inevitably, the tracking requirements will be considered when real-time performance implications are

discovered. For example, assume that an original tracking algorithm handles rotation changes in a tracked target (i.e. rotational invariance). But if the chosen algorithm requires several iterations to handle rotational invariance then this could impact a real-time performance requirement, so this constraint will either need to be lessened or a new algorithm chosen that might impact another constraint.

Tracking targets through changes in translation is one of the most common uses of target tracking. This typically means that the tracking algorithm should support movement of the target over the entire sensor field of view. For example, if the target exists at pixel (x_0, y_0) at frame 0, then at frame 1 the tracking system should be able to find the centroid of the target exists at (x_1, y_1). As with any type of invariance, there are certain degrees to which a tracking system will track a target under translational movement. Here are some factors for determining the translational tracking performance of the algorithm with respect to the tracking scenario.

1. Target speed
2. Sensor speed
    - Accuracy of sensor movement
    - Accuracy of the resultant projected 2-D target movement with respect to the camera movement
3. The limit for the magnitude of translations that the algorithm should handle
4. Output framerate of the sensor

If the system is tracking a slow target from a fixed platform, then we would only need to handle a target translation of one to four pixels so an algorithm based on image registration could prove quite helpful. However, if the system is tracking a jet from a movable sensor platform, then it will probably need an algorithm that can handle drastic translation differences. This can be alleviated by using a sensor

with a faster framerate since the time delta between the detected target positions is smaller. On the other hand, reducing the framerate also reduces the available runtime of the algorithm unless faster hardware or a parallelizable implementation is available. Target translations can be described by matrix 1.1 given by [8]. The variables (v_x, v_y, v_z) represent the target velocity.

$$T_{\mathbf{v}} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

For aerial tracking applications, rotation invariance of targets is a significant issue since the point-of-view of the sensor might change with respect to the orbit of the tracking sensor. For instance, tracking a building from a moving car can be done without a rotational invariant algorithm but an aerial case would benefit much more from this property. Even if the physical position of the sensor does not lend itself to a rotational invariant algorithm many sensors will rotate based on physical constraints when the target and tracking vehicle both follow a straight path. The most common case for rotational invariance occurs when the sensor reaches gimbal lock and needs to make a significant rotation to handle a track. The matrix in equation 1.2 below describes a 2-D rotation about the x-axis [8]. Figures 1.1-1.4 show an example of resultant rotation on a stationary vehicle when the aircraft is orbiting a target.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{1.2}$$

Figure 1.1. Rotation frame η_1.

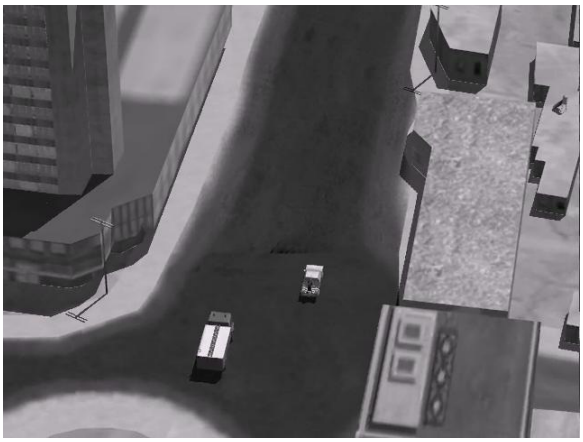

Figure 1.2. Rotation frame η_2.

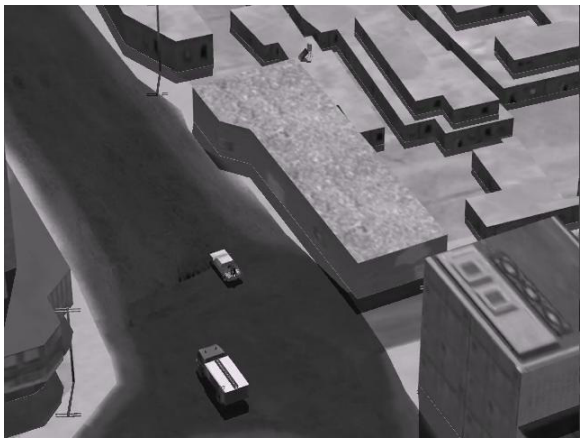

Figure 1.3. Rotation frame η_3.

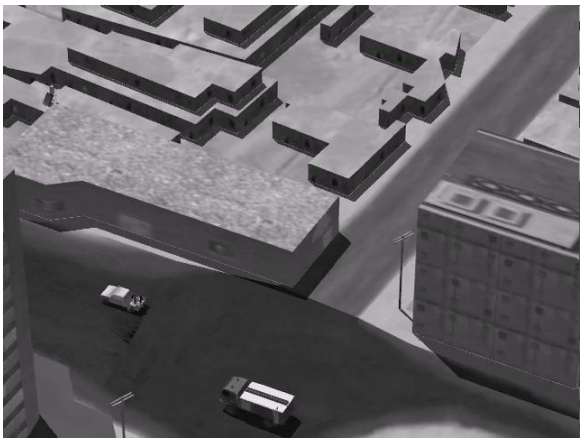

Figure 1.4. Rotation frame η_4.

Typically, the most drastic changes in scale occur when the sensor is capable of being zoomed. In that case, it is easier to determine the scale changes since the tracking system would be aware of the scale factor delta of the sensor. Scale changes can also occur when the distance between the target and sensor changes. This case is more difficult to detect since it is not a direct input to the sensor. This particular problem is solved by the algorithm since many scale variant algorithms can be modified in order to support scale invariance by performing two additional iterations of the algorithm based on different scale factors of the target [9]. The

highest scoring iteration will give the tracking system information on how the scale of the target has changed. The scale changes can be described by equation 1.3 [8].

$$S_v = \begin{bmatrix} v_x & 0 & 0 & 0 \\ 0 & v_y & 0 & 0 \\ 0 & 0 & v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

The next type of invariance that should be addressed is intensity invariance. Intensity in this context is used to refer the pixel intensity of a single color channel. For FLIR sensors, the targets are usually hot engines which will typically result in higher intensity pixels [10]. This intensity can change depending on the state of the engine and whether or not the algorithm is applied after gain and level operations. The background can have a tremendous impact on the target intensity if it contains brighter elements such as the sun which can saturate the sensor. Color sensors are not immune to this change either since shadows and lighting are significant factors for the resultant color of the target.

1.2.2 Tracking System Output

The next issue that needs to be addressed is that of presenting the user with enough information regarding the bounds and location of the target. The easiest case would be for the algorithm to return the point at which the target exists or the centroid of the target. Any of the algorithms described in this paper can return this information. Another relatively simple option is to return a bounding box. Note that an algorithm that is not scale invariant can still return a bounding box but it might

not change size as required. For face tracking [11], it can be helpful to use an ellipse since a higher percentage of the tracked area can be used to track the foreground. For tracking man made objects, a rectangle might work well depending on the target type [12]. All of the above types of tracking output are relatively easy to determine assuming that the tracking solution is correct, but the algorithm gets more complex when tracking contours due to performance limits. Contour based tracking results in a highly detailed bounding region that surround the exact boundary of the target. The condensation [4] algorithm attempts to address this by using a reduced state space based on the results of a learning algorithm. This reduction enables them to use a reasonable number of parameters to describe a B-spline that can change while maintaining real time performance. This approach works well when the state space is reduced significantly but poses performance complications when tracking a generic type of object due to the computational complexity.

1.2.3 Physical Model of Target

Assuming that the sensor is stationary, another tracking attribute that needs attention is to determine if the target has a physical model that can be leveraged to predict positions and states in subsequent frames. In one extreme, the physics model of a bouncing ball can be predicted to such accuracy that a visual-based target tracking system is unnecessary. For the opposite extreme, a physical model might be impractical for tracking the exact position of a hovering hummingbird if the framerate of our sensor is too low and the required tracking accuracy is high. The latter case would work totally on visual based techniques while the former case could merely be described with a physics model. Many cases exist in between these two extremes that could use both a visual and physical model in order to perform target tracking. One of these cases could involve tracking a car from an elevated sensor. While there are

some basic laws of physics that can be modeled for cars, the position of the car is also controlled by the human driver that cannot be reliably modeled. Physics will keep a typical truck from reaching Mach 1 and reaching across a reasonable field of view in one frame. This implies that a search area can be deduced from the physical model in order to limit the visual computations. At the same time, the human driver may decide to accelerate or turn at any given moment so a visual model is also needed for a complete tracking solution. When determining a physical model, the following issues must be addressed.

1. The velocity limits of the target with respect to the sensor in pixels per frame
2. The acceleration limits of the target with respect to the sensor
3. Target rotation with respect to the sensor
    - Determine if the tracking solution should track through rotations
4. Target size adjustments with respect to the sensor
5. Target intensity and color changes with respect to the sensor

Notice that the above questions are all in respect to the tracking sensor. Determining the actual 3D real-world position requires more information and/or multiple sensors as discussed in [13].

1.2.4 Physical Model of Sensor

Another set of important factors to consider are the attributes of the sensor itself. Visual, FLIR, and NVG sensors all have various advantages and disadvantages depending on the applications. There also exist subcategories for every sensor type which might pick up different spectral bands better than others. Another factor that needs to be determined is whether or not the sensor will move, rotate, or have zooming capabilities. Fortunately, changes in the sensor can be fed directly into the tracking

system since accelerometers and GPS tracking devices have become more affordable. These can include changes in heading, pitch, roll, zoom, and absolute position.

1.2.5 Performance and Multiple Target Tracking

If tracking can be performed on an offline basis, then the algorithmic options are numerous since time will not be a significant factor. However, if real-time performance is needed processing time will be limited and the system most likely needs a good physical model to reduce the target state search space. The implementation also depends on the software and hardware constraints. If a set of parallel CPUs or GPUs are used, then an algorithm that works on pixels independently of one another will be preferred over an algorithm that needs all of the information in the current frame at a single point in time [14].

1.3 Application Scenario

The focus of the described approach is to develop a tracking system that handles tracking a target through an occlusion event over a short period of time. It is desirable to maintain track of a target even though visibility might be lost for a period of time in certain tracking applications. This attribute is especially desirable when designing an air to ground tracking system where ground vehicles can be obstructed from line of sight by a variety of obstacles such as buildings, tunnels, and other vehicles. This is also beneficial for air to air tracking as cases can arise where the line of sight to a tracked aircraft can be blocked by terrain, clouds, and other aircraft.

Following the aforementioned tracking attributes list, table 1.1, lists the importance of each attribute. The invariance attributes for the tracking scenario are quite difficult as two out of three attributes have a high importance. Scaling is weighted

Table 1.1. Tracking Invariance Attributes of Aircraft Sensor Targets

Attribute	**Importance**
Rotational	High
Translational	High
Scale	Medium

Table 1.2. Tracking Output Priorities of Aircraft Sensor Targets

Attribute	**Importance**
Point	High
Bounding Box	Medium
Elliptical	None
Curve	None
3-D Rep.	None

lower in this case since it is used in fewer instances than translational and rotational invariance. The attributes in table 1.2 show that a bounding box surrounding the target is all that will be needed as the output of the system. For sensor tracking, the single most important tracking detail attribute is the accuracy of the tracking centroid since it is used in order to command sensor position alterations. The bounding box is secondary and many current sensor tracking implementations use a fixed bounding box [10].

Table 1.3. Physical Model Prediction of Aircraft Sensor Targets

Attribute	**Predictability**
Velocity	Medium
Acceleration	Medium
Rotational	Medium
Scale	None
Intensity	High

Table 1.4. Intrinsic Physical Model Prediction of Aircraft Sensors

Attribute	**Predictability**
Velocity	High
Acceleration	High
Rotational	High
Scale	High
Intensity	High

The physical attributes 1.3, provide the tracking algorithm with several possible trackable attributes. The velocity and acceleration attributes are predictable to a certain degree since the targets must follow the laws of physics, but the human controllers can alter the direction of motion. For FLIR sensors, intensity is perhaps the most predictable attribute as it will remain within a small range in most cases. This is especially true when tracking with an air target since engine exhaust typically saturates the scene. The sensor state attributes for this application are highly predictable due to the accelerometers, gyroscopes, and other sensor systems currently available to an aircraft. Table 1.4 shows this is the case since the aircraft is aware of the zoom level, position, and orientation of the pod at any given time. It is also important that the tracker work in real-time since the algorithm is driving a hardware component that will position the sensor. In many cases, sensor pods can track many targets, but for the purposes of this application a single target track will be assumed since multiple target tracking can be treated as a completely different subject.

Table 1.5. Additional Attributes of Aircraft Sensors

Attribute	**Value**
Run-time Performance	Real-time
Multiple Targets	Yes

This tracking application presents many challenges since the targets can undergo a variety of affine transforms. At the same time, there are many attributes that can be used to maintain track while undergoing affine transformations.

CHAPTER 2

BACKGROUND

2.1 Tracking Model

All of the aforementioned target tracking system attributes have been previously studied in some form or fashion [15],[16],[17],[18],[19],[1]. The next step is to collect the more promising methods and determine how they fit into the target tracking attributes that have been defined.

2.1.1 Kinematic Model

Given a case where the tracking system can avoid searching the entire field of view for the target, it would be helpful to characterize several techniques that would aide in narrowing the search area. Perhaps one of the most simple approaches is to model the velocity of the target given an initial starting velocity $v_0 = (vx_0, vy_0)$, initial starting position $p_0 = (px_0, py_0)$, and determine the estimated position p_1^- of the target in the next frame. Equation 2.1 shows a simple equation that could be used to project the position into the next frame.

$$p_1^- = p_0 + v_0 t \tag{2.1}$$

This approach simply increases the position by the given velocity for every frame. A simple technique to determine the search box size (b) is to use the maximum velocity of the target v_{max}.

$$b = 2v_{max} \tag{2.2}$$

Once this is determined, a search box and corresponding image pixels can be sent to the vision algorithm (V). This algorithm will then return a refined position (p_1) based on the image pixels (I) in the search box as follows

$$p_1 = V(I, p_1^-, b) \tag{2.3}$$

For this simplistic example, both velocity and the bounding box size are assumed to be constant. When used as shown, the sole purpose of the vision algorithm is to correct the estimates from the physics based model. When this notation is converted to a more generic form with respect to time step (k), it would produce the following set of equations based on the notation from Kalman Filtering [20].

$$p_k^- = p_{k-1} + vt \tag{2.4}$$

$$b = 2v_{max} \tag{2.5}$$

$$p_k = V(I, p_k^-, b) \tag{2.6}$$

For this simple example, the only parameters that vary from frame to frame is the position or centroid of the target since this is a constant velocity model. A further improvement could be made to the model by adding variable velocity capability with

$$v_k^- = v_{k-1} + at \tag{2.7}$$

$$p_k^- = p_{k-1} + v_k^- t \tag{2.8}$$

$$\mathrm{b} = 2\mathrm{v}_{max} \tag{2.9}$$

$$p_k = V(I, p_k^-, b) \tag{2.10}$$

where a is the acceleration constant. The tracking model now includes a velocity vector that will change from frame to frame. For notational convenience, these target state variables are grouped into a state vector according to the Kalman [20] notation.

$$x_k = (p_k, v_k) \tag{2.11}$$

where x is the state of our target at time slice k. Since both p and v are vectors we can further expand this equation:

$$x_k = (px_k, py_k, vx_k, vy_k) \tag{2.12}$$

Adding a variable for acceleration would merely result in the addition of another two variables to the state. However, at some point the law of diminishing returns comes into play and the computation time required for all of the variables will outweigh any small tracking benefit that might be found.

Up to this point the assumption has been that the vision algorithm (V) will refine the original estimation of the target's position. In practice, V will not always find the exact position of the target depending on scene contrast, noise, and background clutter. The motion model can help out in this case as well if a good motion model can be found for the tracking application. Many vision algorithms will weight positions closer to the estimated position higher which is one way to get around this issue. Another popular way that can be used in conjunction with this technique is Kalman filtering. This type of temporal filtering was originally developed for 1D signal processing. Kalman filtering is described in [21], but an overview of the main concepts needs to be described for reference purposes.

As in the simple motion model, there are two major steps used in a Kalman filter target tracking implementation:

1. Predict the state of the target
2. Use this prediction and the results from V to update the predicted position

The first step used the following two predictions:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{2.13}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.14}$$

where

$\hat{x}_k^-$: Prior or predicted state of the target at time slice k

$\hat{x}_k$: Posterior or estimated state of the target at time slice k

A: The motion model matrix

u_k : Control inputs used to rotate, move, and zoom the sensor

B : Maps control input u_k to the target state

P_k : Posterior error covariance matrix

Q : Process noise covariance matrix

In the second step, V is used along with the prior state estimate to produce the posterior state estimate:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.15}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.16}$$

$$P_k = (I - K_k H)P_k^- \tag{2.17}$$

where

K_k : The Kalman gain at time slice k

z_k : The result of the vision algorithm V

R : The measurement error covariance

H : Maps the output of V to the target state

I : The identity matrix

Other than a more specific notation in converting the results of measurements and sensor inputs to target state, the only additional items that need to be found from a simple motion model are the error covariance matrices. In practice, these matrices will need to be generated from an extensive set of test cases. When generating the test cases the following items need to be addressed.

1. The accuracy of V
2. The accuracy of the motion model

From the test cases, the average error can be determined for each of these parameters. However, this poses a risk since the test cases may not match an actual scenario so the covariances might differ by a significant amount. Even in the test cases, the covariance might differ significantly due to the dynamic implications of a given target tracking application. In fact, if this difference becomes dramatic it might be more accurate to zero out R and Q.

Kalman filtering assumes that one is dealing with a linear system, but in many target tracking cases this will not be true. For cases where a human is in the loop driving a target, one cannot reliably predict which move the human will make. The Extended Kalman Filter (EKF) is an attempt to modify the original filter to work for a non-linear system by using

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_k, 0) \tag{2.18}$$

and

$$\tilde{z}_k = h(\tilde{x}_k, 0) \tag{2.19}$$

where f is a non-linear function [22]. For example, this would be helpful in a case where a human driver is being tracked on a highway and the position where the human will turn is known. However, this data is often not available. Another option is to use an Unscented Kalman Filter (UKF) [23], which approximates the probability distribution using a deterministic sample set which is easier to approximate than an arbitrary nonlinear function. Like the other Kalman filters, this has two major steps as well. The prediction step will first start by creating multiple samples also known as sigma points.

$$x^0_{k-1} = x^a_{k-1} \tag{2.20}$$

$$-1 < W^0 < 1 \tag{2.21}$$

$$x^i_{k-1} = x^a_{k-1} + \left(\sqrt{\frac{n}{1-W^0}P_{k-1}}\right)_i, forall\ i = 1...n \tag{2.22}$$

$$x^{i+n}_{k-1} = x^a_{k-1} - \left(\sqrt{\frac{n}{1-W^0}P_{k-1}}\right)_i, forall\ i = 1...n \tag{2.23}$$

$$W^j = \frac{1-W^0}{2n} forall\ j = 1...2n \tag{2.24}$$

where the subscript i of the square root represents the ith column or row of the matrix and n is the dimension of the state space. The next part of the prediction step propagates the sigma points through the process model.

$$x_k^{f,j} = f(x_{k-1}^j) \tag{2.25}$$

$$x_k^f = \sum_{j=0}^{2n} W^j x_k^{f,j} \tag{2.26}$$

$$P_k^f = \sum_{j=0}^{2n} W^j \left(x_k^{f,j} - x_k^f\right)\left(x_k^{f,j} - x_k^f\right)^T + Q_{k-1} \tag{2.27}$$

Then the system propagates the points through the measurement model

$$z_{k-1}^f = \sum_{j=0}^{2n} W^j z_{k-1}^{f,j} \tag{2.28}$$

$$Cov(\tilde{z}_{k-1}^f) = \sum_{j=0}^{2n} W^j \left(z_{k-1}^{f,j} - z_{k-1}^f\right)\left(z_{k-1}^{f,j} - z_{k-1}^f\right)^T + R_k \tag{2.29}$$

$$Cov(\tilde{x}_k^f, \tilde{z}_{k-1}^f) = \sum_{j=0}^{2n} W^j \left(x_k^{f,j} - x_k^f\right)\left(z_{k-1}^{f,j} - z_{k-1}^f\right)^T + R_k \tag{2.30}$$

The next step is to use the data collected from the prediction step and the current measurement data to arrive at the estimated state.

$$x_k^a = x_k^f + K_k(z_k - z_{k-1}^f) \tag{2.31}$$

$$K_k = Cov(\tilde{x}_k^f, \tilde{z}_{k-1}^f) Cov^{-1}(\tilde{z}_{k-1}^f) \tag{2.32}$$

$$P_k = P_k^f - K_k Cov(\tilde{z}_{k-1}^f) K_k^T \tag{2.33}$$

where x_k^a is our estimated state which could also be represented as $\hat{x}_k$. This demonstrates a feasible way of representing a non-linear system with a Kalman filter. This technique will be able to handle human guided targets as they change course to a slightly greater degree, but it still does not handle a truly nonlinear function.

2.1.2 Particle Filters

In extending the kinematic model, it would be useful to investigate particle filters. Particle filters consist of a set of particles that are distributed randomly within the image bounds. Each particle moves according to a motion model which could be as simple as just using the speed from the previous two frames or using something more complex as calculating relative bearings of targets and using the real-world specs of the target (i.e. F16, F18, etc) and transferring that image to a 2D space where the probabilities of the target position are known precisely. Each particle also has a weight which is determined by scoring the current position of a particle. For example, if golf ball where being tracked the weight would be determined by how close the particle is to the center of the ball. This can be determined by looking for an edge with a constant radius around the particle. Particles with a higher weight will typically survive, while particles with a lower weight will eventually be filtered out and new ones regenerated based on the particles with higher weights. These are the basic steps of a particle system which will be described in more detail:

1. Initialize the particle positions and states
2. Use a motion model to move the particles around
3. Calculate weights/importance of the particle by analyzing the surrounding pixels
4. Resample and weed out the particles with low weights

There are many different options to take when considering the initialization of the particle system. One approach is to randomly distribute the particles in the image and utilize these particles to find new targets. Another option is to distribute the particles in places where targets are known to exist. Target positions can either be determined from user input or from an additional target recognition algorithm that

feeds input into the target tracker.

For now, it is assumed that the initial target positions are known so particles will initially be near the target areas. The next attribute to determine is the number of particles to use for each target. The most straightforward approach would just to use P/T particles where P is the number of particles in the system and T is the number of targets. However, if targets are weighted by importance then more particles should be allocated to the targets with higher importance.

The next step is to move the particles based on a motion model with a probability distribution. The main disadvantage with this approach is that the particle behavior is not deterministic so it will be more difficult to create automated tests and to duplicate bugs. A straightforward approach that leads to reasonable results is to just use the previous two positions to calculate a velocity for each particle. So the Matlab code would look something like this:

```
particle_velocity ( i ) =
  new_position ( i ) - old_position ( i )
new_position ( i ) =
  new_position ( i ) + rand() * velocity_sigma
                              + particle_velocity ( i )
```

Notice that a call is made to a random number generator so that the particle will visit different parts of the image. Since targets have inertia and can only move to certain areas of the image this model works fairly well and will cause the particles to move in a fashion that is close to physical accuracy. On the other hand, it cannot predicted with complete certainty where a target will be in the next frame, so that is

why a random element is inserted.

Since each particle moves around in a random fashion it is important to score each particle based on the surrounding pixels. This score could be based on edge information, thresholding, or some combination of the two. If a white-hot target is being tracked, then the system would score particles that are surrounded by high-intensity pixels higher than those that are not. However, this scheme is a bit simplistic and would probably need to be combined with edge information for more effective tracking. For each frame the system would calculate the product of the new weight along with the current weight of the particle so that particles that have a history of finding targets will get higher scores rather than those that just happen upon high-intensity areas for a frame. The weights are then normalized each frame to keep them from approaching zero and to keep the particles on the same weighting scale.

There are many ways to resample the particles with lower weights in order to keep an effective particle system. The main point is that the system needs to weed out the bad particles and keep the good ones. The following list shows some possible ways to determine the correct time to resample.

1. Only resample when the weight variance reaches a certain threshold. A low variance is dangerous because it would imply that all of the particles are in the same place while a high variance would imply that the particles are too scattered and the system might eventually lose the track.
2. Eliminate the lowest X% of particles every frame

Here are some techniques to determine how to resample.

1. Replace all particles by randomly picking from the previous particle distribution. This gives particles with a higher weight a higher likelihood of getting picked.

2. Replace only a certain percentage of the lowest scoring particles.
    - Pick the particles from the probability distribution of the previous particles
    - Place the new particles in a completely random location

So, now that the system contains a list of particles the next major issue to discuss is where to declare the position of the target or where to draw the targeting symbology. If the system is performing single-target tracking, it can choose the highest weighted target, but this might cause it to lose track on background clutter. A safer way would be to just take a weighted average of the particles and use that, however that would not work well for multiple target tracking since it would just find the point in between two targets which would be useless. A more robust way would be to find particle groups and take the weighted average of each group. Alternatively, if a particle has a high score but is not surrounded by other particles the system could just weed that particle out and this would reduce the background clutter problems but still allow us to track multiple targets.

Particle filters are a popular way of performing target tracking but they do have their drawbacks.

1. Pros
    - Can be used for effective multiple target tracking
    - Can be extended for target acquisition
    - Can be as simple or complex as we want since there are many options for the motion, weight, and resampling models
2. Cons
    - Are not deterministic, which makes testing difficult, yet not impossible

- Creating a weight calculation model that works for any trackable object at any angle can be difficult. This would probably involve using intensity thresholding to establish a track and using a shape model once that track has been established.
- Can be resource intensive depending on the amount of particles and weight calculation model
- Finding multiple target points can be resource intensive since it would involve finding particle groups which could be an $O(n^2)$ operation.

Note that the formal algorithm for particle filters is given by [16] for sequential importance resampling, assume $L = 1, \ldots, P$:

1. Pick samples from the proposal distribution based on observations and previous samples

$$x_k^{(L)} \sim \pi(x_k | x_{0:k-1}^{(L)}, y_{0:k}) \tag{2.34}$$

2. Calculate new weights by factoring in the old weights and the new observations

$$\hat{w}_k^{(L)} = w_{k-1}^{(L)} p(y_k | x_k^{(L)}), \tag{2.35}$$

3. Normalize the weights between 0 and 1

$$w_k^{(L)} = \frac{\hat{w}_k^{(L)}}{\sum_{J=1}^{P} \hat{w}_k^{(J)}} \tag{2.36}$$

4. Determine the number of useful particles

$$\hat{N}_{\mathit{eff}} = \frac{1}{\sum_{L=1}^{P} \left(w_k^{(L)} \right)^2} \tag{2.37}$$

5. If the effective number of particles is less than a threshold $\hat{N}_{\mathit{eff}} < N_{thr}$, then perform resampling

- Pick P particles from the current set of particles with probabilities proportional to their normalized weights and reset the weights.
- $w_k^{(L)} = 1/P$.

This is similar to the techniques used for the final implemented particle filter, except for the fact that the resampling phase keeps the top X percent of the particles to assure that the good particles survive.

2.2 Vision Algorithms

Up to this point, many kinematic tracking algorithms have been defined, but now it will be beneficial to look at some vision algorithms that will help with target tracking. In other words, the focus will be shifted on defining the aforementioned V. Addressing every algorithm that has been considered in the field of target tracking is beyond scope, but what follows will be a brief description of the more popular algorithms.

2.2.1 Threshold Tracking

Perhaps one of the easiest algorithms to implement with real-time performance is threshold tracking. Threshold tracking consists of picking a pixel intensity range $T_{range} = [T_{min}, T_{max}]$. Sometimes it can be as simple as picking a T_{min} in the case of tracking a hot engine in the case of a FLIR sensor. If, for example, a user can specify target pixel intensity characteristics that will remain within a small range during tracking then the threshold can remain constant. In this case the algorithm steps would be:

1. Define T_{range}
2. Threshold the image
   - $I_{thresh} = (I > T_{min})\,\&\&\,(I < T_{max})$

Figure 2.1. Sample model aircraft.

3. Find the connected components
   - $C = ConnectedComponents(I_{thresh})$
4. Filter the connected components and find the best match
   - $M_{best} = Filter(C)$

$Filter()$ would most likely use some form of temporal or kinematic tracking to determine the best match. This technique works quite well if the background pixels do not overlap the range of the foreground pixels. The pixel intensity ranges also cannot change dramatically. However, if the pixel intensities do change, then the system can fight this with some variable threshold techniques as shown below.

One of the simplest variable threshold techniques is just to base T_{range} on statistics collected from the image pixels using a simple equation such as

$$T_{thresh} = [mean(I) - \alpha_{min} stdev(I), mean(I) + \alpha_{max} stdev(I)] \tag{2.38}$$

where α represents an arbitrary constant. The constants are typically chosen through a series of test cases. However, what the system really needs to find is the pixel intensity that separates the foreground and background pixels. For instance, take the histogram of the model aircraft in 2.1 and the corresponding histogram in 2.2.

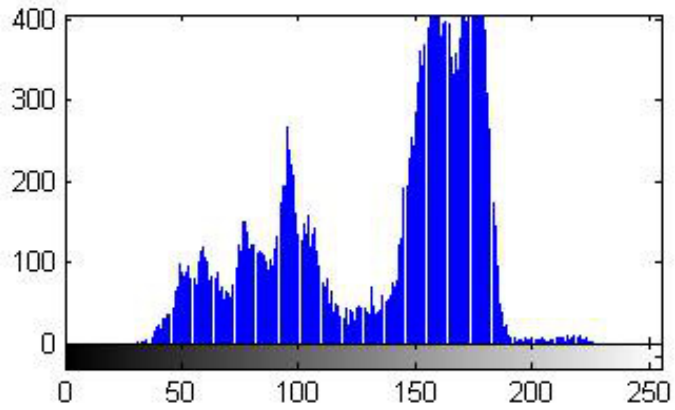

Figure 2.2. Sample model aircraft histogram.

One can see a significant valley occurring around pixel intensity 125. It turns out that this is a good threshold if the system is tracking the darker areas of the aircraft. But there are also other valleys around 70, 80, and 170 so it can be difficult to determine the threshold. There are a couple of algorithms that can be used to address this issue.

One technique that can be used to find a significant valley is to take the derivative of the histogram and find the midpoint of the two highest points on the derivative. However, this will not work in all cases due to the nature of real-world histograms. This technique is easily defeated by histogram noise that emanates from background clutter. Another technique to use is based on K-means thresholding [24] or Otsu thresholding [24].

2.2.2 Template Tracking

A tracking technique that can be used is based on an image sample of the target T (i.e. the template) and determining the best fit of the template within the next frame. The basic technique works only if the target does not change dramatically

from frame to frame, but there are some modifications that can be made in order to handle small changes between frames. Template based techniques tend to require more hardware resources and/or frametime.

One technique that can be used to compare T to the next frame, I_k is to move the template over I_k and subtract each pixel of T from I_k and then take the absolute value of the difference and sum the differences. The best match will have the lowest score. This technique is easy to implement but issues start occurring when the image intensity changes from frame to frame.

Cross correlation is rooted in 1D signal processing where it is used to find the time domain offset between two similar signals. In image processing, it can be used in 2D where the goal is to find the X and Y offsets instead of the time offset. This technique tends to fail for brighter areas in the image since the multiplication operations will tend to overweight those areas. Equation 2.39 performs the cross correlation between two functions f and g. This operation needs to be performed for every element of interest and can be an expensive operation if the image size is too large [42].

$$(f \star g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\ g[n+m]. \tag{2.39}$$

Normalized cross correlation is an improvement upon standard cross correlation since it attempts to account for bright spots in the image and is also more resistant to intensity changes from frame to frame. Equation 2.40 shows this operation and how standard deviation and mean are used in order to combat false matches [42].

$$\frac{1}{n-1}\sum_{x,y}\frac{(f(x,y)-\overline{f})(t(x,y)-\bar{t})}{\sigma_f\sigma_t} \tag{2.40}$$

One major fallback of spatial domain correlation is that it is extremely resource intensive. One way around this is to perform Discrete Fourier Transforms (DFT) on the template and image and then perform a multiplication on the output from the DFT. The best match will be the highest scoring pixel in the result image R.

$$R = DFT(Rotate180(T)) \cdot DFT(I) \tag{2.41}$$

2.2.3 Histogram Based Tracking

The histogram based techniques mentioned here use the histogram of a template $H_{template}$ in order to find the match in subsequent frames. Mean shift tracking is an algorithm based on a feature histogram of the target and the localization of the target can be found by determining a similarity measure between the template and a set of points on the tracking window [9]. The paper by Comaniciu and Ramesh claims that this algorithm will cope with the following typical target tracking issues:

1. Camera Motion
2. Partial Occlusions
3. Clutter
4. Target Scale variations

A mean-shift based approach can separate the components of a target tracker into the following categories

1. Target Representation and Localization
    - Copes with target appearance changes
2. Filtering and Data Association
    - Top down process which deals with the dynamics of the tracked object

The reference target model or template is represented by a propagation density function (q) in feature space. This representation can simply be a color histogram of the target. Using an isotropic kernel in the spatial domain we can produce a continuous similarity function for which it will be easer to find a similarity measure.

The target model is represented within an elliptical region where the row and column dimensions are first normalized resulting in a circular region to start. The system then assigns smaller weights to pixels further from the center since outlying pixels tend to be occluded more often. It then defines distance as an inverse of what one normally defines as distance:

$$d(y) = \sqrt{1 - p[\hat{p}(y), \hat{q}]} \tag{2.42}$$

$$\hat{p}(y) = p[\hat{p}(y), \hat{q}] = \sum_{u=1}^{m} \sqrt{\hat{p_u}(y)\hat{q}_u} \tag{2.43}$$

This may seem like a strange way to define distance since it just emphasizes larger values and then inverses them. However, [9] claims that it will provide a similarity measure that is computationally feasible to optimize. Once a taylor series expansion is created, the distance can be formulated in terms of the previous location and the current locations. This equation can be minimized using the mean shift procedure. This procedure is described in the next itemized list. The Bhattacharyya coefficient mentioned in the list is a measure of similarity between two functions [9]. This equation used to for this coefficient will be described in the next chapter.

1. Initialize the location of the target and compute the Bhattacharyya coefficient.
2. Calculate the weights of each feature (i.e. bin in the histogram)
3. Compute the location of the next target candidate

4. Compute the next Bhattacharyya coefficient
5. While the new coefficient is less than the previous
    - Set the new position halfway between the current and previous
    - Calculate the new coefficient
6. If the new position is less than a minimum value (epsilon), then stop
    - Otherwise set $y_0 = y_1$ and go to step 2

The minimum value, also referred to as epsilon, is typically limited to pixel resolution. The number of iterations tends to be equal to 4 in practice, but is limited to 20 in this paper. Removing step 5 can be done to improve performance but might result in some errant maximizations in 0.1% of the cases according to this paper. If this is done, the B coefficient can be computed once per frame in order to provide a similarity coefficient. This claims that the algorithm finds the local maximum match within the current neighborhood.

In order to handle scale changes in the target, this paper [9] proposes running this mean shift (MS) algorithm 3 times with a smaller, same, and larger scale size respectively. Then the best scoring run is chosen and alpha blended with the previous kernel to prevent over-sensitive adaptation.

The computation complexity of this algorithm is reported to be $N \cdot n_h \cdot c_s$ where n_h is the cost of the histogram and c_s is the cost of an addition, square root and division and N is the average number of iterations per frame. That's quite impressive considering that the average number of iterations for the application in [9] is four. On a 1GHz PC with scale adaptations, this can run at 150fps while tracking 5 targets [9].

2.2.4 Contour Based Tracking

In addition to intensity and histogram information targets can also be tracked by the contour of the target. The condensation algorithm [4] uses this approach in addition to a particle filter, active contour frameworks, and Bayesian nonlinear filtering techniques. It also switches between multiple probabilistic models in an attempt to improve target tracking results.

Condensation tracking as implemented by Isard uses a B-spline framework for curve tracking. Using a B-spline curve to track an object is computationally expensive since both position and changes to the contour must be accounted for from frame to frame. Condensation tracking works around this by defining a shape space that drastically reduces the number of B-spline states such that it is a tractable real-time problem. However, this approach requires extensive training data in order to reduce the shape space. This would be an ideal tracking solution for tracking sign language symbols since the training set it well known but this becomes more of an issue for the dynamic targets that can be tracked with a targeting pod. Since the pilot can choose to track anything from aircraft and ground vehicles to patches of land and crowds the shape space would be quite extensive. However, one aspect that was used in Isard's implementation of condensation tracking is the use of a particle filter which can be adapted to a sensor pod tracking application at minimal expense. Particle filters in and of themselves are a generic solution that can be used in a wide range of applications.

2.2.5 Optical Flow Techniques

Optic flow based techniques tend to have their roots based in image registration where the frame to frame translations and rotations are found for every frame. One

technique that was researched under this branch of target tracking was the Lucas Kanade feature tracker.

An implementation of the Lucas Kanade feature tracker is described in [25]. The definitions for the Lucas Kanade tracker are given below:

- $I(x) = I(x, y)$: The first image or the previous frame
- $J(x) = J(x, y)$: The second image or the current frame
- $u = [u_x u_y]$: The centroid of a feature point in I
- $v = u + d = [u_x + d_x, u_y + d_y]$: The same feature point in J offset by d
- w_x: Distance between centroid of feature to edge of window in the x dimension
- w_y: Distance between centroid of feature to edge of window in the y dimension
- $\epsilon(d) = \epsilon(d_x, d_y) = \sum_{y=u_y-w_y}^{u_y+w_y} \sum_{x=u_x-w_x}^{u_x+w_x} (I(x, y) - J(x + d_x, y + d_y))^2$: The sum to minimize

The use of a small window will prevent the smoothing out of smaller features, but large windows are necessary to pick up fast motion sequences so unless other matching techniques are used:

$$d_x \leq w_x \tag{2.44}$$

$$d_y \leq w_y \tag{2.45}$$

The paper [25] then goes on to describe the image as a pyramid of resolutions where level zero represents the highest resolution and the image is basically mipmapped for performance purposes. It then describes some notational convenience rules by defining that the last pixel can be referenced by the width and height coordinates without a -1. Pixels can be computed for different mip levels using the following equations:

$$u^L = u/2^L \tag{2.46}$$

The algorithm proceeds by finding the optical flow at the lowest resolution and uses the flow (g) at level L as a starting point ($g^L = [g_x^L g_y^L]^T$) to find the flow at L-1. The resulting error equation is then

$$\epsilon^L(d^L) = \sum_{x=u_x^L-w_x}^{u_x^L+w_x} \sum_{y=u_y^L-w_y}^{u_y^L+w_y} (I^L(x,y) - J^L(x+g_x^L+d_x^L, y+g_y^L+d_y^L))^2. \tag{2.47}$$

This will make the Lucas Kanade algorithm costs small since we will only move by a small amount for each resolution level. The g value is propagated as follows:

$$g^{L-1} = 2(g^L + d^L) \tag{2.48}$$

The paper [25] then describes the core Lucas-Kanade optical flow computation, with the images being represented as A and B instead of I and J. It also redefines the other parameters that were used before to avoid confusion, but the same basic equations are used with the exception that the first image A uses a range between $(2w_x + 3) \cdot (2w_y + 3)$.

The Lucas Kanade algorithm then defines the image derivatives and how the Sharr or central difference operators can be used to compute them. It mentions that the Lucas Kanade optical flow equation is only valid for small pixel displacements which fits well into the hierarchical scheme. Once the equations are defined, the algorithm uses the equation in an iterative fashion in order to find the optimal optical flow values. For every iteration the system computes

$$\bar{\eta}^k = G^{-1}\bar{b}_k \tag{2.49}$$

where G is the gradient matrix and $\bar{b}_k$ is a vector based on the gradient of the image difference. This value is then fed into the next iteration to refine the offset. The value typically converges within about 5 iterations [25].

The base Lucas Kanade algorithm can also be applied on separate mipmap levels in order to handle larger movements. It mentions that allowing for subpixel accuracy is important and that bilinear interpolation should be used for computing intermediate brightness values. Tracking around the border is also important since the algorithm is using a mipmap scheme which increases the dead zone size. However, the Lucas Kanade equations are still valid as long as the system integrates only over the valid area of the window. One challenge with Lucas Kanade tracking in general is to determine when a track is lost. The system can threshold based on the epsilon value, but this can vary significantly from frame to frame while keeping a good track. It is instead recommended to use a second level detailed tracker that is able to track affine transformations [2].

One important facet of this algorithm is selecting a reasonable feature to track, [25] describes the following process:

1. Compute the G matrix and its minimum eigenvalue λ_m
2. Call λ_{max} the maximum eigenvalue over the entire image
3. Keep the top X percent pixels with the highest eigenvalues
4. Find the set of local max pixels from each 3x3 neighborhood
5. Keep the subset of those pixels so that the minimum distance is larger than a threshold distance

The resulting pixels are then deemed to be good track points.

CHAPTER 3

LUCAS KALMAN TRACKING WITH PARTICLE FILTERS

3.1 Tracking with Particle Filters

Particle filters are an extremely powerful kinematic tracking tool, [4] Isard used them to allow a dynamically changing object to be tracked in realtime. While the implementation worked well for previously defined objects, a more generic solution is needed for sensor pod target tracking in aircraft. Since the pilot should be able to track anything that can be seen on a sensor pod, it would be computationally expensive to go with a training based target tracking approach. However, the generic particle filter approach can be used as a kinematic tracker for a generic target tracking implementation. Much of the focus in developing a new particle filter implementation was put into the weighting scheme. In the initial implementation, the particle filter used a simple intensity based approach that would weight particles higher based on the minimization of the intensity space distance. For example, if the target had an average intensity range of $i_{range} = [i_{min}, i_{max}]$ with a median of i_{med} we could define the distance function as

$$d_{rad}(i_{pixel}) = |i_{med} - i_{pixel}| \tag{3.1}$$

where i_{pixel} is defined as the intensity of the pixel of interest. This pixel could be the centroid of a particle. A simple video was taken using a golf ball on a contoured surface. This example was chosen for testing because a physics based particle system would represent the golf ball motion well.

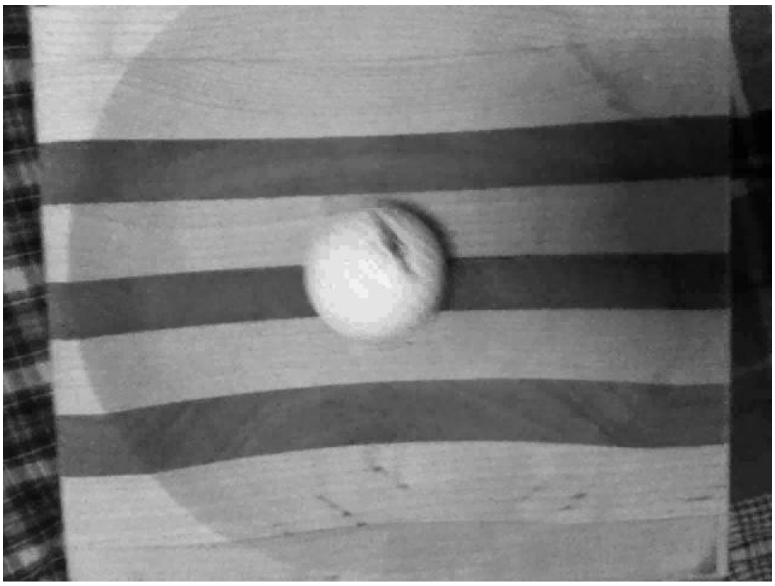

Figure 3.1. Golf ball frame.

Some items to notice about this case is that the ball can be identified by a high intensity range so that it is easy to pick out amongst the background. Using a particle system with a intensity based weighting is ideal for this case if the only concern is making sure that a portion of the ball is tracked. From an aircraft sensor application perspective this would give the sensor control system enough information to keep the target within view. However, it is always good practice to determine the centroid of the target so that when extreme maneuvers, background noise, and clutter is involved the tracking algorithm will be more resistant to change. Under the currently described intensity based particle system, the centroid of the ball can be found by using a large number of particles that will cover the entire ball. From this, the average particle position can be found which will correspond to the center of the target.

3.1.1 Tracking the Centroid

Another option for tracking the centroid would be to find the edges of the frame and each particle could then determine the score based on its distance from the edges of the target. This case is simple because the target that is being tracked is a sphere and the target radius is constant given that the camera exhibits minimal movement and motion blur is minimized.

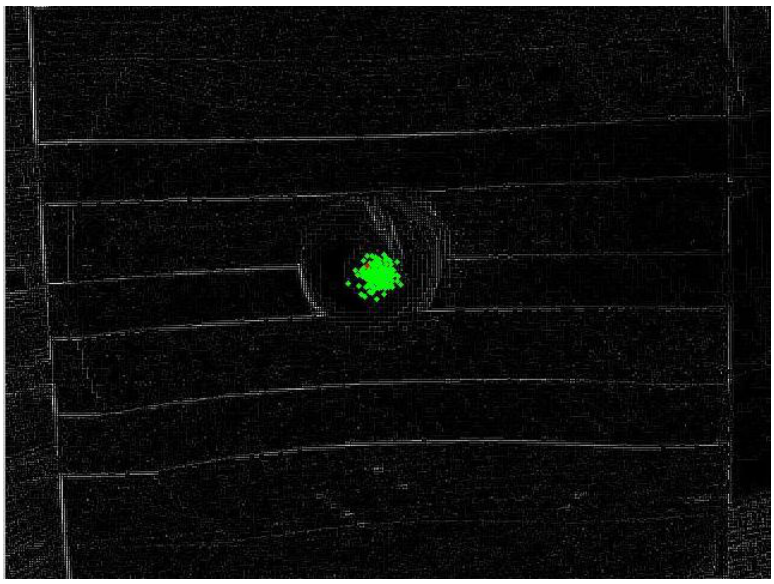

Figure 3.2. Edge filter applied to golf ball.

By taking the Laplacian of the frame, the edges of the ball are highlighted; however the background edges are also highlighted. Specular highlights also cause edges to appear near the center of the ball. Even in this simplified case, there is noise and background clutter that will need to be considered when improving the particle system.

The first step in creating the particle system is to define the motion model. In this case, a simple, velocity based motion model will be used which is defined by for each particle

$$p_k^- = p_{k-1} + v_{k-1} + \sigma \cdot \gamma \tag{3.2}$$

$$\tag{3.3}$$

where

σ : The standard deviation of the particle velocities

γ : A gaussian distributed random function that is called for each particle

p_k : The predicted position of the particle for frame k

p_{k-1} : The calculated position from the previous frame

v_{k-1} : The calculated velocity from the previous frame

These equations project the position in the next frame by adding in a mean velocity that was determined from the previous frame and multiplying it by a deviation scale and a gaussian distributed random number. A Gaussian distribution was chosen for this implementation since it tends to represent natural motion well. The velocity means used by the next frame can be calculated by

$$v_k = p_k - p_{k-1} \tag{3.4}$$

where

v_k : The particle velocity for the current frame

p_{k-1} : The calculated particle position from the previous frame

Note that up to this point, the image processing aspects of the algorithm are not considered. The particles are just following a physical model and would distribute themselves randomly throughout the entire frame given enough time. More complex motion models can be determined on a per application basis and include acceleration models.

The next step considered in the particle system is a weighting algorithm that should be applied to each particle in the system. For the Laplacian based frame, the particle will need to determine the location of the edges. In order to avoid excessive pixel sampling that takes place for each particle a certain range of pixels will be sampled based on the initial value of the target radius. This sampling strip is then multiplied by a Gaussian distribution in order to give particles that are near the centroid a higher weight. The graph in 3.3 shows a plot of a typical curve used to modulate the particle weighting. This curve gives a higher weight to edges that occur near the areas that where edges are expected. The plot in 3.4 shows an example edge plot in a region, higher values correspond to more significant edges. The final plot in 3.5 shows the modulation result when the normal curve is applied to the edge curve. The value of the modulated curve can then be summed to provide a particle weight for the ray originating at the origin at the estimate particle position. This is described by the equations in 3.5

$$w_{r_j} = \sum_{i=1}^{n} \epsilon_i \cdot \eta_i \tag{3.5}$$

where

w_{r_j} : The weight of a particle for ray j

n : The number of sample positions for a ray

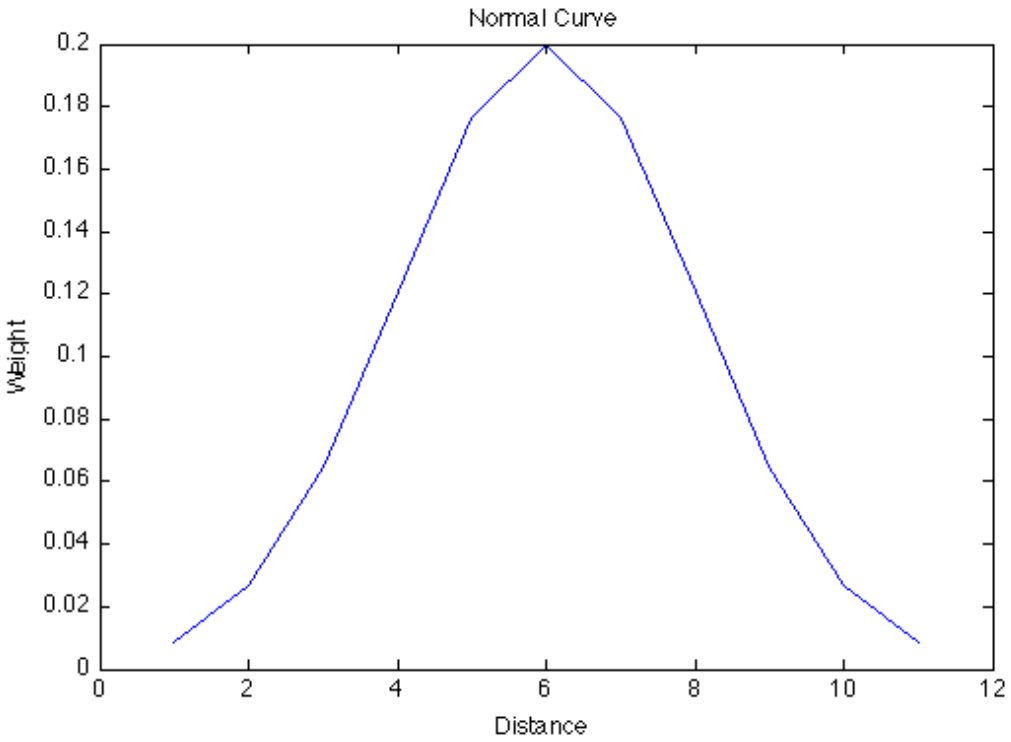

Figure 3.3. Example normal plot curve.

η : The normal curve

ϵ : The sample edges for the ray

In testing, a set of four strip samples where taken based on the North, South, East, and West positions of each particle. Then the weights were totaled using

$$w = w_{r_{north}} + w_{r_{south}} + w_{r_{east}} + w_{r_{west}} \tag{3.6}$$

where

w_r : The total weight for the particle rays

In addition to this, the intensity distance can also be added in with the edge based weights. The weight of each component can be tweaked and is usually and application specific task.

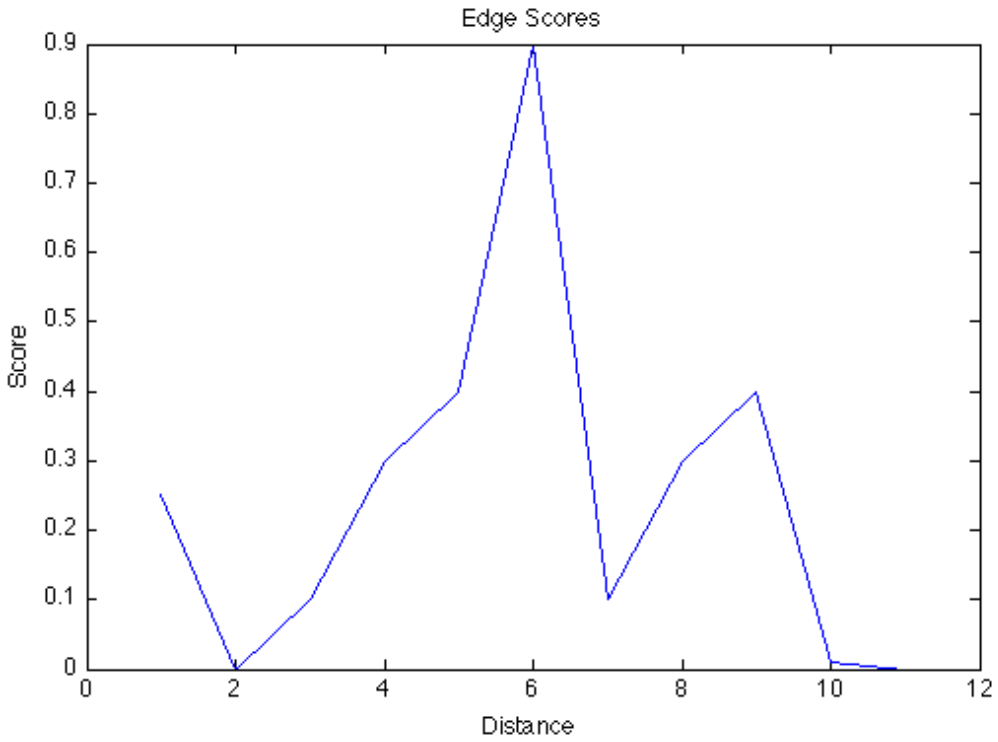

Figure 3.4. Example plot of edges.

$$w = \alpha \cdot w_r + (1 - \alpha) \cdot w_i \tag{3.7}$$

where

w : The total particle weight

α : The weight scale for the edges

w_i : The intensity weight

The factors are used in order to adjust which attribute is more important. These types of values are typically tweaked on a application and sensor basis. For an FLIR sensor, the intensity will be more important due to the fact that targets typically have high heat intensities which correspond to higher pixel intensities. The last step in the weighting model is to multiply the current weights by the weights from the previous frame so that the overall performance of the particle can be evaluated.

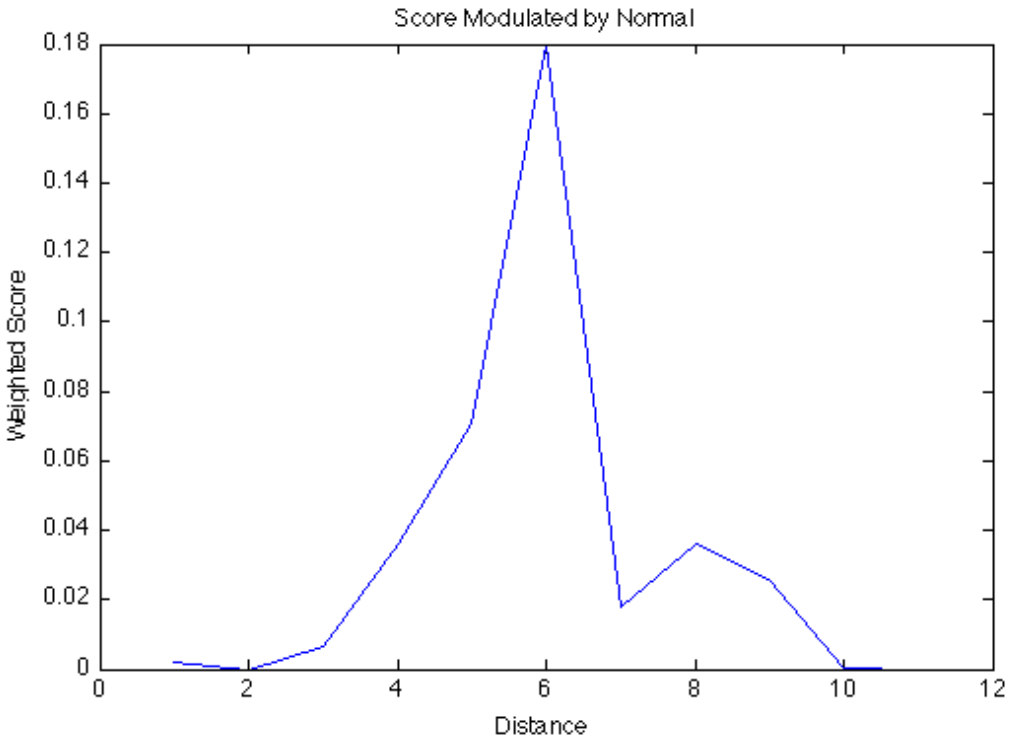

Figure 3.5. Example plot of edges modulated with normal curve.

$$w_k = w \cdot w_{k-1} \tag{3.8}$$

This step makes the algorithm a recursive process, but dynamic programming is used so that the system does not have to keep track of all of the weights for every frame that have passed.

Now that each particle has been weighted, another decision needs to be made in determining the correct position of the target. The simplest approach is to use the highest weighted particle, but that might make the system more susceptible to anomalies. Another option is to take the weighted average which works for this case since only a single target is being tracked. The highest and lowest weighted particles could also be thrown out if they pass a certain threshold. This would prevent the

particles from making a contribution to the estimated target position. Another item to note is that the particle system is working in two dimensions so both scores will need to be accounted for in the final weight. The weights should be normalized based on the minimum and maximum considered particles before resampling occurs

$$w_{norm} = \frac{w_k}{\sum w_k} \tag{3.9}$$

This portion of the algorithm performs the work of removing the bad particles from the system and resampling a new set of particles based on the state of the stronger particles. This is effectively survival of the fittest based on the weightings calculated from the previous step. In some cases, resampling is not needed so an effectiveness score will be calculated by

$$n_{eff} = \frac{1}{\sum w_{norm}} \tag{3.10}$$

and compared to a threshold. This comparison will then be used to determine if any particles need to be resampled. When the resampling phase occurs, a predetermined percentage of the lowest weighted particles (p_{regen}) will be regenerated. The old particles will be replaced by a weighted sampling of the surviving particles such that the best particles will be replicated more often.

3.1.2 Particle Drift

From testing, this edge tracking technique tends to work for some time, but will eventually lose track due to the background edges. For a noiseless video sequence, this approach would work quite well, but the video was degraded purposely in order to test the impact of noise upon the algorithm.

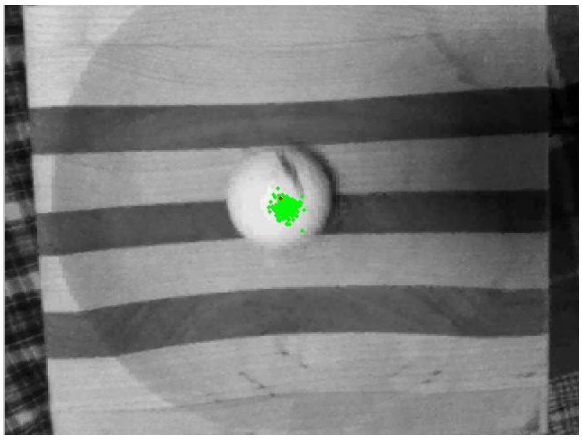

Figure 3.6. Starting positions for particles.

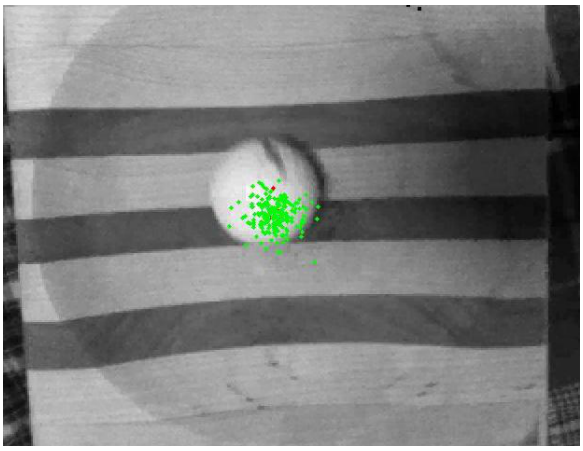

Figure 3.7. Cluster expansion.

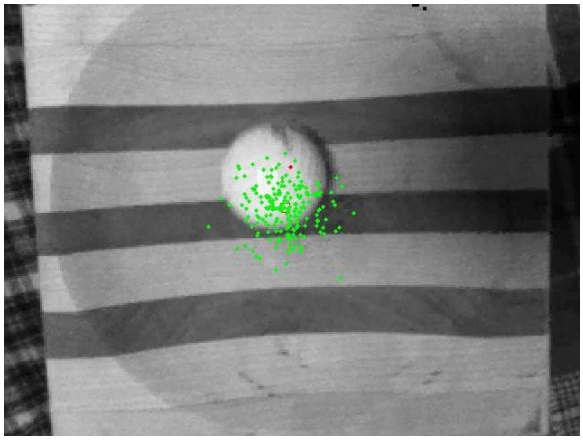

Figure 3.8. Losing track.

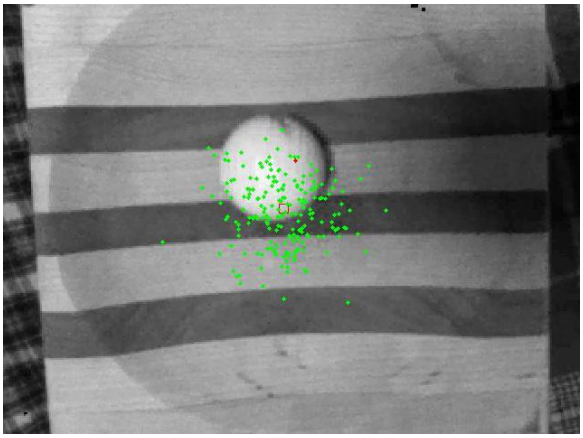

Figure 3.9. Approaching chaos.

Once the particles start scattering outside the bounds of the golf ball, they will find the edges on the background and become highly weighted which leads to tracking system degradation over 100s of frames. Using a smaller edge falloff and increasing the intensity weighting resulted in much better performance.

Although there is still some drift, tweaking the parameters can improve performance dramatically.

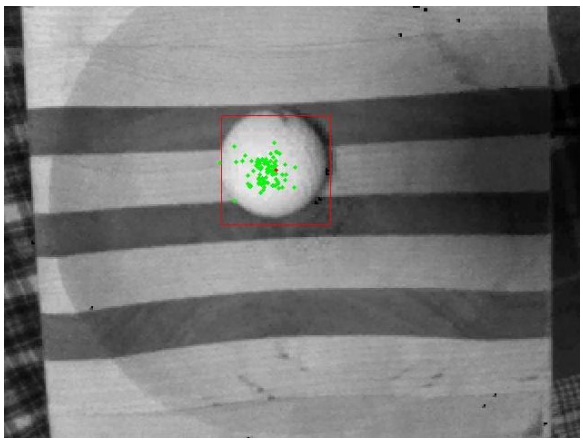

Figure 3.10. Initial Frame.

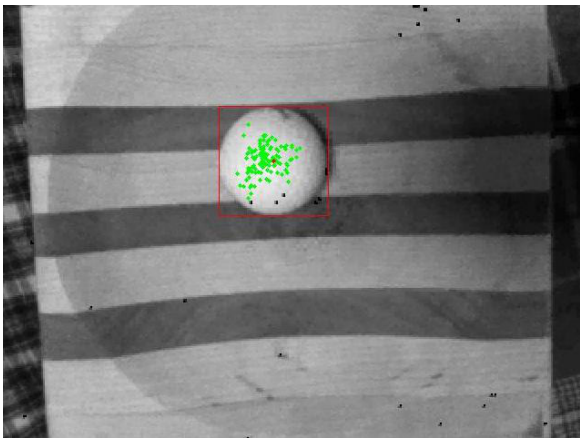

Figure 3.11. Coherent subsequent track.

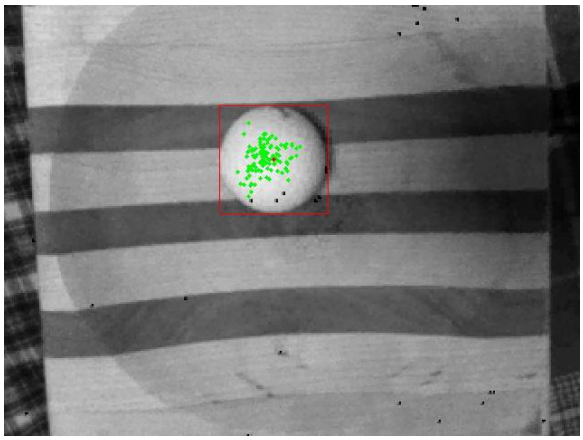

Figure 3.12. Maintaining track.

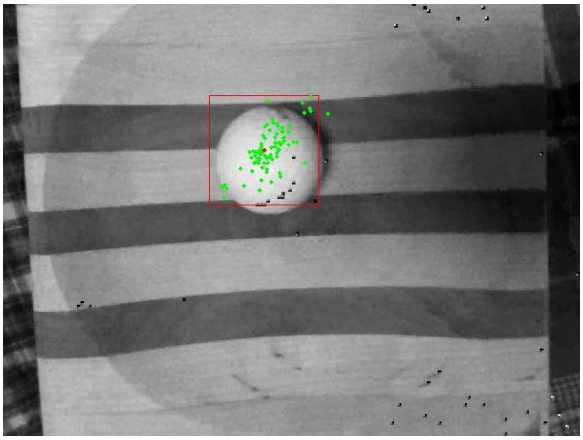

Figure 3.13. Losing track.

3.1.3 Variable Sample Ray Tracking

An addition to the particle filter algorithm was made in order to attempt tracking a target that could have a variable radius. In practice, variable radius targets are prevalent on sensors that have zoom factors that can effectively increase the size of the target. The focus of this change was to add scale invariance to the tracker such that the tracking algorithm could still ascertain the target position through changes in the zoom level of the sensor.

For the curve tracking implementation, the basic approach was to use the existing position and velocity measurement model and to use a new radial edge tracking that would allow the target to change shapes using

$$r_k = r_{k-1} + r_v \tag{3.11}$$

where

r_k : Radius at time k

r_{k-1} : Radius from the previous frame

r_v : The rate of change for the radius

The rate of change is determined in the same way that the particle velocity is determined by

$$r_v = (r_k - r_{k-1}) + \sigma \cdot \gamma \tag{3.12}$$

This calculation will be repeated for every particle ray that originates at the center of a given particle. For the implementation, eight sample angles where used with 100 particles. This is overkill for a spherical target since only the radius will change with the zoom level. But, when tracking a target with a complex shape, the extra sample angles are needed in order to determine the shape change of the target. Instead of using a fixed four direction implementation, the algorithm was expanded to sample rays in n directions.

This approach is similar to the previous technique that used a gaussian falloff in order to calculate the score of each particle. The only difference with this algorithm is that it is more generic and can be sampled n times. For testing, a sample size of

eight was used for each particle in the system. It achieves this rotation by using the simple 2D rotation matrix

$$\begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \tag{3.13}$$

As before, particles whose samples edges are found closer to where the particle predicted are weighted using

$$w_{r_j} = \sum_{i=1}^{n} \epsilon_i \cdot \eta_i \tag{3.14}$$

Where η_i is the set of sample laplacian edge values for one radial sample set, ϵ_i is is the gaussian distribution. As would be expected, performance tended to improve as more particles were added to the system up to a point. When using 100 particles, the ball is tracked well at first, but performance degrades after a period of time.

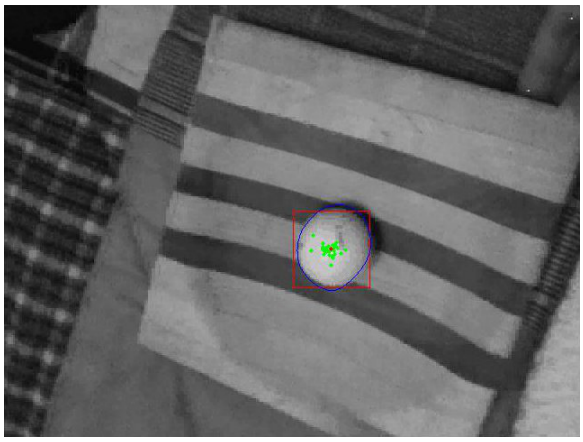

Figure 3.14. Tracking using radial samples.

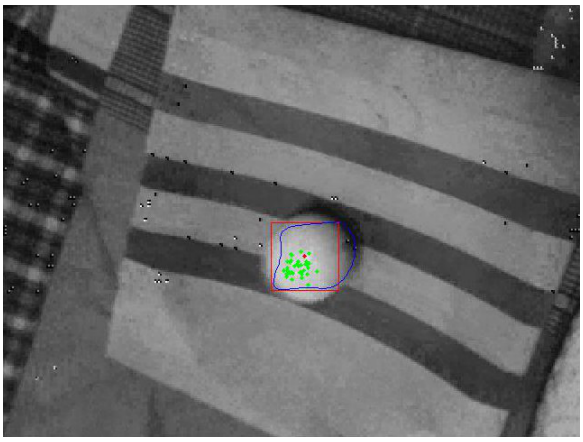

Figure 3.15. Chaotic radial samples.

The first image in 3.14 shows the system tracking the golf ball in the first few frames. It appears that the the particles are keeping to the center of the target as well as choosing accurate points for the edges of the ball, but 3.15 shows that this

performance degrades significantly over time. This can be predicted because the state space of the tracking system has been increased from

$$s = \begin{pmatrix} x & y & dx & dy \end{pmatrix} \tag{3.15}$$

to a state space that has more than twice the number of elements with a value of

$$s = \begin{pmatrix} x & y & dx & dy & r_1 & ... & r_8 \end{pmatrix} \tag{3.16}$$

for the tracking algorithm. This behavior can be described by the combinatorial complexity associated with increasing the state space using

$$|s| = w^2 \times h^2 \times \prod_{k=1}^{8} \sqrt{(h^2 + w^2)} \tag{3.17}$$

$$= w^2 \times h^2 \times \left(\sqrt{(h/2)^2 \times (w/2)^2} \right)^8 \tag{3.18}$$

where in the tested case

$$width = w = 640$$

$$height = h = 480$$

leading to the result of

$$|s| = 1.58 \times 10^{34} \tag{3.19}$$

for the size of the state space using eight radial sample points. The previous case that had only four elements in the state space resulted in a size of 9.44×10^{10}. These numbers assume that the maximum radius from the center is less than or equal to

the diagonal across the frame.

Given the state space issue there are still some tweaks that can be made to improve tracking system. Covariance weights could be used to assure that the sample radii stay roughly similar. But, in the final implementation for a target tracking system, aircraft and other complex objects will be tracked where equal sample radii cannot be a valid assumption. Increasing the size of each radial sample does not improve performance either as this makes the particles more susceptible to noise. The one improvement that had the greatest impact was increasing the number of particles, but this only worked well to a point and the tracking algorithm would erupt into chaos given enough time.

3.1.4 Dynamic Size Model for Particle Filters

The dynamic contours faced many performance issues due to the size of the state space so it is worth investigating ways to reduce this state space while achieving a more dynamic particle filter. The easiest extension to the simple position and velocity model is to add a single radius such that the state becomes

$$s = \begin{pmatrix} x & y & dx & dy & r_1 \end{pmatrix}. \tag{3.20}$$

Although the state space of the particle system has been drastically reduced, the weighting system can still take advantage of the multiple radius samples by factoring in a standard deviation component to the weighting scheme. This component will help assure that the object being tracked is circular without the expense of having the extra state elements. The radius of the target can then be determined by taking the average of the radial samples. In practice, this approach worked much better than

simply taking the highest weighted ray.

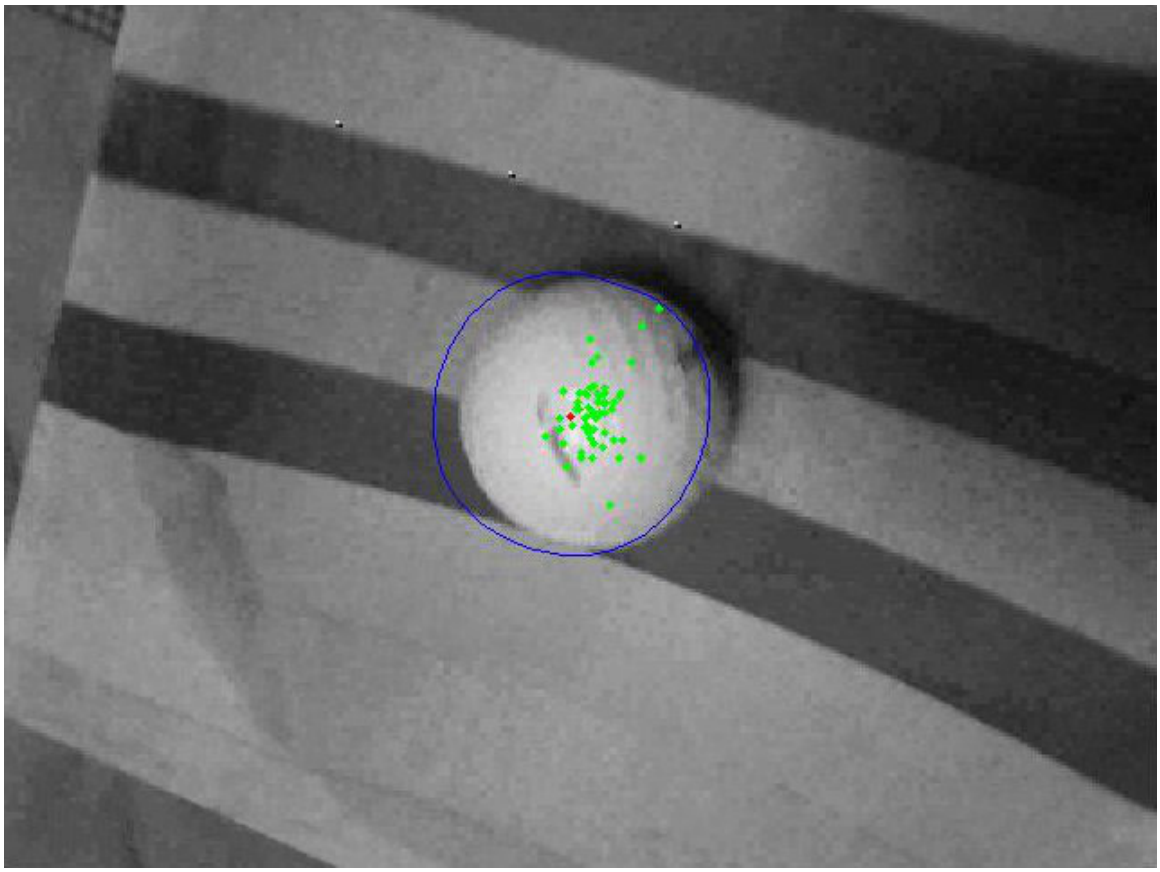

Figure 3.16. Particle filter using dynamic size model.

This approach is still susceptible to noise and in this example would typically lock onto the shadow of the golf ball. This resulted in significant jitter in the size of the tracking circle since some particles would find the correct edges while some would follow background noise. Looking at the result of the edge filter output provides more insight into why this is occurring. The laplacian edge filter does not provide a solid line for the edges so it was beneficial to investigate other edge finding techniques.

Among the algorithms tested were Prewitt, Canny, and Sobel edge detection, but Sobel appeared to provide the best results as it typically highlighted most of the important edges.

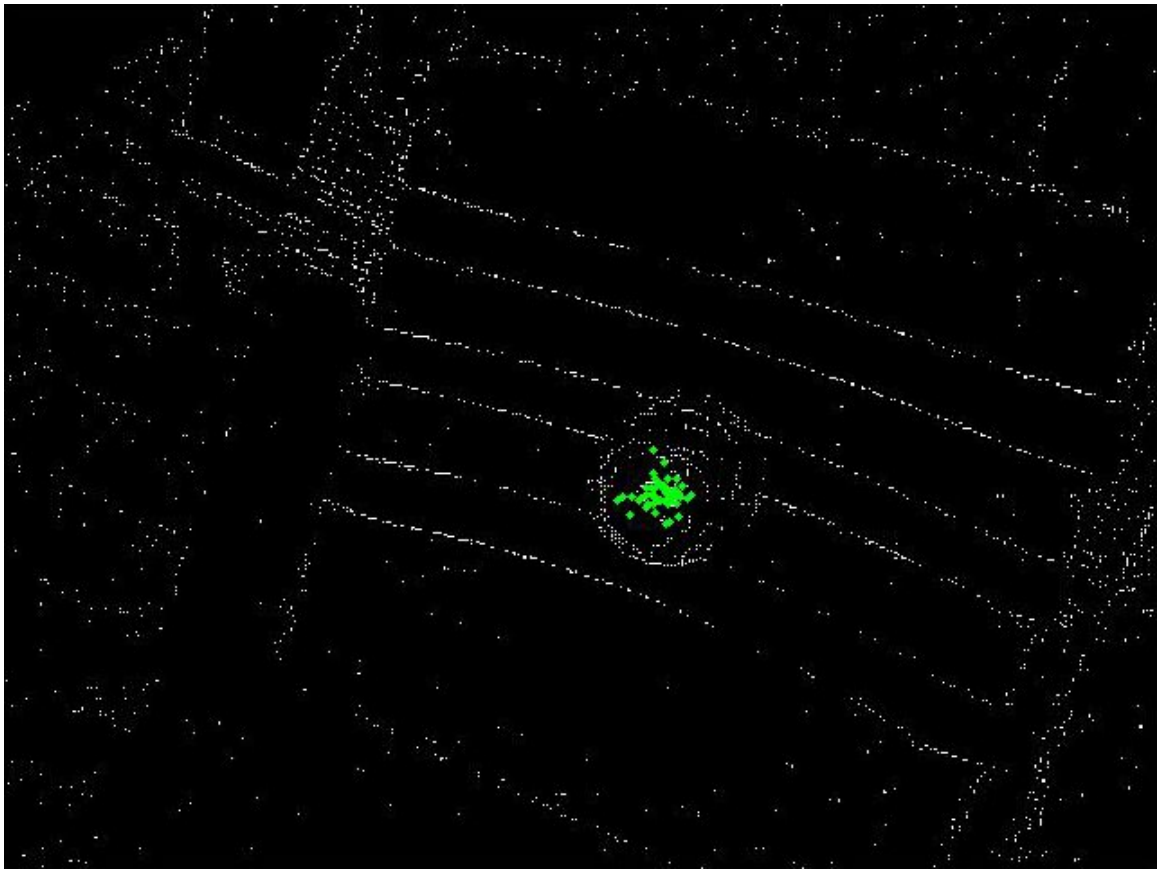

Figure 3.17. Laplacian edge detection.

In figure 3.19, a dilation operation is applied after the Sobel edge detector for the purpose of reducing the edge aliasing. This helped keep the particles more tightly packed since they were more likely to find the correct edges in this case. This model also enabled the algorithm to continue working through changes in the zoom level

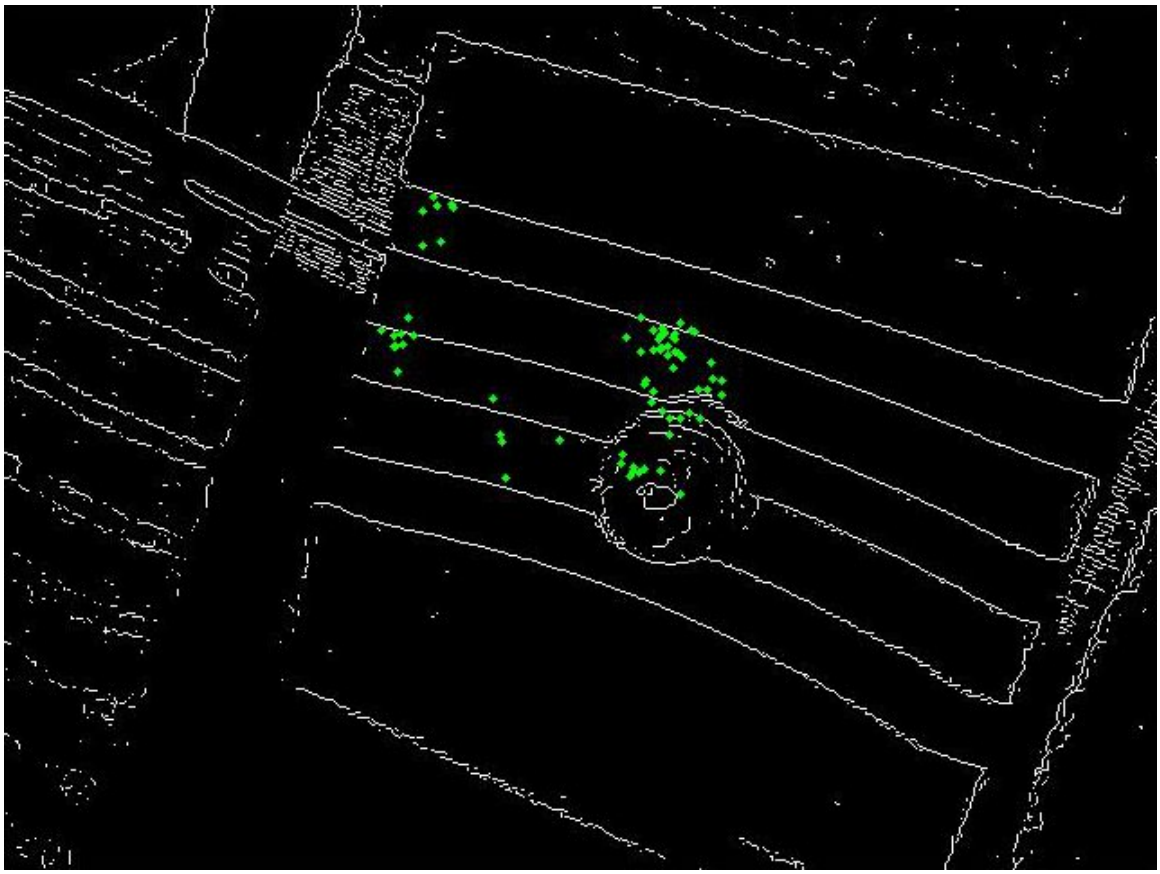

Figure 3.18. Sobel edge detection.

of the camera. There were also some application specific tweaks that were made to improve performance. Each particle was given a reasonable minimum and maximum radius in order to avoid cases that would skew the results. Since the weighting is now dependent on both standard deviation and the value of each radius sample with respect to the radius state of the particle, the coefficients needed to be tweaked in order to achieve the desired performance.

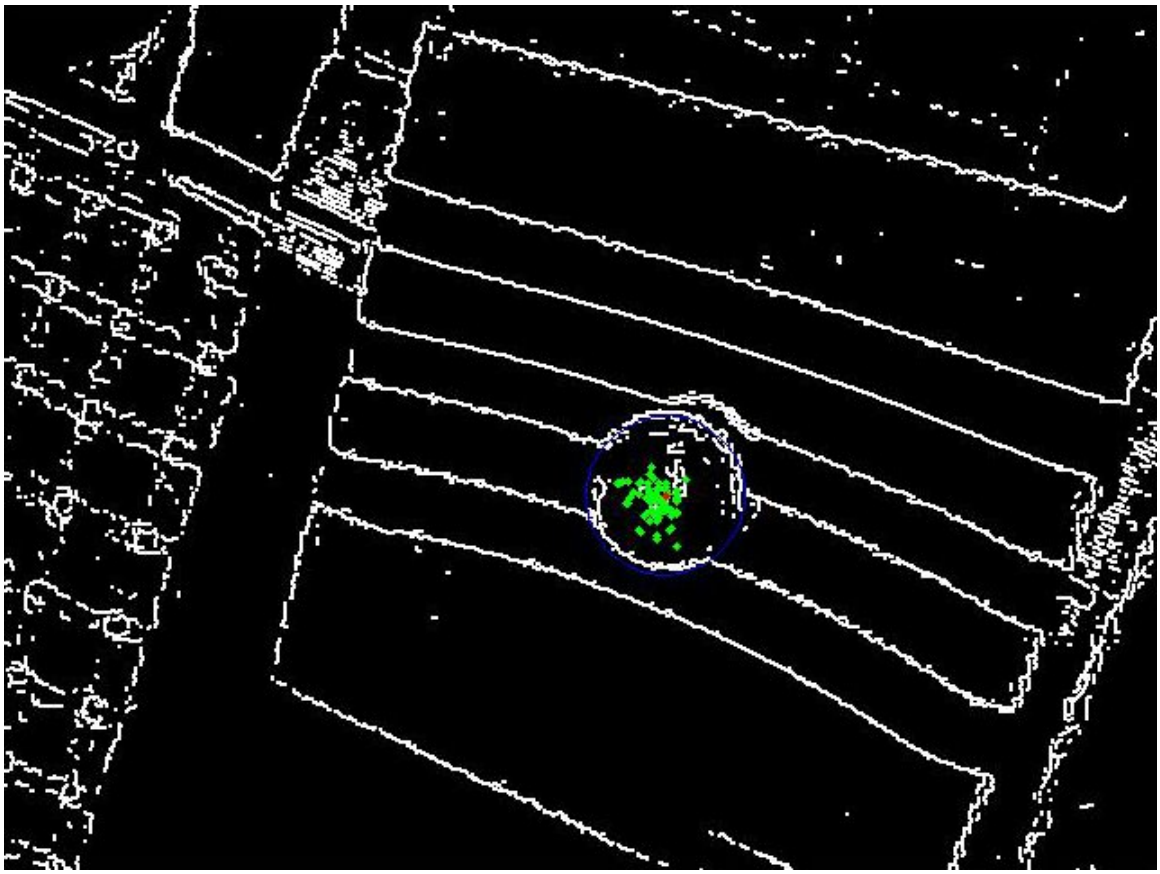

Figure 3.19. Sobel edge detection with dilate.

3.2 Acquisition with Particle Filters

Since the particle filter was tweaked to track a radial target in the previous algorithm description, it can be tweaked to detect radial targets given a few more constraints. If the min and max intensities are adjusted to a small range, then the standard deviation and intensity tracking will find the target. The deviation of the particle filter needs to be increased in order to spread the particles over the state space. After the first frames, the standard deviation of the gaussian distribution will need to be decreased so that the particles will converge to a single point in the state space. The resample threshold needs to be set higher so that the higher scor-

ing particles will effectively be refined until a high match weight is reached. In this implementation, it is assumed that there is only one target that needs to be tracked.

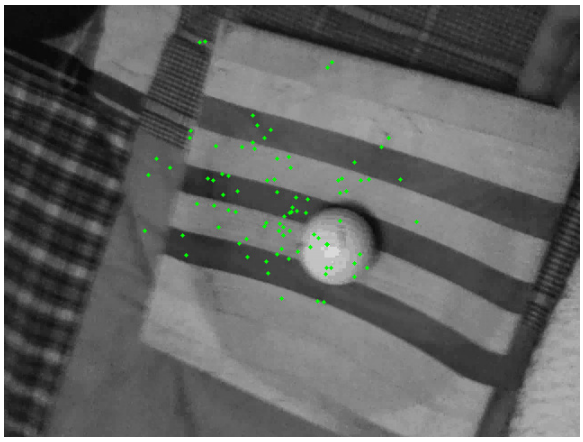

Figure 3.20. 1st frame of particle acquisition.

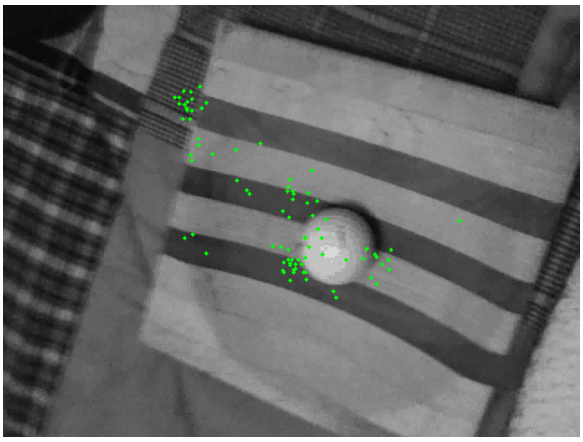

Figure 3.21. 4th frame of particle acquisition.

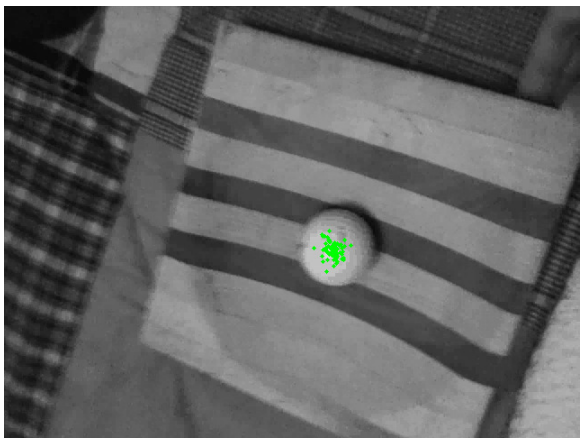

Figure 3.22. 6th frame of particle acquisition.

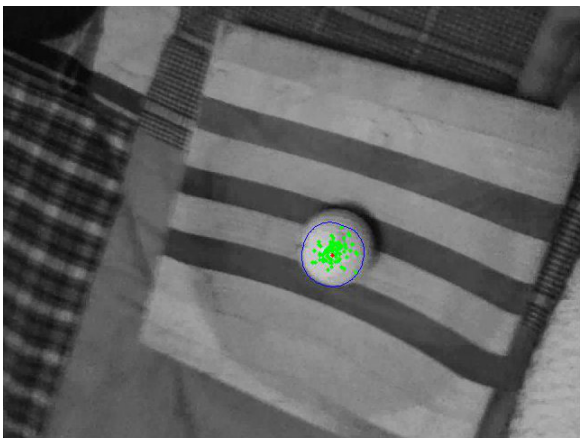

Figure 3.23. Target acquired with particle filter.

Figure 3.20 shows one of the first frames in the particle filter acquisition phase, the subsequent images show the particles eventually converging on the target.

3.3 Mean Shift Tracking

Given that the particle filter implementations thus far are performance intensive for tracking an object through affine transformations it would be interesting to look at some different approaches that could help improve the particle filter algorithms. One of the algorithms that was investigated was mean shift tracking using a color histogram [11]. Since histograms are independent of the pixel positions within a template region this will aide in tracking a target through affine transformations. The test cases used for testing this algorithm consisted of a series of snapshots beginning at the starboard side of a model aircraft and orbiting about the target's center of gravity.

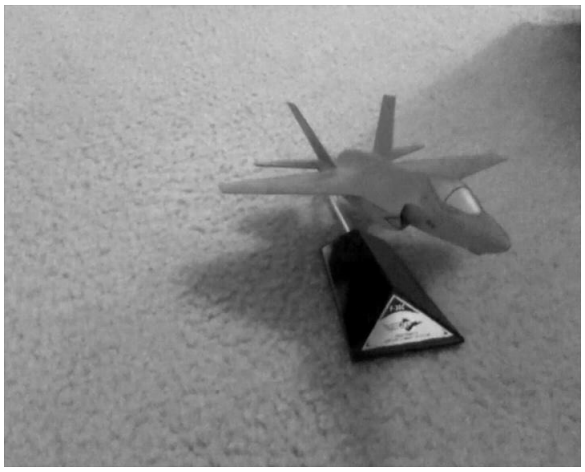

Figure 3.24. First frame of model jet.

Figure 3.25. Overhead view of model jet.

3.3.1 Bhattacharyya Distance

One of the more common color histogram distances to use is the Bhattacharyya distance which is given by the coefficient [11]

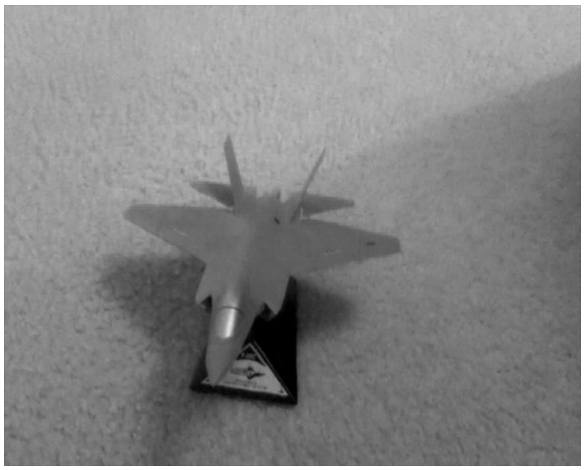

Figure 3.26. Port side of model jet.

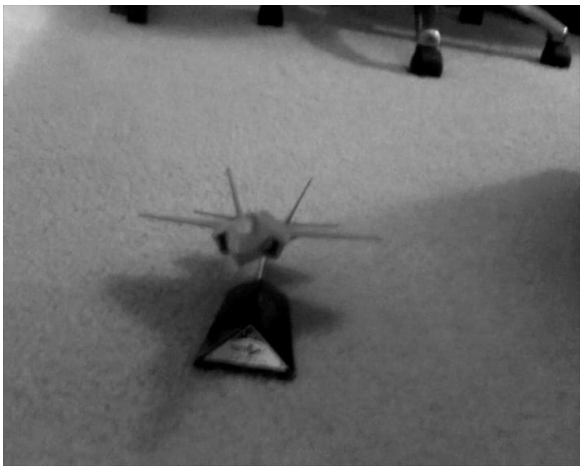

Figure 3.27. Last frame of model jet.

$$B = \sum_{i=1}^{n} \sqrt{\Sigma p_i \cdot \Sigma q_i} \tag{3.21}$$

and used with

$$D = \sqrt{1 - B} \tag{3.22}$$

in order to calculate the histogram distance. In this case p is the candidate image and q is the template. A construct that is normally used with mean shift is to use a distance kernel such that pixels on the edge of the template will get weighted less than those in the center. This is used to counter occlusion that has a higher chance of occurring around the edges of a target. This is done during histogram creation by multiplying a distance equation such as

$$1 - ||x||^2 \tag{3.23}$$

with the corresponding color intensity for that distance [9]. This algorithm was applied to the model jet test case along with spatial correlation and Fourier transform coefficient distance. The distance of for each test frame was then plotted for each

of the distance algorithms in 3.28. The last three tested frames are solid black, white, and gray respectively that serve as test patterns for the algorithms. All three algorithms show a spike at these points. Although the algorithms follow a general trend as the sensor orbits around the target, the Bhattacharyya distance is much more sensitive to small changes. These results can be interpreted as meaning that the Bhattacharyya distance is more susceptible to jitter while tracking. And although the Bhattacharyya distance as implemented here would be more susceptible to miss 3D rotations, it would not be misled by 2D sensor rotations since it is based on an intensity histogram.

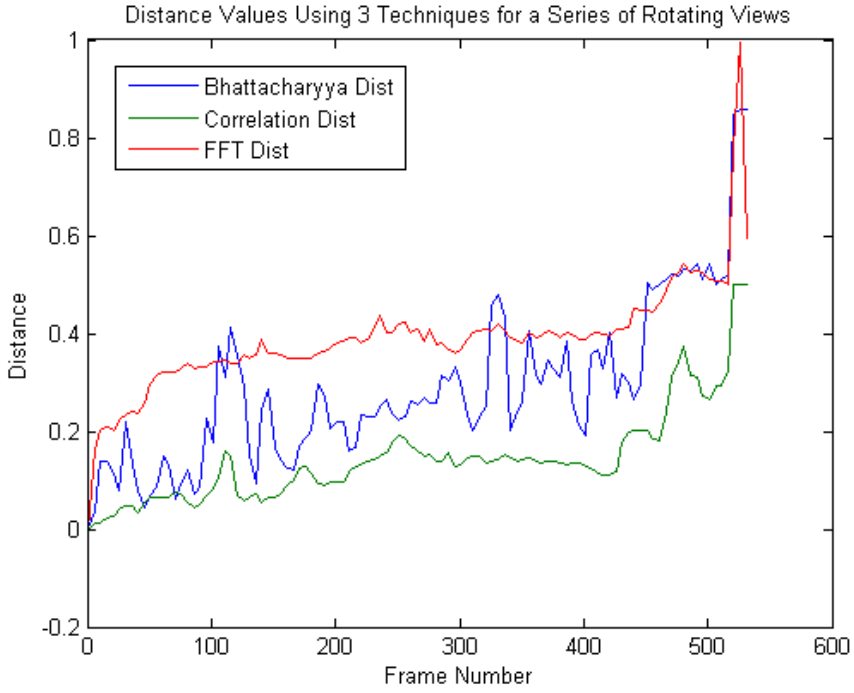

Figure 3.28. Histogram distance of model jet.

One of the greatest advantages of a color histogram distance implementation is the speed.

Table 3.1. Distance Algorithm Timings

Algorithm	Time Per Frame (ms)
Color Histogram	9.35
Correlation	325.79
FFT	73.89

Note that color histogram distance calculation is over 30 times faster than correlation and about 7x faster than FFT compare. This is even more impressive since the FFT and correlation operations are implemented in C and the color histogram kernel algorithm is written in Matlab and there could still be more optimizations. Note that these numbers were created with a 173x216 image. In practice, the system would use a much smaller template, so the timing would put us in real time range for a color histogram approach. Also, note that the number of bins used was 64, in practice, the system could use 32 or even less. With a mean shift algorithm, typically only four shifts are needed so this still puts it under the FFT timings.

3.4 Subtarget Tracking

As part of the investigation into the generic target tracking system implementation a subtarget based approach was implemented. This approach picks a search area based on user input and divides the area into subtargets or features. The goal of this approach is to choose the most interesting features on the target and to track those features between frames. Assuming the subtarget sizes are small enough this

approach will yield an algorithm that can track the target through affine transformations without the penalty of the multiple matrix transformations that are needed to test the orientation and scaling of the target.

The initial subtarget tracking implementation consisted of six steps that included

1. Create a high pass frame
2. Split the target window into subtargets
3. Find the best N subtargets based on standard deviation
4. Use normalized cross correlation to find the position of the subtarget in the next frame
5. Use a weighted average of the movement vectors to determine the next target position
6. Repeat the first step for M frames

The following image shows the tracking boxes for the aircraft wing where the light blue boxes represent the subtarget positions at the previous frame, the blue boxes represent the new position of the subtargets and the red lines represent the distance each subtarget has traveled in the frame. The pink box represents the new target position based on the weighted vector set of the subtargets. The yellow boxes, which are actually 3 times bigger than the subtarget, show the search distance for each subtarget. The legend in table 3.2 explains the meaning of the targeting symbology.

The sequence of images starting with 3.30 show a tracking sequence using the subtarget algorithm. The tracker works well at first, but as the speed of the aircraft increases and causes motion blur it will lose the target. This is a difficult test case as the background of the plane changes drastically with the last frame showing the

Table 3.2. Target Tracking Legend

Yellow Box	Subtarget search area
Blue Box	New subtarget position
Red Line	Subtarget motion vector
Light Blue	Old subtarget position

plane directly against the sky with edges that are completely different from the initial set of frames.

The edge maps shown in figures 3.33-3.34 for this set of frames shows the issues with edge degradation for template tracking. This is because the edges start to disappear as the plane gains speed. The subtargets will then not be able to find similar features within the next frame eventually leading to template degradation.

In this case, the subtargets will need to be larger than initially anticipated because they will need to be able to find features that will exist from frame to frame. However this makes the tracking algorithm more susceptible to lose track during affine transformations.

3.5 Lucas Kanade Feature Tracking

The previous subtarget tracking framework was created in such a way that any subtarget based algorithm could be used. In the next attempt, a Lucas Kanade feature tracker was used in order to determine the movement of each subtarget. In the previous version of the algorithm the subtargets where segmented in predetermined intervals, but for this implementation the subtargets could be chosen anywhere within the targeting area. The feature finding algorithm was also modified to include the algorithm described by [2] which uses eigenvalues in order to determine the trackable features. This involves keeping track of the highest eigenvalues over the search area.

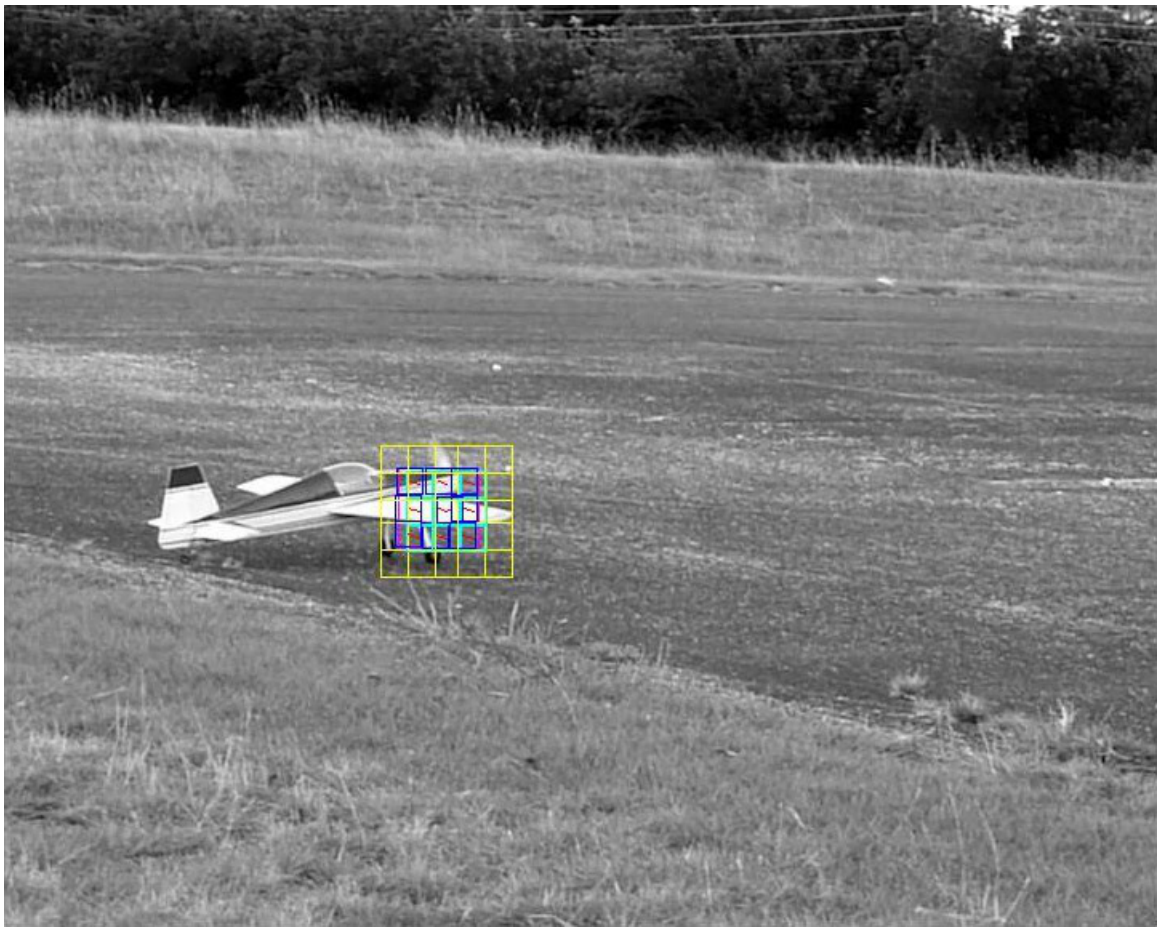

Figure 3.29. Wing track with subtarget based algorithm.

The algorithm is rather simple but some implementation tricks are needed in order to prevent a cluster of subtargets to be found in the same area. The quick approach used in this implementation was to use the following algorithm given that ri and ci are the row and column indices of the image, respectively and I is the image:

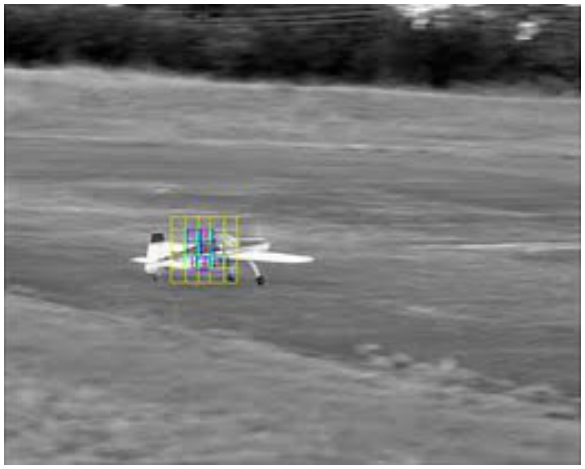

Figure 3.30. Representation of initial wing track frames.

Figure 3.31. Takeoff sequence for wing track.

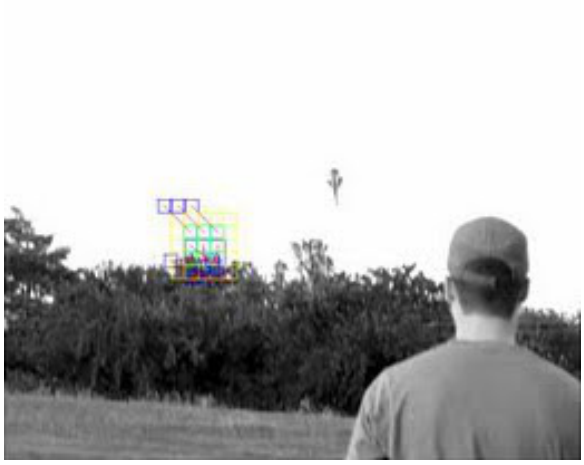

Figure 3.32. Takeoff track lost for subtarget tracking.

1. Compute IX and IY for each pixel

$$IX_{ri,ci} = (I_{ri,ci-1} - I_{ri,ci+1})/2 \tag{3.24}$$

$$IY_{ri,ci} = (I_{ri-1,ci} - I_{ri+1,ci})/2 \tag{3.25}$$

$$IXX_{ri,ci} = (IX_{ri,ci})^2 \tag{3.26}$$

$$IXY_{ri,ci} = IX_{ri,ci} \cdot IY_{ri,ci} \tag{3.27}$$

$$IYY_{ri,ci} = (IY_{ri,ci})^2 \tag{3.28}$$

$$\tag{3.29}$$

2. For each pixel position compute the gradient matrix and store an eigenvalue of this matrix as the score

$$G = \sum_{ci=p_x-w_x}^{p_x+w_x} \sum_{ri=p_y-w_y}^{p_y+w_y} \begin{pmatrix} IXX_{ri,ci} & IXY_{ri,ci} \\ IXY_{ri,ci} & IYY_{ri,ci} \end{pmatrix} \tag{3.30}$$

$$S_{ri,ci} = min(\lambda_G) \tag{3.31}$$

3. Store each pixel position in the score matrix S and separate the high scoring pixels by flag matrix F and region size k and flag region size f

$$F = \begin{pmatrix} 0 & ... & 0 & 0 & 0 & ... & 0 \\ 0 & ... & 0 & ... & 0 & ... & 0 \\ 0 & ... & 0 & 0 & 0 & ... & 0 \\ 0 & ... & 0 & 1 & 0 & ... & 0 \\ 0 & ... & 0 & 0 & 0 & ... & 0 \\ 0 & ... & 0 & ... & 0 & ... & 0 \\ 0 & ... & 0 & 0 & 0 & ... & 0 \end{pmatrix} \tag{3.32}$$

$$S_{rm,cm} = max(S_{ri-k/2...ri+k/2,ci-k/2...ci+k/2}) \tag{3.33}$$

$$S_{rm-f/2...rm+f/2,cm-f/2...cm+f/2} = F \tag{3.34}$$

4. Take the top n eigenvalues and use those for the trackable features

This algorithm effectively finds the edges of the image which contain the most information and uses the flag matrix to assure that all of the features are separated by a reasonable amount of space. This can be specified as an input to the algorithm and will vary with respect to the tracking application. In order to reduce susceptibility to noise, the gradient was taken over four pixels with

$$IX_{ri,ci} = \sum_{k=-2}^{2} sgn(k) \cdot k^0 I_{ri,ci}/4 \tag{3.35}$$

where the same affect could be achieved by smoothing the entire subregion.

The algorithm for finding the feature in the next frame is similar in notation and functionality to the feature initialization algorithm [26]. The feature finding algorithm will iteratively adjust the estimated position until a minimum threshold distance is reached or a maximum number of iterations are performed. For notational purposes P is the previous frame and v is the global movement vector that is incremented every frame.

1. Compute the gradient matrix for I as was done for the feature finding algorithm

$$IX, IY, IXX, IYY, IXY, G \tag{3.36}$$

2. Define the algorithm thresholds

   i_{max} : Maximum number of iterations

   d_{thresh} : The minimum distance that each iteration must traverse or otherwise exit

3. Enter loop until i_{max} or d_{thresh} thresholds are reached

$$\delta_{ik} = P - I \tag{3.37}$$

$$\delta_{ik_{ix}} = \sum_{ci=p_x-w_x}^{p_x+w_x} \sum_{ri=p_y-w_y}^{p_y+w_y} \delta_{ik} \cdot \cdot IX_{ri,ci} \tag{3.38}$$

$$\delta_{ik_{iy}} = \sum_{ci=p_x-w_x}^{p_x+w_x} \sum_{ri=p_y-w_y}^{p_x+w_y} \delta_{ik} \cdot \cdot IY_{ri,ci} \tag{3.39}$$

$$b_k = [\delta_{ik_{ix}}, \delta_{ik_{iy}}] \tag{3.40}$$

$$\eta_k = G^{-1} b_k \tag{3.41}$$

$$v = v + \eta_k \tag{3.42}$$

Once this is complete, the feature should be shifted by v pixels. This algorithm iteratively refines the distance jump (η_k) every iteration based on the subtarget differences and the gradient of the initial image. A test case used for this algorithm was to track a face of a deer, as seen in the frames in 3.35, the algorithm picks a good feature to track. In practice, the number of subtargets would be much larger and the size of the subtargets much smaller.

From testing, the Lucas Kanade implementation could keep track of targets throughout affine transformations and occlusions. Although there are still some improvements to be made to handle occlusion and maintain stability.

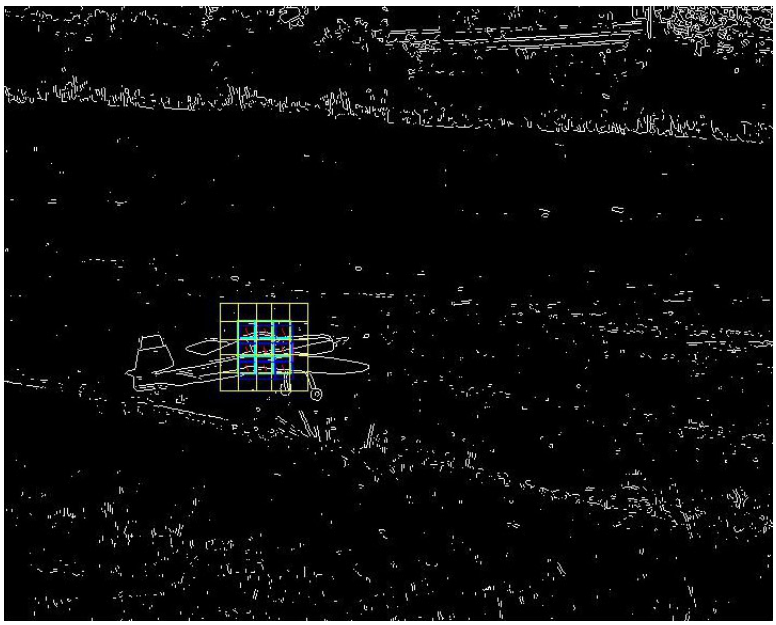

Figure 3.33. High pass filter before aircraft takeoff.

Figure 3.34. High pass filter after aircraft takeoff.

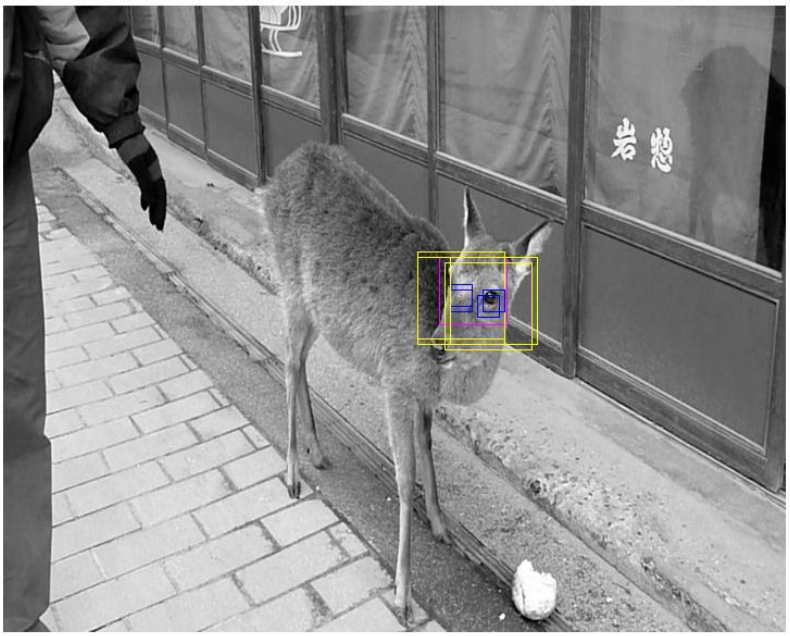

Figure 3.35. Face track of deer.

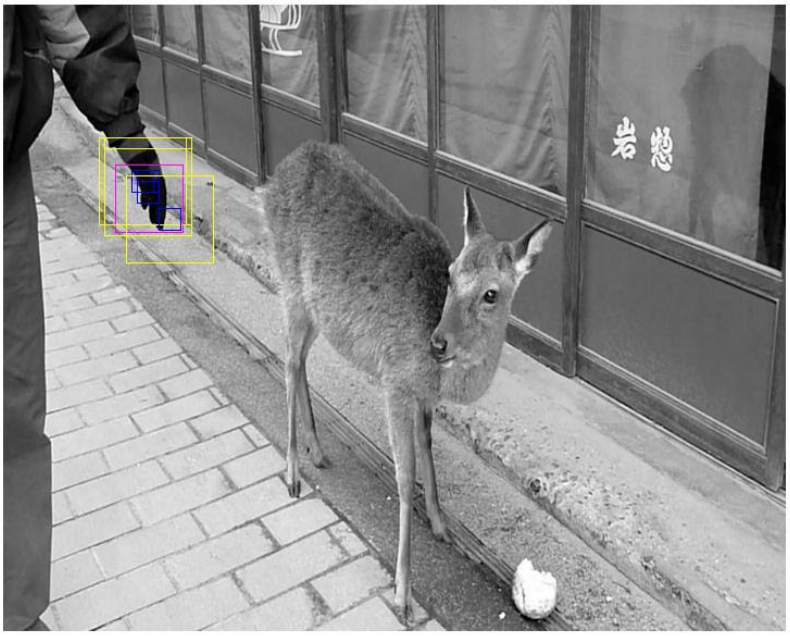

Figure 3.36. Hand tracking.

CHAPTER 4

RESULTS

The previous chapter discussed a variety of algorithms that could be useful for tracking a generic target. The subtarget based Lucas Kanade approach shows promise and particle filters are a great way of adding a kinematic component to the tracker when the state change functions are non linear. The final implementation for the generic target tracker includes a Lucas Kalman and particle filter hybrid with some modifications.

In and of itself, the Lucas Kanade tracker does not require a particle filter since it finds a single feature in the next frame in roughly five iterations. However it can be used to supplement a Lucas Kanade tracker when the tracker is no longer able to maintain a good track on the target. One of the tricks used to accomplish this is to determine when the target should follow the Lucas Kanade tracker or the particle system output.

4.1 Particle Filter Implementation

For this particle system implementation, the weighting system will have two modes that can be activated at a time that is convenient for the tracking system. The first mode will take the final output of the Lucas Kanade tracker and use those those as inputs to the particle system measurements. The second phase will rely solely on the pixel comparisons within the weighting scheme for each particle and be independent of the Lucas Kanade tracker since it can no longer be trusted.

4.1.1 Lucas Kanade Tracking Phase

The first phase for the particle system is used to prepare the system for a possible second phase where an occlusion case is encountered. At this point in the algorithm, the particle system is not being used for active tracking and the outputs of the particle system may even be hidden from the user. However the particle system is still going through all of the previously described steps which consists of

1. Predict the next frame by the constant velocity motion model
2. Using the current frame, weight each particle based on its distance from the Lucas Kanade targeting output.
3. Adjust the speeds based on the gaussian random distribution
4. Resample the particles based on the weight
5. Repeat step 1 until complete

Under this phase, the particles will follow the Lucas Kalman results and although this may seem unnecessary, it provides information to the particle system so that it can adjust the particle velocities which will become important in cases with occlusions. Similar results could be achieved by activating the particle system once the Lucas Kanade tracker fails, but it would require tweaking the initial conditions more.

The weighting framework is rather simple for this case and results in a light computational load which would be virtually transparent from a performance perspective unless the number of particles becomes unmanageable. This is because the weights are calculated with

$$p_w = \sqrt{w^2 + h^2} - \sqrt{(x_x - z_x)^2 + (x_y - z_y)^2} \tag{4.1}$$

where

p_w : particle weight

w : width of the image

h : height of the image

x_x : the x position of the particle

x_y : the y position of the particle

z_x : the x position of the Lucas-Kanade measurement

z_y : the y position of the Lucas-Kanade measurement

The prediction phase of the particle system is not computationally complex as all that needs to be computed is the constant velocity projection into the next frame with

$$x_{k+1} = x_k + v_k + \gamma \cdot \sigma \tag{4.2}$$

where

x_{k+1} : the position state for the next frame at $k+1$

x_k : the position state for the current frame at k

v_k : the velocity state for the current frame

γ : the gaussian distribution

σ : the standard deviation of the gaussian

The velocity should also be adjusted on a frame to frame basis for each particle. Otherwise the velocity would not be a state of the particle system and this is computed by

$$v_{k+1} = x_k + x_{k+1}. \tag{4.3}$$

The only other step used in the system is to resample the particle system based on the weighting of the particles computed from the measurement distance to each particle. This resampling is highly tunable and is based on the number of particles that the user would like to resample each frame. Resampling a large number of particles will keep the particles tightly clustered and implies that there is a high certainty around a small region of the state space. Alternatively, a low resample weight will spread the particles out more so that they can explore more of the state space. This means that there is less certainty in the measurement so a higher percentage of the state space is available.

4.1.2 Pixel Comparison Phase

The second phase occurs when the Lucas Kanade algorithm returns a high pixel difference score which may correspond to occlusion or a state of rapid change for the target. At this point, a template snapshot is taken from the previous frame and stored for use by the particle filter weighting logic. Although it would be possible to take an intensity histogram snapshot or edge snapshot, the implementation used an intensity pixel snapshot. Each snapshot type has a use depending on application specific criteria and could be used as an input into a generic target tracking system. For this case, the template is assumed to be static and is not modulated until this tracking phase has passed. If the target undergoes an affine transformation while being occulted the tracking system will most likely not be able to find the target once the occlusion is finished. Also, if the target changes velocity drastically, there will likely be few particles at the points where the target emerges. In this case, a histogram based approach might lend itself to affine transformation invariance might be a better choice for this phase. But, each technique has its disadvantages as the

color histogram approach might lead to background noise and clutter susceptibility.

Once the template snapshot is formed, the particle filter will still use the same velocity and position models that were used in the previous phase. The key differences will be in the weighting scheme and how to display the target state to the user. The weighting scheme used for this implementation involved the use of the sum of absolute differences between the snapshot template and the area around each pixel.

$$p_w = w_{template} h_{template} - |P - T| \tag{4.4}$$

where

p_w : the particle weight

$w_{template}$: the width of the template

$h_{template}$: the height of the template

P : the area of interest surrounding the particle

T : the static template snapshot from the start of this phase

The highest weight corresponds to the minimum difference between the static template and the area of interest for the best particle. There are many distance schemes that could be used for this calculation, but this one seemed to work in practice. A common practice is to use the mean square error, but that did not make a noticeable impact on the performance of this implementation.

The other item that needs to be addressed for this phase is a methodology to determine where the symbology should be displayed to the user. In previous cases, the weighted average of all the particles was used, but this case is slightly different since each particle has a larger area of interest and is less likely to be distracted by

noise. The disadvantage with this is that particles will take longer to compute their weight. However, since the particles will be more accurate, the system can use fewer of them if a performance boost is needed. Given the greater detail in the weighting algorithm, the highest scoring target can typically be chosen if the particle window is large enough.

This phase will end once the frame difference of the best particle reaches a threshold. It is at this point that the assumption can be made that the target is visible again and Lucas Kanade tracking can resume.

4.2 Lucas Kanade Implementation

The Lucas Kanade implementation follows the same two stage approach as the particle filter and complements the particle filter approach when the target is undergoing affine transformations. The first phase will consist of taking the output from the Lucas Kanade algorithm as the position of the target. When the frame difference reaches a threshold, the Lucas Kanade algorithm will stop tracking and freeze its templates such that they are not corrupted by the occlusion pixels.

4.2.1 Lucas Kanade Tracking Phase

When the target is initialized, the tracking algorithm will initialize the Lucas Kanade tracker with the target pixels in the first frame. The predefined targeting area will then be searched for features based on the gradient eigenvalue algorithm described in [25]. Once the features are found, the tracker will then attempt to find the features in the next frame by using Lucas Kanade optical flow. Once this is accomplished, the algorithm will traverse the results for each feature or subtarget and terminate the subtargets if the difference score between the previous frame and the

current frame reaches a predefined threshold. The terminated subtargets will then be replaced by resampling the targeting area using the same feature finding algorithm that was used during initialization.

Disregarding the thrown out subtargets, the new target position is based on a weighted average of the subtargets that passed the threshold test. The templates for each subtarget are updated with the pixels from the current frame. This is dangerous for simple template tracking as it makes the system more susceptible to background noise. But, multiple subtargets are being used and thrown out if they fail to meet the criteria so this becomes less of an issue. At the same time, this approach allows the subtargets to change between frames contributing to a more robust tracking system that is resistant to affine transforms. Overall the Lucas Kanade portion of the algorithm will follow the steps listed below.

1. Initialize the Lucas Kanade tracker with subtargets based on the gradient eigenvalue
2. Use Lucas Kanade feature finding to track the features/subtargets in the next frame
3. Filter the subtargets that fail to pass distance and score thresholds
4. Use the remaining subtargets to move the target centroid by the weighted average of the subtargets
5. Resample the targeting area for new subtargets if terminations occurred in this frame

4.2.2 Pixel Comparison Phase

Once the target fails to pass the threshold criteria the Lucas Kanade tracker stops executing and does not update the subtarget templates in order to avoid tem-

plate corruption during occlusion events. At this point, the particle system controls the tracking and the feature tracker does not need to run until the particle system threshold is reached. However, once that event occurs, the feature tracker will be reinitialized with the last template from the particle filter.

4.3 Phase Determination

Given the two phases used in the tracking algorithm, there are many options for determining when a phase transition should occur. The criteria for going from the feature tracker phase to the particle filter phase include

$$|I - T| >= \epsilon_t \tag{4.5}$$

$$||x_k - x_{k+1}|| >= \epsilon_d \tag{4.6}$$

where

I : pixels from the current target in the current frame

T : pixels from the previous target in the previous frame

x_k : target position from the previous frame

x_{k+1} : target position from the current frame

ϵ_t : threshold for difference score

ϵ_d : threshold for positional distance score

This just uses the magnitude of the position vector differences and the difference between the pixel intensities in order to determine when to undergo a phase transition. When switching from the particle filter phase to the feature finding phase there is only a single intensity constraint since the particles follow a physics-based motion model.

$$|I - T| <= \epsilon_f \tag{4.7}$$

where

I : image pixels from the current particle search area

T : static template pixels

ϵ_f : threshold for difference score

4.4 Test Cases

The algorithm was verified through a series of test cases that exercise the various components of the algorithm. The first test cases were simple scenarios that showed the algorithm could handle basic cases.

4.4.1 Box Tracking

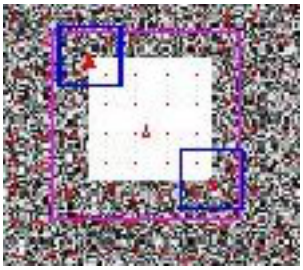

Figure 4.1. Gradients for box test.

Figure 4.1 shows that the algorithm found features around the corner of the white box. This is where the most information is contained and will result in excellent tracking points. If the algorithm had found an edge instead of a corner, it would be allowed to slide along the edge as the target moved and eventually lead to a lost track. Figure 4.2 shows a zoomed in snapshot of the previous case where a corner is

being tracked. The red lines show the resultant gradients where the thick red lines are added for emphasis. The gradients are larger near the edges of the box than in the rest of the image even though it consists of a randomized background. This was based on a four sample gradient in addition to using the result of an averaged frame. The red arrows simply show the motion vectors for the target.

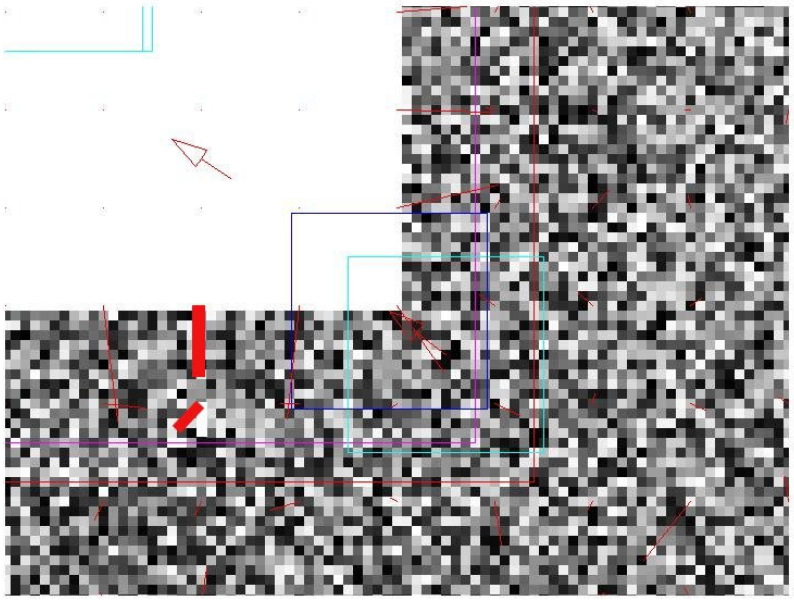

Figure 4.2. Corner movement of box test.

A test case that shows the gradient for the edges and corners can be seen more clearly in figure 4.3. The corners show the largest gradients and the gradients are zero at every position that does not have an edge or corner. The smaller boxes represent subtargets while the larger box represents the target as a whole.

This verifies the performance of theLucas Kanade tracker against simple test cases, so the next step is to see how it performed against a more complex test case.

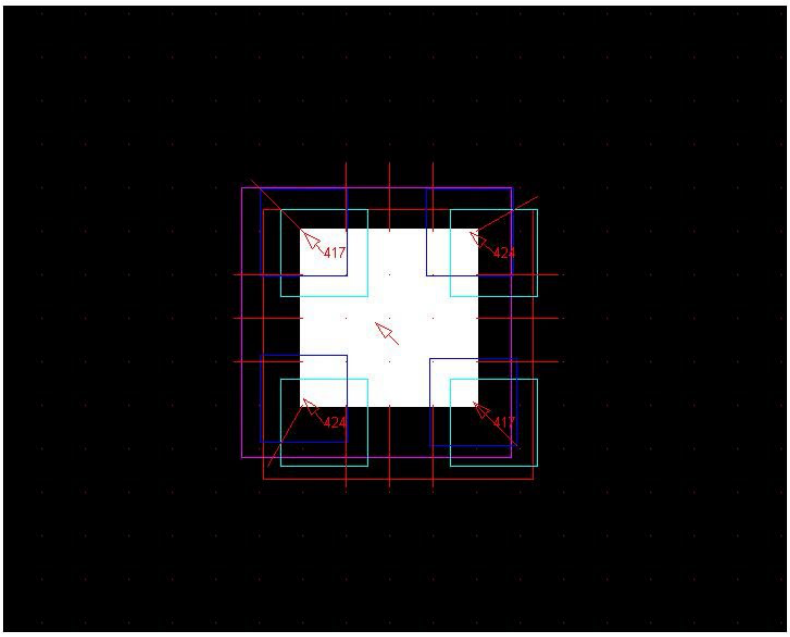

Figure 4.3. Box test on black background.

The test case in figure 4.4 shows the tracker following a bus around a traffic circle. This is a case that includes affine transforms in addition to occlusions. Although some part of the bus is always within the frame, the front of the bus is self occluded in the second half of the frame sequence. This particular test case revealed two major issues that need to be resolved for any Lucas Kanade based implementation to work well. In the initial versions of the tracker, the tracker would lose track after about thirty frames. This was due to the fact that Lucas Kanade returns a floating point distance approximation at every iteration. At first, this value was rounded so that the discrete pixel values could be returned. But this throws off the feature tracking after a certain amount of time. The fix for this was to interpolate the pixels for the current feature position based on the iteration values from Lucas Kanade. In Matlab, this can be accomplished with the *interp*2 command. The next important issue deals

with achieving subpixel resolution between frames. Since the templates are updated every frame and objects rarely travel in entire pixel increments it is important to keep track of the exact position that the tracker returns. Otherwise template degradation occurs and the template will start sliding and eventually lead the target track off course without a major score change. This can be detected by constantly monitoring the feature scores every frame and performing resampling based on that, but this was easily solved by just using floating point values for the target positions and rounding to integers when presenting the results to the user.

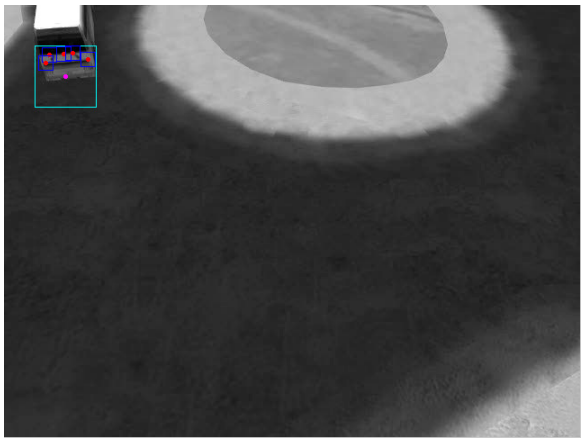

Figure 4.4. First bus frame.

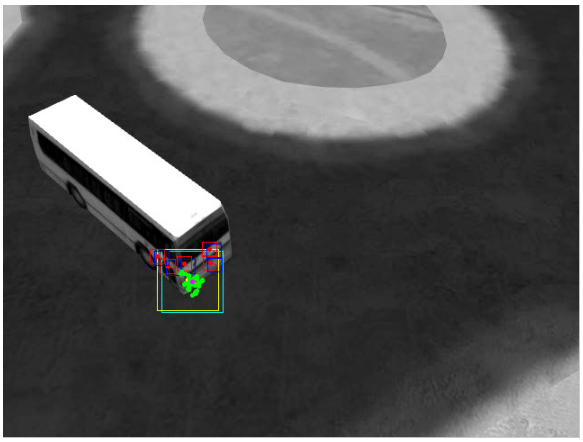

Figure 4.5. Bus frame at start of turn.

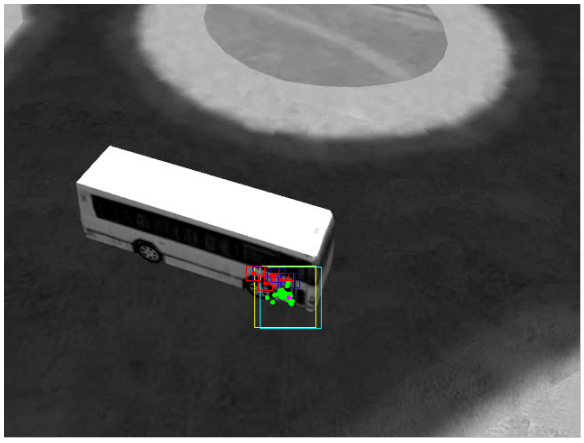

Figure 4.6. Bus test with occlusion.

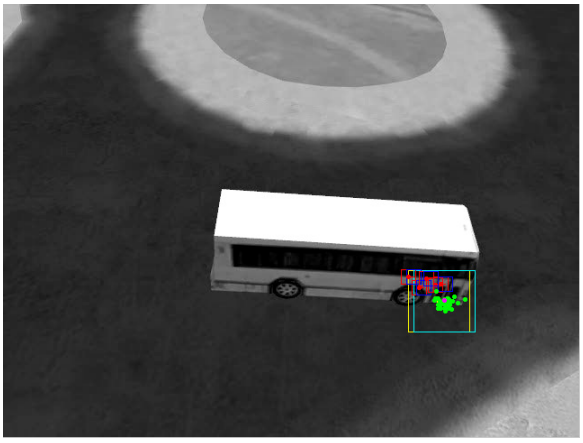

Figure 4.7. Bus turn complete.

The next frame sequence starting at figure 4.8 shows what happens when two aircraft cross paths when using the standalone Lucas Kanade algorithm. The tracker works well until the occlusion event occurs and the starts resampling targets and eventually finds the target going in the opposite direction. This would be undesirable behavior in a tracking scenario since it might be the job of certain pilots to track a certain target and in this case two pilots could end up tracking the same target while the other target is free to fly untracked. The tracker becomes lost because the two targets are extremely similar and the target being tracked is occluded so the tracking occurs on the wrong target. Since the template is updated every frame the tracker will move to the wrong target and will continue tracking it unless it becomes occluded by a similar feature.

Applying the particle filter shows a dramatic improvement as the tracker is able to maintain lock on the target throughout the occlusion event. This is shown in figure 4.12 where the aircraft starts out as before except for the addition of the green particle points. Once the occlusion event occurs, the targeting box disappears since that was maintained by the Lucas Kanade feature tracker. The red mark shows the highest weighted particle and would mark the position of the target to the user. Throughout the particle filter phase, the particles are able to use their motion model to estimate the position of the target. This, in addition to the sum of absolute differences based weighting scheme, leads to an accurate track. Once the occlusion event is finished, the particle filter scoring system is able to provide enough input to the phase transition scheme to switch back to Lucas Kanade tracking. This transition is based on the difference between the template snapshot that happened during the last transition with the current highest weighted particle snapshot.

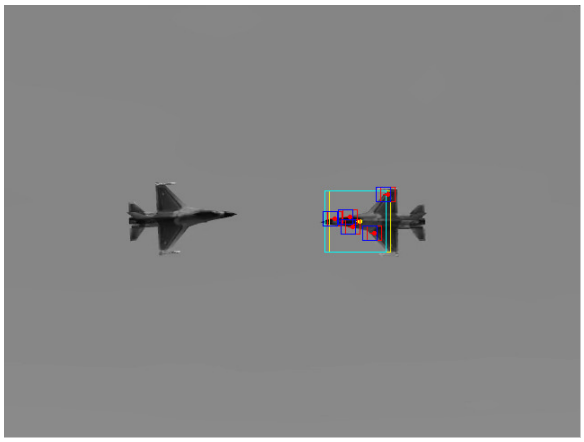

Figure 4.8. Aircraft before occlusion frame.

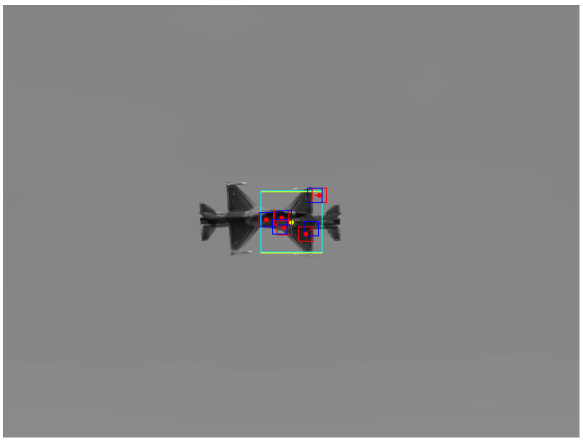

Figure 4.9. Aircraft start of occlusion event.

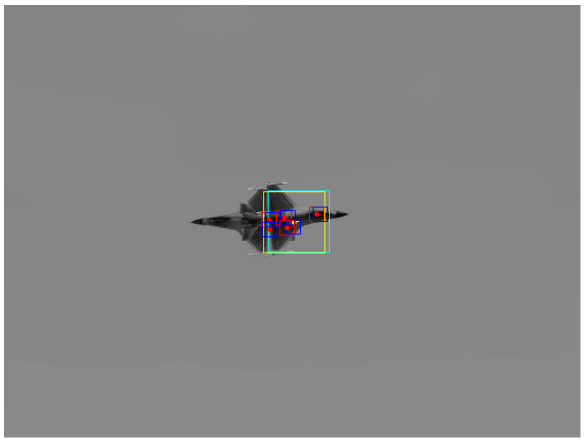

Figure 4.10. Tracker picking up alternate aircraft initially.

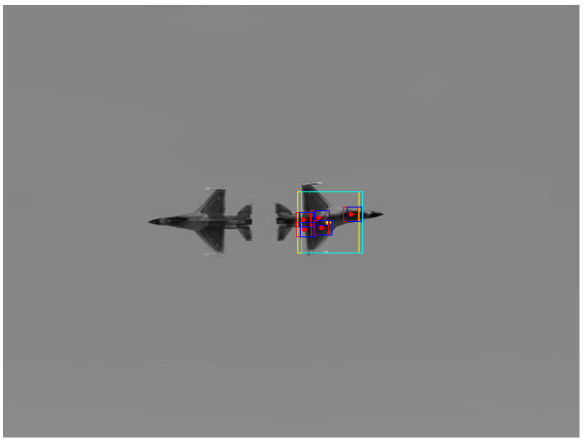

Figure 4.11. Tracker acquisition on 2nd aircraft complete.

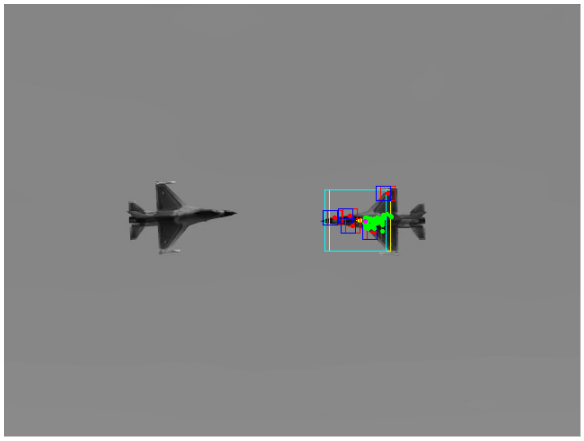

Figure 4.12. Aircraft acquisition.

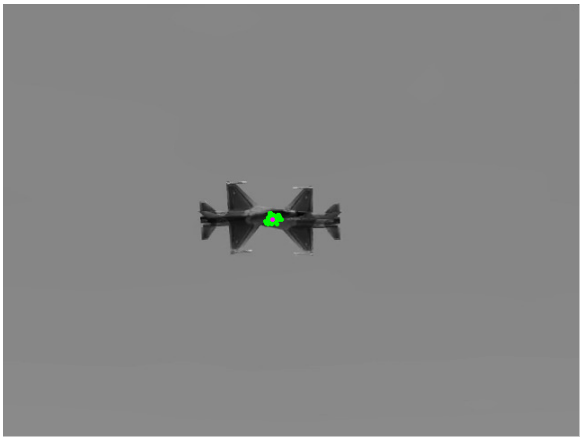

Figure 4.13. Beginning of occlusion event.

Figure 4.14. Particle filter tracking.

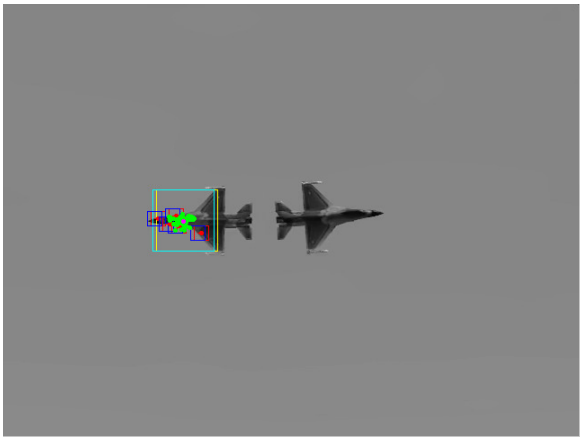

Figure 4.15. Lucas Kanade reacquisition.

4.5 Conclusions

This algorithm shows some promise for sensor pod tracking since it is able to maintain track through occlusion and affine transforms. Although it is far from perfect, there are still a variety of improvements that can be made to the algorithm so that it can meet application specific requirements. The Matlab implementation of the Lucas Kanade and particle filter hybrid can run at roughly 5 frames per second on a 1GHz machine so the implementation is not suitable for real time yet. But, the code is running in Matlab and is not optimized to run in real time as the code was designed to be flexible enough to add a variety of algorithms into the tracker for rapid prototyping purposes. However, there is evidence that suggests the Lucas Kanade feature tracker can easily work in real-time as OpenCV currently has a C implementation of the algorithm that is able to track faces in real-time. The addition of the scoring and particle filter logic will not affect the timing of the algorithm enough to prevent the system from running in real-time if it is designed with an eye for efficiency.

The tracker, as it is implemented now, tends to perform better when used on larger targets. However, in some cases this is not practical in the real world as sensor pods are often tracking multiple targets at large distances. But, when the sensor provides enough target surface area, the algorithm can perform quite well. It is therefore advisable to use another algorithm for small targets in addition to this one which will perform well when tracking larger targets.

The tracker makes use of a variety of popular algorithms that have been used to track targets in the past including Lucas Kanade, particle filters, and image processing techniques. It also uses a variety of subtargets and manages them in such a way to prevent rogue subtargets from causing loss of track.

CHAPTER 5

FUTURE IMPROVEMENTS

The hybrid approach used for the target tracker worked well for the test cases, but there are some improvements that can be made to it in the future. The first item to improve would be the run time performance since the current algorithm runs at 5 frames per second. A C implementation would be recommended, but the Matlab code could likely be tweaked enough so that it would give decent performance. A tracking performance tweak that might be interesting to explore would be to add a mean shift tracker to the framework. This would likely yield better affine transformation performance at the expense of picking up background clutter more often.

Another tweak to consider would be to add a particle system to every feature or subtarget. However, this gets more difficult as particles from different features would often find themselves in the same area of interest and would lead to a more complex algorithm. Although the current implementation has some built in features to improve affine transformation tracking, the targeting box is always a constant size. This could be improved by just using the outer subtargets as the boundary of the target. But, the algorithm would need to be careful in order to assure that there are not rogue subtargets that are making the targeting area too large.

The current implementation does not use a pyramidal hierarchy approach for the Lucas Kanade tracker [25]. Using a pyramidal scheme allows the tracker to pick up larger target motions. This would prove to be beneficial for tracking fast targets or

when using narrow field of view sensors. The state change logic is highly application specific under the algorithm described in this paper as the state change thresholds need to be tweaked quite often. However, this can be improved by using a variety of techniques. One possible method for detecting an occlusion event would be to count the number of feature resamples over subsequent frames and use that to form a score that could be thresholded. Using a sequence of frames would have the benefit of averaging noisy results so that the algorithm does not switch modes too quickly. This could also be used with the sum of absolute differences averaged over subsequent frames.

APPENDIX A

ADDITIONAL FRAME CAPTURES

A.1 Humvee Tracking

This section consists of additional snapshots for the Lucas Kanade particle filter tracker. The figures starting at A.1 show the tracker working on a humvee in a pseudo sine wave pattern which exercises some of the affine invariance properties of the algorithm.

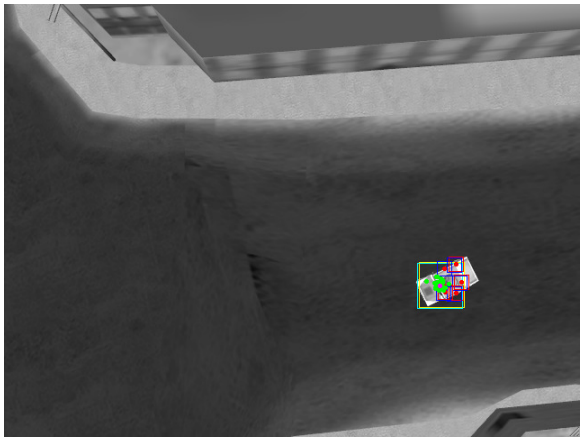

Figure A.1. Humvee sine pattern α_1.

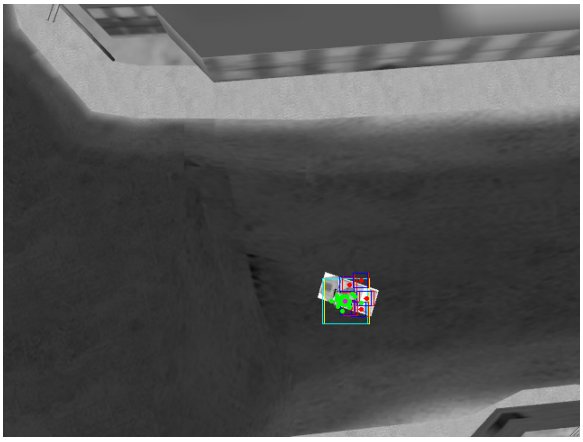

Figure A.2. Humvee sine pattern α_2.

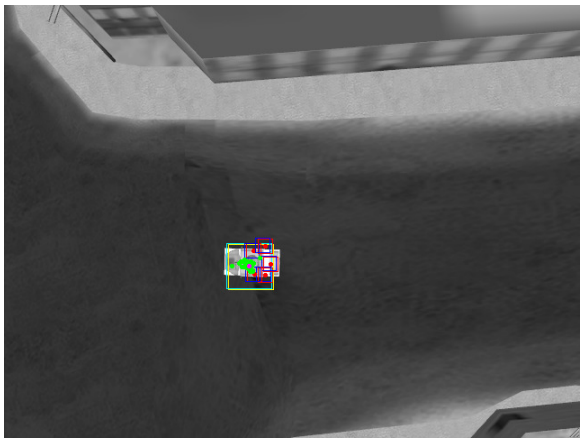

Figure A.3. Humvee sine pattern α_3.

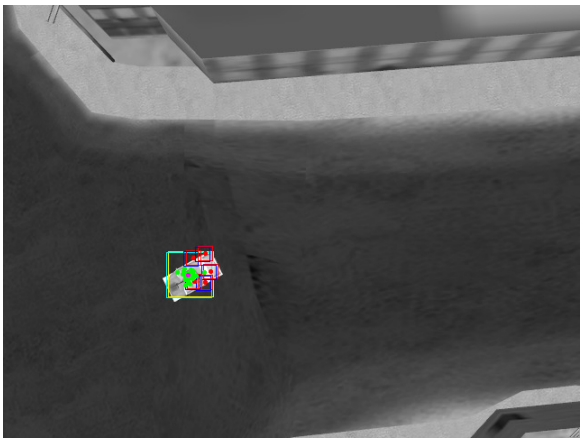

Figure A.4. Humvee sine pattern α_4.

A.2 Jeep Tracking

The next set of frames starting at figure A.5, show the tracker working through an occlusion with another vehicle.

Figure A.5. Jeep acquisition.

Figure A.6. Jeep point of occlusion.

Figure A.7. Occulted jeep.

Figure A.8. LK tracker reacquire.

A.3 Track During Orbit

The following figures show the hybrid tracker working throughout an air to ground orbit sequence. Orbiting a target is especially useful for recon operations from mobile air platforms.

Figure A.9. Orbit acquisition.

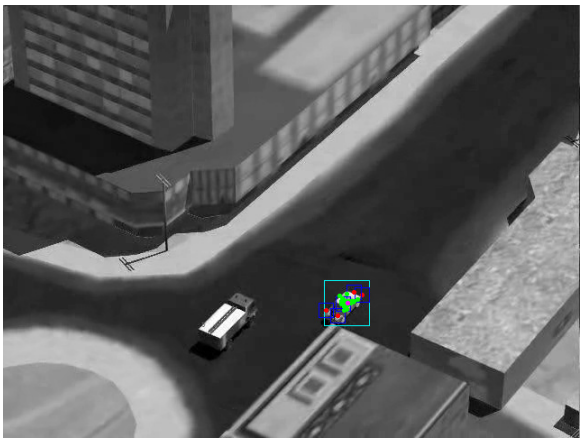

Figure A.10. Orbit frame α_2.

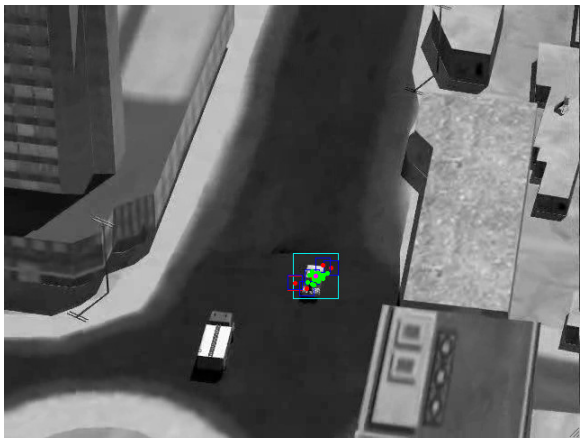

Figure A.11. Orbit frame α_3.

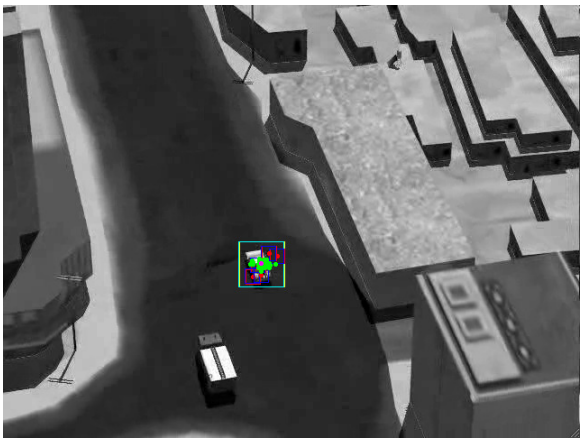

Figure A.12. Orbit frame α_4.

A.4 F16 Clouds with Occlusion

The next sequence of frames shows an F16 occlusion scenario with the camera viewpoint moving with a positive pitch rate with a cloudy background.

Figure A.13. F16 acquisition.

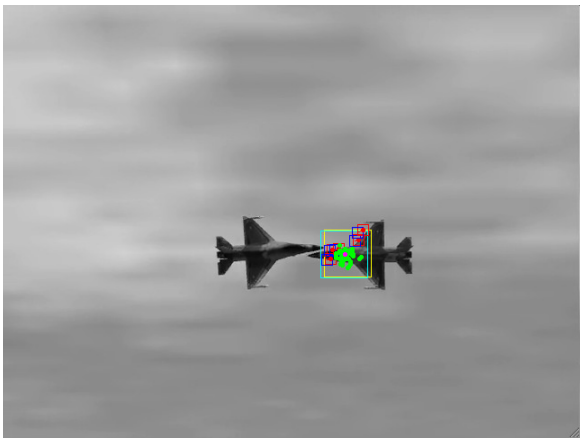

Figure A.14. Point of occlusion.

Figure A.15. F16 occulted.

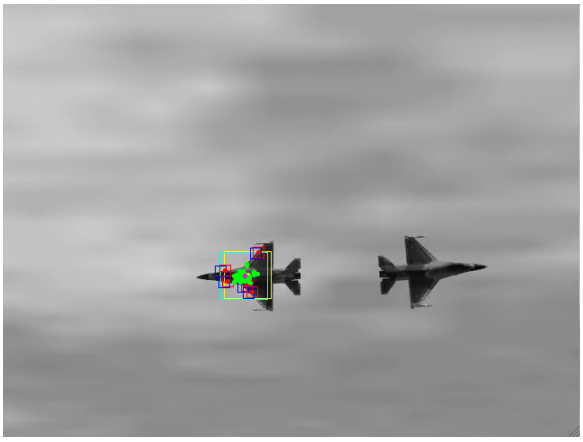

Figure A.16. Reacquisition.

A.5 F16 Clouds with Occlusion Minus Particle Filter

The next sequence of frames shows an F16 occlusion scenario with the camera viewpoint moving with a positive pitch rate with a cloudy background without particle filters. The system picks up the incorrect aircraft after the occlusion event.

Figure A.17. F16 acquisition.

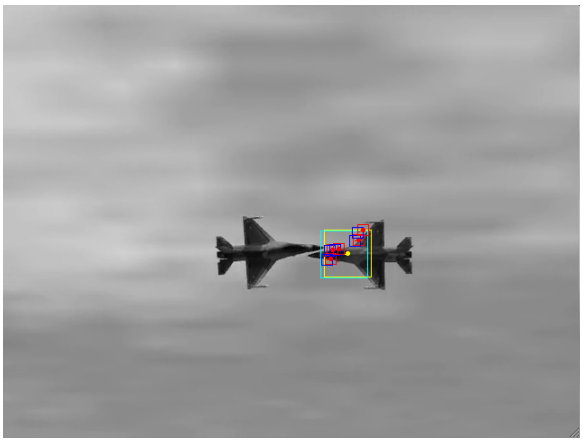

Figure A.18. Point of occlusion.

Figure A.19. F16 occulted.

Figure A.20. Reacquisition.

APPENDIX B

ACRONYMS

- EO: (Electro-optical) A sensor that picks up visible spectral bands and emits grayscale frames
- FLIR: (Forward Looking Infrared) An infrared based sensor
- IR: (Infrared) A spectral band used in aircraft sensors
- LK: (Lucas Kanade) In this context, an algorithm for determining optical flow
- MS: (Mean Shift) A vision algorithm that is based on shifting a color histogram to find targets
- NVG: (Night Vision Goggles) Goggles that pick up the near infrared spectral band leading to better nightvision performance
- PDF: (Probability Density Function) A function that describes the probabilistic behavior of a system.
- PF: (Particle Filter) Used as a kinematic filter for occlusion tracking

REFERENCES

[1] G. Pulford, "Taxonomy of multiple target tracking methods," Hampshire, UK, Aug. 2004.

[2] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," Intel Corporation, Tech. Rep., 2000.

[3] S.-M. Lee, *Spline Curve Matching With Sparse Knot Sets*. Seoul, Republic of Korea: Asian Federation of Computer Vision Societies, 2004.

[4] M. A. Isard, "Visual motion analysis by probabilistic propagation of conditional density," Ph.D. dissertation, University of Oxford, September 1998.

[5] D. W. Wagener and B. Herbst, "Face tracking: An implementation of the kanade-lucas-tomasi tracking algorithm," South Africa.

[6] "An improved method for tracking a single target in variable cluttered environments," in *IEEE International Symposium on Signal Processing and Information Technology*, Dec. 2008.

[7] F. Wang, E. Liu, and J. Y. R. Liu, "Target tracking based on a hybrid tracker with two hierarchical appearance models," *IOPscience*, pp. 2546–2554, 2007.

[8] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips, *Introduction to Computer Graphics*. Addison-Wesley Professional, 1993.

[9] "Real-time tracking of non-rigid objects using mean shift," in *IEEE CVPR*, 2000.

[10] "A method for flir target tracking based on distance updating," in *Congress on Image and Signal Processing*, Beijing, China, 2008.

[11] C. Zhao, A. Knight, and I. Reid, "Target tracking using mean-shift and affine structure," in *International Conference on Pattern Recognition*, 2008.

[12] S. Wong, "Advanced correlation tracking of objects in cluttered imagery," in *Proc. SPIE: Acquisition, Tracking and Pointing XIX*, 2005.

[13] A. Verri, *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, Mar. 1998.

[14] J. Aranda, J. Climent, and A. Grau, "A fpga implementation of video rate multi-target tracking system," in *Proc. of 24th Euromicro Conference*, Vasteras, Sweden, Aug. 1998.

[15] M.-C. Roh, T.-Y. Kim, J. Park, and S.-W. Lee, "Accurate object contour tracking based on boundary edge detection," *Pattern Recognition*, pp. 931–943, 2007.

[16] N. Gordon, "Beyond the kalman filter: Particle filters for tracking applications," Defence Science and Technology Organisation, Australia, Tech. Rep., Oct. 2003.

[17] C. Chang, R. Ansari, and A. Khokhar, "Multiple object tracking with kernel particle filter," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Chicago, IL, 2005.

[18] Y.-F. Ma and H.-J. Zhang, "Detecting motion object by spatio-temporal entropy," Microsoft Research, Beijing, China, Tech. Rep.

[19] G. Zhu and K. Wu, "A automatic accurate target tracking algorithm based on the infrared image," in *Proc. International Workshop on Information Security and Application*, Nov. 2009.

[20] G. Welch and G. Bishop, "An introduction to the kalman filter," University of North Carolina at Chapel Hill, Chapel Hill, NC, Tech. Rep., Mar. 2002.

[21] ——, "An introduction to the kalman filter," in *SIGGRAPH*, Chapel Hill, NC, Aug. 2001, pp. 1–47.

[22] S. J. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," in *Proc. of the IEEE*, Mar. 2004, pp. 401–422.

[23] G. A. Terenjanu, "Unscented kalman filter tutorial," University at Buffalo, Buffalo, NY, Tech. Rep.

[24] L. G. Shapiro and G. C. Stockman, *Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 2001.

[25] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker description of the algorithm," Intel Corporation, Tech. Rep., 2000.

[26] S. Baker, R. Gross, T. Ishikawa, and I. Matthews, "Lucas-kanade 20 years on: A unifying framework: Part 2," CMU-RI-TR-03-01, Tech. Rep.

[27] P. Torma and C. Szepesvari, "Combining local search, neural networks and particle filters to achieve fast and reliable contour tracking," Mindmaker, Budapest, HU, Tech. Rep.

[28] C. Yang, R. Duraiswami, and L. Davis, University of Maryland, College Park, MD, Tech. Rep.

[29] Y. Rui and Y. Chen, "Better proposal distributions: Object tracking using unscented particle filter," Microsoft Research, Redmond, WA, Tech. Rep.

[30] I. M. Rekleitis, "A particle filter tutorial for mobile robot localization," McGill University, Montreal, Quebec, Tech. Rep.

[31] H. Alt and L. J. Guibas, "Discrete geometric shapes: Matching, interpolation, and approximation," Stanford University, Stanford, CA, Tech. Rep., Dec. 1996.

[32] D. Moore, "A real-world system for human motion detection and tracking," California Institute of Technology, Pasadena, CA, Tech. Rep., June 2003.

[33] Z. Zivkovic and F. van der Heijeden, "Better features to track by estimating the trackign convergence region," University of Twente, Netherlands, Tech. Rep.

[34] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 564–577, May 2003.

[35] I. Leichter, M. Lindenbaum, and E. Rivlin, "Tracking by affine kernel transformations using color and boundary cues," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 164–171, Jan. 2009.

[36] G. Fan, V. Venkataraman, L. Tang, and J. Havlicek, "A comparative study of boosted and adaptive particle filters for affine-invariant target detection and tracking," in *Proc. of Conference on Computer Vision and Pattern Recognition Workshop*, 2006.

[37] D. Musicki, B. F. L. Scala, and R. J. Evans, "Integrated track splitting filter: Efficient multi-scan single target tracking in clutter," Australia.

[38] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. of Imaging Understanding Workshop*, Pennsylvania, 1981, pp. 121–130.

[39] "Matching and recognition of shapes using chord-based point density graphs," New Brunswick, NJ, USA.

[40] "Study of multi-target tracking and data association based on sequential monte carlo algorithm," in *International Seminar on Future BioMedical Information Engineering*, 2008.

[41] H. Ryu, "Multiple object tracking using particle filters," Master's thesis, Aug. 2006.

[42] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River, NJ: Prentice Hall, 2008.

BIOGRAPHICAL STATEMENT

Justin G. Graham was born in Wharton, TX in 1980. He received his B.S. degree from Texas A&M University, College Station, TX in 2002. From 2003 to 2008, he worked for Lockheed Martin in Fort Worth, TX as a Software Engineer designing and implementing aircraft simulators. From 2006 to 2008, he participated in the Engineering Leadership Development Program sponsored by Lockheed Martin in Orlando, FL. During that time he contributed to identification and security products for Transportation and Security Solutions under Lockheed Martin as well as developing logistics systems for their Simulation, Training and Support division. In 2008, he began working with L-3 Communications as a Software Engineer primarily focused on image generation for military aircraft simulators. His current research interest is primarily aimed at investigating sensor pod target tracking simulation for military aircraft.