Open access • Proceedings Article • DOI:10.2514/6.2019-0387

Multiple Target Tracking Via Dynamic Point Clustering on a UAV Platform

Seth D. Holsinger, Rajnikant Sharma

Institutions: University of Cincinnati

Published on: 07 Jan 2019

Topics: Cluster analysis

University of Cincinnati

Date: 11/18/2018

I, Seth D Holsinger, hereby submit this original work as part of the requirements for the degree of Master of Science in Aerospace Engineering.

It is entitled:

Multiple Target Tracking Via Dynamic Point Clustering on a UAV Platform

Student's name: **Seth D Holsinger**

This work and its defense approved by:

Committee chair: Rajnikant Sharma, Ph.D.

Committee member: Manish Kumar, Ph.D.

Committee member: George T. Black, M.S.

31828

Multiple Target Tracking via Dynamic Point Clustering on a UAV Platform

A thesis submitted to the

Graduate School

of the University of Cincinnati

in partial fulfillment of the

requirements for the degree of

Master of Science

in Aerospace Engineering and Engineering Mechanics

by

Seth Holsinger

B.S. University of Cincinnati

May 2017

Committee Chair: Rajnikant Sharma, Ph.D.

Abstract

This research focuses on solving the problem of tracking multiple ground targets from a single or multiple UAV's utilizing a dynamic clustering algorithm. Each UAV is assumed to have only one camera that is mounted on a two axis gimbal. The location of each of the targets can be ascertained by using a geolocation method with the camera as the primary sensor. The targets will not be able to transmit any of their location data to the UAV's. An extended Kalman filter is used to estimate the location and velocity of the targets with the afore mentioned geolocation used as the update step. The algorithm uses the estimated target locations and the distance from the UAV to different point locations to find the maximum number of targets that the camera can view at once. These clusters of points are then subject to a cost function where the goal is to minimize the overall uncertainty of all of the system. While also staying below a maximum allowable uncertainty for individual targets. The approach taken by this method is scalable with the number of UAV's and ground targets that it can track to allow for it to be used in a wide variety of circumstances. The targets future locations can also be evaluated by the clustering algorithm to get the locations that the camera should point in order to minimize the uncertainty.

Acknowledgements

I would like to thank my advisor Dr. Rajnikant Sharma for his guidance through the research process. I would also like to thank the students whom I worked with at RISC lab for their advice. Finally I would like to thank my family for their unending support while I was working on this thesis.

Contents

List of Figures

Symbols

p^i_{obj} position of the object to be tracked in the inertial frame

p^i_{UAV} position of the UAV in the inertial frame

R^i_b rotation matrix from body frame to inertial frame

R^b_g rotation matrix from gimbal frame to body frame

R^c_g rotation matrix from camera frame to gimbal frame

l^c the vector representing the distance from the UAV to the target in camera frame

$\check{l}$ the unit vector from the UAV to the target

L the distance between the UAV and the target

h the altitude of the UAV

φ angle between the h and $\check{l}$

k^i unit vector representing the k axis

Chapter 1

Introduction

Object tracking is a challenging problem and one that has been the focus of researchers for years. Particularly the problem of doing so using aircraft to track objects on the ground. Aircraft provide a number of advantages namely speed, the ability to move over obstacles and having a height advantage over ground based tracking methods. There are many different methods that are able to provide location estimates of objects on the ground. However, it is still quite difficult for a single aircraft to effectively track multiple ground objects. In addition these tracking missions also require human operators. Automating these missions, another feature of this research, will free up valuable time for other required tasks.

Primarily, the research being presented in this paper aims to provide a flexible solution to the problem of tracking multiple ground objects with limited aerial resources. As such

the scenarios being evaluated focus on situations where the number of objects that require tracking out number the UAVs intended to track them. Parts of this method namely the approach to sensor fusion is agnostic to the sensors used. While other parts such as the algorithm to determine how many objects can fit within the cameras field of view are not. Though this could potentially be replaced with some forms of radar with some required modifications.

Cooperation between the aircraft is the best way to effectively track ground objects. This is especially true when the number of objects outnumbers the number of UAVs. Even with cooperation, if resources are not managed correctly the tracking process will begin to break down. Once an object has been lost it can become extremely challenging to find it again. Due to the importance of resource allocation part of the aim of this research was to have the UAVs cooperate with one another as much as possible.

1.1 Related Work

Another solution to this particular problem was proposed in [2]. However, the method in this paper differs in the type of approach that was taken. In [2] the approach uses geolocatization in combination with an EKF (extended Kalman filter). The algorithm that decides which target to observe decides based off of a optimization function that determines which targets will lower the total covariance taken from the EKFs for each target. While the approach taken in this paper, while also using EKFs and geolocalization,

decides which target to look at by calculating which location to point that has the greatest number of targets as well as what will lower the total system covariance the most. The key difference between the two methods being that while [2] considers only a single target at a time. The method purposed in this paper considers groups of targets. In addition this research expands the scope of the problem to include multiple UAVs. One of the goals was to create an approach which is able to be scaled up or down depending on the number of targets and UAVs with cooperation in mind.

The sensors used in the method purposed by this paper to estimate the position of the ground targets are the GPS (Global Positioning System) aboard the aircraft, a barometer to improve the accuracy of the measured height of the aircraft, a gyroscope on the aircraft as well as on a gimbaled camera, and a camera. This differs from the sensors used in [1] which uses a ground sensing radar to detect objects. The aircraft camera sensors are used with the other aircraft sensors in a geolocalization EKF defined in [4]. The target estimate from the geolocation kalman filter is then fed into another EKF that tracks position and velocity. This is done to ensure that if two aircraft happen to have overlapping camera images both estimates can be used to get the targets' positions.

The processes that make up the larger algorithm fall into four main sections. The first is assigning targets to each aircraft. The next is determining the best place for each aircraft to point its cameras. Target estimatation is done by using the geolocalization EKF in [4] to estimate the location of each of the ground targets. Finally the estimate from the

geolocalization method are fused into a single estimate using another EKF. There is a substantial number of gains that require tuning in each of the EKFs. This tuning is done through the use of genetic algorithms. These processes were simulated using Matlab and Simulink.In addition parts of the processes were implemented on actual hardware. The hardware test focused on the control of a fixed wing aircraft and online geolocalization.

Chapter 2

Methodology

2.1 Geolocalization

The process that this paper uses to estimate the position of the ground objects is a combination of two methods: geolocalization using a camera and two extended Kalman filters. Geolocalization is a process which calculates the position of the ground targets through the use of a gimbaled camera carried by a UAV. However, geolocalization alone is not enough when the number of objects that require tracking exceed the number of UAVs available. If only geolocalization were to be used during the periods of time in which the camera is not observing an object the position estimate would not change. This becomes an issue if the object is moving as it could move far enough away from its initial position so that when the camera tries to point the last known location of the object it will no longer be there. Due to the autonomous nature of the aircraft it will then either not be

able to track the object or will have to spend valuable time trying to locate the object again. Depending on the type of object recognition used it may also be very difficult if not impossible to determine if the newly found object is even the object that is suppose to be tracked.

This necessitates the use of some sort of method to estimate the position of the object even when it is not being observed by a camera. The method chosen to do this estimation is an extended Kalman filter or EKF. Both the geolocalization equations and the EKF used in this research were taken from [4], but this form of geolocalization was first proposed in [3]. The equation for geolocalization is as follows:

$$p^i_{obj} = p^i_{UAV} + R^i_b R^b_g R^c_g l^c \tag{2.1}$$

This equation is not yet useful as l^c is not known. Thus using the following relations we can derive an equation of variables that are known:

$$\check{l} = \frac{l}{L} \tag{2.2}$$

$$L = \frac{h}{cos(\varphi)} \tag{2.3}$$

$$cos(\varphi) = \frac{h}{k^i \cdot R_b^i R_g^b R_g^c \check{l^c}} \tag{2.4}$$

Plugging all of these equations back into the original equation and one gets:

$$p_{obj}^i = p_{UAV}^i + h\frac{R_b^i R_g^b R_g^c \check{l^c}}{k^i \cdot R_b^i R_g^b R_g^c \check{l^c}} \tag{2.5}$$

The φ is the angle between the k axis and the $l$vector. To simplify this calculation it is assumed that the UAVs are close enough to the earth that the surface of the earth can be assumed to be flat. This is illustrated in Figure 3.1. More complex equations must be used if the scale of the simulated situations is increased. It should be noted that this is not a source of error within the simulated environment as all of the objects travel on a flat plane. In a real world scenario this would introduce a small amount of error in the estimate of an objects position, but this would be small at the altitudes where object detection is possible without optical magnification or the use of extremely high camera resolutions.

As stated before the geolocalization equation by itself is not enough for an object to be effectively tracked. So the equations need to be used to create a EKF. The derivation of

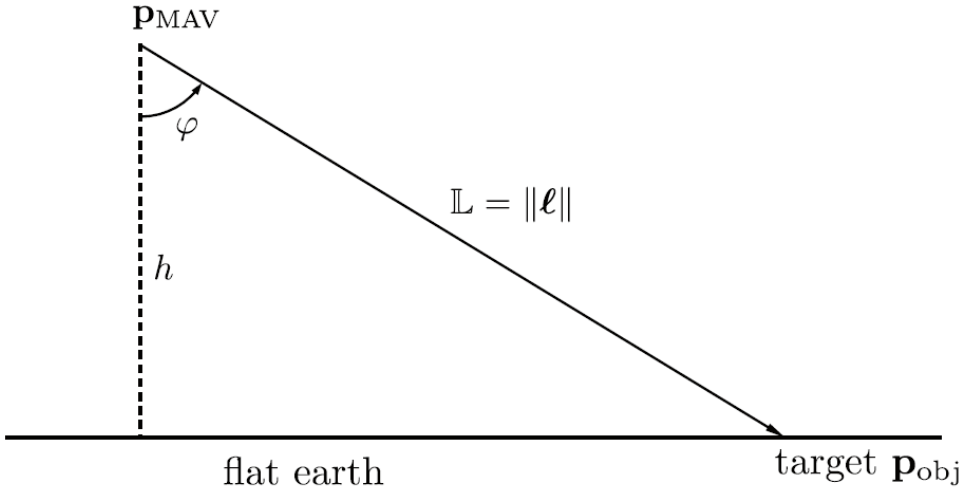

FIGURE 2.1: Flat Earth Assumption
Shows the effect of the flat earth assumption on the calculation of φ. Image taken from [4], page 232, Figure 13.2

the equations used in the EKF has also been taken from [4] chapter 13. The estimated states are P^i_{obj}, L, and V^i_{obj}. This differs slightly with [4] as this does not assume the object to be stationary and thus $\dot{P^i_{obj}}$ does not equal to zero in the equations of motion. Instead, $\dot{P^i_{obj}}$ is equal to V^i_{obj} and $\dot{V^i_{obj}}$ is equal to zero. The equations for $\dot{L}$ is given as the following:

$$\dot{L} = \frac{(P^i_{obj} - P^i_{UAV})^T(\dot{P^i_{obj}} - \dot{P^i_{UAV}})}{L} \tag{2.6}$$

Where $\dot{P^i_{UAV}}$ is equal to:

$$P_{UAV}^{i\dot{}} = \begin{pmatrix} \hat{V}gcos(\hat{\chi}) \\ \hat{V}gsin(\hat{\chi}) \\ 0 \end{pmatrix} \tag{2.7}$$

Together these equations form the prediction equations which can be summarized as follows:

$$f = \begin{pmatrix} Vn \\ Ve \\ 0 \\ 0 \\ \frac{(P_{obj}^{i} - P_{UAV}^{i})^T (P_{obj}^{i\dot{}} - P_{UAV}^{i\dot{}})}{L} \end{pmatrix} \tag{2.8}$$

The UAVs GPS is the only sensor that directly measures position. As such it makes sense for the position of the UAV to be the update equation. So the estimate of L is taken from the estimation and is used in conjunction with the estimate of the ground object's position to form the measurement equation. Taken from chapter 13 of [4] the measurement equations can be written in the form:

$$h = P^i_{UAV} = P^i_{obj} - L(R^i_b R^b_b R^g_c \check{l}^c) \tag{2.9}$$

With all of these equations the EKF can be created using the standard algorithm. The process of which can be found in chapter 8, on page 156 of [4].

2.2 Sensor Fusion

An important aspect of UAVs operating together to track multiple ground objects allowing for the estimates of one UAV to help the other UAVs estimates. This is done so that there is a unified estimate of the position of each of the ground targets and not a unique estimate for each aircraft. In addition the unified estimate should benefit from having the information from multiple UAVs rather than a single UAV. Many processes where attempted to try and unify the geolocation EKFs, but it proved quite difficult to find a method that worked. Two successful approaches where created. The first method relied upon meshing the covariance matrices and position estimates of each aircraft together to come up with a unified position estimate that would be common accross all of the UAV geolocalization filters. In practice there are many problems that are associated with trying to mesh the Kalman filters together to get a unified estimate of targets' positions. The first of these is that one of the values that the extended Kalman filters are estimating is the distance between the camera/UAV and each ground target, L. This is different

for each UAV as they are not located in the same position. However, the other four quantities being estimated for each target: north position, east position, north velocity, and east velocity are common between the UAVs. This means that four of the estimated values can be mixed but one cannot. The covariance of the Kalman filters cannot be mixed without care as it can greatly affect the estimates if not mixed properly. For these reasons the sensor fusion uses the following methods to make sure proper estimates are performed. Each UAV runs a Kalman filter to estimate the position of each ground target. The UAVs also keeps track of which of the targets have received updates. Whenever a UAV that was not estimating a target starts to try and estimate the position of that target it receives the covariance of the Kalman filter with the current lowest covariance for a particular target. However, all values but the diagonal set to zeros and the term on the diagonal associated with the length between the UAV and the target is reset to a standardized value. The other terms on the diagonal also receive a small increase to their uncertainty. The following equations are used:

$$P_{ij} = diag(P_{ij}) + 5. * I_{5x5} \tag{2.10}$$

$$P_{3,3,ij} = P_{3,3,ij} + 15 \tag{2.11}$$

Where P_{ij} represents the 5 by 5 covariance matrix for the extended Kalman filter. The i and j in the subscript of the P represent which Kalman filter is being accessed. In this

case i is for the number of targets and j is for the number of UAV's. I_{5x5} represents a 5 by 5 identity matrix that is used to clear the non diagonal terms as well as add a small amount to each. $P_{3,3,ij}$ represents the element in the covariance matrix that is located in the third row and column.

This is done before the update step is run, after the estimation and update steps are run the estimated positions and velocities are combined based off of a different decision making structure. This structure is roughly as follows: if only one UAV has done an update step the positions of all other UAV's Kalman filters are given its position and velocity estimates, if multiple of the UAVs or all of the UAV's have done an update step then the estimate with the lowest covariance is given eighty percent of the values, and the others that updated are averaged and then this average is given the last twenty percent of the values. If none of the UAV's preformed an update step then the filter with the lowest covariance is given eighty percent of the estimated values and all of the others are averaged and given the remaining twenty percent. This is done for each of the ground targets.

This method proved to be effective, but it suffered from a few problems. Traditionally the more updates steps that are preformed for a particular EKF the better information that filter has on the location of the object. Using this method much of this information is lost as it must be removed to be compatible with the other filters. In addition the information that is passed along suffers as additional uncertainly is added once again to

improve compatibility between filters. As the uncertainly remains higher then it would have been on the original filter the estimates are never quite as accurate as they would be using a single unmodified geolocation EKF. Still the filter did prove relatively stable due to the resetting of the large parts of the covariance matrices. This prevented the covariance from going unstable again at the cost of accuracy.

Due to the drawbacks of the first method to mesh the geolocation estimates together it was clear that a better method had to be created if possible. In order to achieve a superior filter the parts that were weak in the last method had to be improved upon. As such the focus was on retaining information rather than removing it. The solution that presented itself was to use an additional EKF in addition to the geolocation EKFs. The goal of this new EKF is to take the update information from the each geolocation EKF as well as the estimates and fuse them together in the update step of the filter. Information from the geolocation EKFs is passed on to the fusing EKF through both the estimates and the covariance matrix. The measurement noise matrix R in the fusing EKF is made up of values from the geolocation covariance matrix. Essentially what is done is the third row and column of the geolocation covariance matrix is removed and this new four by four matrix becomes the measurement noise matrix. The reason the third row and column are removed is that they correspond to the estimate of the length between the UAV and the ground object L. The values of L is not taken into consideration in the estimation or update steps of the fusing EKF. The reasoning behind this decision is that L is different for each UAV ground target pair and thus cannot be fused together

to form a meaningful estimate. In addition the fused position and velocity estimates do not get fed back into the geolocalization filters immediately. This is done because it is possible that a particular geolocation filter is more accurate then the fusing filter. As such it is important not to disturb the filters. Though, if carefully managed it is possible to improve the time it takes geolocation EKFs that have not received any measurement updates in a while to converge to a more accurate value.

This new method proved to be more effective then the old method when it came to accuracy. It is not without its own drawbacks though. Particularly when it comes to the stability of the EKFs. It is very possible for them to go unstable and provide widely inaccurate estimations of the positions and velocities. Contributing to this problem is one particularity of how the update step is done in the fusing EKF. As there are multiple aircraft which are assigned different groups of targets and so an individual aircraft may go for a fairly significant time period without updating its geolocation EKF for a particular target(s). This means that the next time this particular aircraft looks at the target(s) it has not estimated in a while, the estimate can be extremely poor. This will cause the global estimate to become very poor very quickly and leads to instabilities in the EKFs. There are ways to mitigate this particular problem though they require some tuning. The method that was chosen used a more complex update condition. The condition was initially if a geolocation EKF is receiving updates then the fusing EKF will also run its update step. The updated condition is if a geolocation EKF is receiving updates and the trace of that geolocation EKF's covariance matrix is less then 280 it will run the fusing

EKF's update step. This works well for the most part and significantly better then the initial update condition but fails when objects are far away. Far away objects tend to have higher covariances along the axis from the target to the UAV. If a UAV is looking at a far away target often times it cannot pass updates to the fusion EKF due to this increased covariance. To mitigate these issues a new conditional update statement was required. One that took into account the fact that far away objects would have worse covariances. To do this the conditional was updated so that there is an an added 'or' statement that allows for updates to occur in the fusion EKF if more than six updates have occurred in the geolocation EKF. Additionally, code was added to reset the geolocation EKFs that have not been updated for an extended amount of time. This is to prevent very poor updates from occurring in the fusion EKF as well as further prevention of instability in the EKF estimates. The conditional statement that resets this process checks to make sure that the trace of the covariance matrix of the geolocation EKF is greater then 1600. As well as checking to see if the trace of the covariance matrix for the geolocation EKF is greater then 1.4 times the trace of the fusion EKF's covariance matrix. This ensures that the geolocation EKFs have as little good information as possible when replacing them. If the afore mentioned conditional statement is met then the estimates of the particular geolocation EKF that met the conditions will be replaced and its covariance matrix will be reset. For the estimates they are replaced with the estimates from the fusion EKF except for L as it is not estimated by the fusion matrix. Instead this value is taken from the distance between the target estimates of the fusion EKF and the position of the UAV associated with the geolocation EKF being modified. The new covariance matrix is

created taking the values along the fusion EKF's covariance matrix diagonal and placing them along the new geolocation EKFs covariance matrix diagonal. Except for position three, three. This position uses the existing value from the geolocation EKF covariance matrix. All of the other values in the matrix are set to zero.

The prediction equations for the fusion EFK are as follows:

$$f = \begin{pmatrix} Vn \\ Ve \\ 0 \\ 0 \end{pmatrix} \tag{2.12}$$

The measurement equations that were used in the fusion EKF are as follows:

$$h = \begin{pmatrix} Pn \\ Pe \\ Vn \\ Ve \end{pmatrix} \tag{2.13}$$

As previously mentioned Pn, Pe, Vn, and Ve are taken directly from geolocation EKF for the measurement step. Measurement noise for the fusion EKF, $R_(fuse)$, comes from the covariance of the geolocalization EKF, P_geo. The following equations show how the covariance values are passed from the geolocalization EKF to the fusion EKF:

$$P_{geo} = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} \\ P_{21} & P_{22} & P_{23} & P_{24} & P_{25} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} \\ P_{41} & P_{42} & P_{43} & P_{44} & P_{45} \\ P_{51} & P_{52} & P_{53} & P_{54} & P_{55} \end{pmatrix} \tag{2.14}$$

$$R_{fuse} = \begin{pmatrix} P_{11} & 0 & 0 & 0 \\ 0 & P_{22} & 0 & 0 \\ 0 & 0 & P_{33} & 0 \\ 0 & 0 & 0 & P_{44} \end{pmatrix} \tag{2.15}$$

During the resetting of a geolocalization EKF the following equation is used to find the new estimated distance value, L:

$$L = \sqrt{(\hat{Pn_{fuse}} - Pn_{GPS})^2 + (\hat{Pe_{fuse}} - Pe_{GPS})^2 + (0 - Pd_{GPS})^2} \tag{2.16}$$

2.3 Target Tracking Algorithm

The principles that the target tracking algorithm uses are quite straight forward but they do rely on several principles that are best understood by doing several thought experiments. The goal of this algorithm is simple: to keep the covariance of each ground target from the Kalman filter low to ensure the targets' positions are being estimated with a reasonable degree of accuracy. In order to achieve this goal the algorithm attempts to fit the largest number of ground targets in the field of view of its camera as possible at its given time and position. In order to understand the maximum number of points that can fit within the camera's field of view it is easier to look at a simpler case: what is the maximum number of points that can fit within a circle of a given radius. The solution to this problem is quite simple: two points exist within the circle if the distance between them is less than the diameter of the circle. Following this logic a cluster of points fits within the same circle if every point in the cluster is less than the diameter of the circle in distance away from every other point in the cluster. Now, one needs to find where to place the circle so that every point is within it. This is a much more difficult question as in practice there can be a very large number of solutions to this question. A reasonable approximation of where this point is can be made is that it is the point

gotten by averaging the two points that are farthest away from one another in the cluster of points. Of course the cameras in this method are projecting a view of the ground that is not a circle but some form of quadrilateral. However, the logic created for the circle seems to work well with the projected camera view as well. As such the algorithm boils down to simulating the projection of camera pointing at possible cluster centers and determining the number of points that lie within that simulated camera projected view. The points that are checked are each of the estimated locations of the ground targets as well as the points gotten by selecting two points and taking the average of these two points. This is more points than would need to be checked by the above logic but more points than necessary are checked due to the difference between the logic from the thought experiments being for a circle and the actual system using quadrilaterals. This process is illustrated in Fig. 2.2.

The process that has been described so far has only been for a single UAV, but this method allows for the use of multiple UAVs working together. As such the ground targets are divided between UAVs through the use of clustering via the K-means method which was originally proposed in [5]. However, the exact algorithm that was used is the standard 'kmeans' function provided in MATLAB. Where the number of clusters is equal to the number of UAV's. This does not always lead to an even distribution of points, but the points should be grouped by how close they are to one another. Which is more important for the camera pointing section of the algorithm. The UAV's are commanded to fly an orbit around the center of whichever cluster is closest to them with only one UAV to a

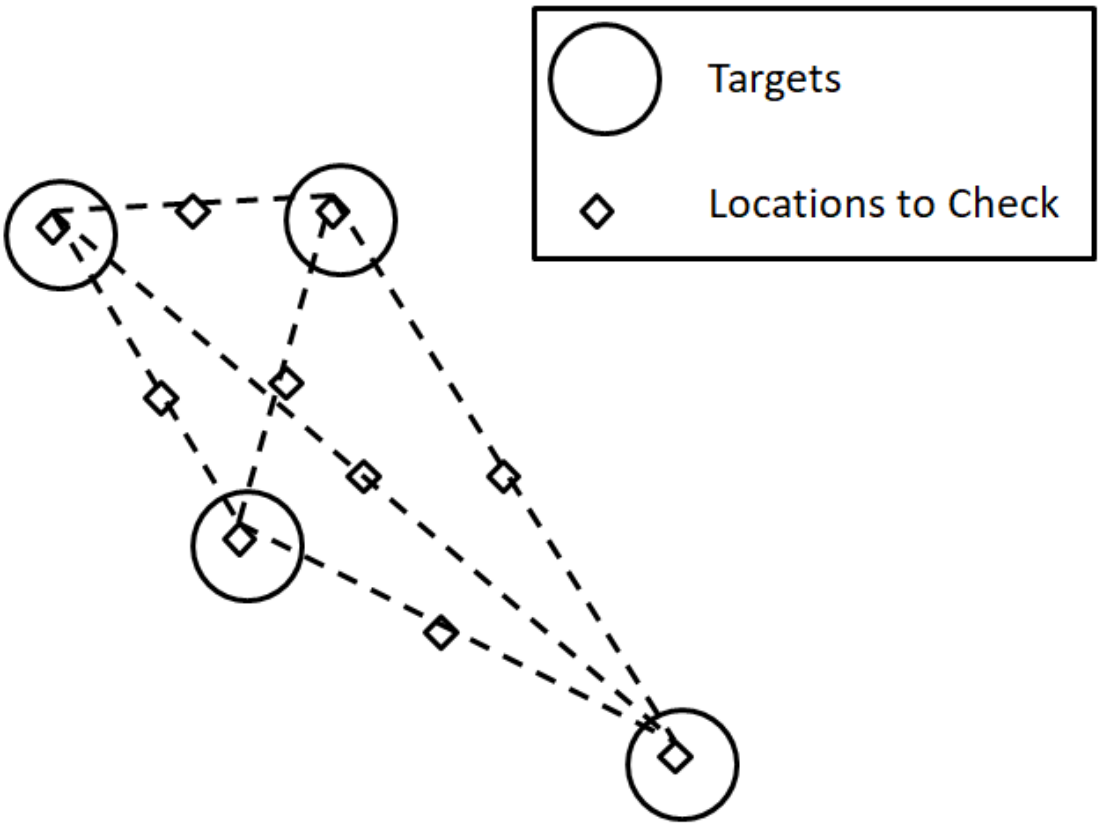

FIGURE 2.2: Depiction of Target Clustering
This diagram shows the points that are checked to find a place to point the camera.

cluster center.

	Target Cluster a	Target Cluster b	Target Cluster c	Target Cluster d
UAV 1	20	40	50	10
UAV 2	30	90	30	20
UAV 3	80	40	50	60
UAV 4	70	10	80	40

UAV 1	Target Cluster d	10
UAV 2	Target Cluster d	20
UAV 3	Target Cluster b	40
UAV 4	Target Cluster b	10

	Target Cluster a	Target Cluster b	Target Cluster c	Target Cluster d
UAV 1	9999	9999	9999	9999
UAV 2	30	90	30	9999
UAV 3	80	40	50	9999
UAV 4	70	10	80	9999

FIGURE 2.3: Target Cluster Assignment Algorithm

This diagram shows the process of assigning each cluster center to a UAV.

There were many changes that were made while codifying the afore mentioned processes. First of which is the process of the picking the point at which the camera will point. As stated above an excellent estimate of where to point to fit all of the objects in the camera frame is between the two objects that are farthest apart. This was changed to a process that has less computational intensity. Instead the point that was used in the cluster finding process that had all of the points with in itself is used. This can be calculated in conjunction with the cluster finding process and as such cuts down on computational

time. Though there was only one major change there were other non optimal/close to optimal decisions that were made. Namely in the processes of assigning targets to each UAV. As stated before this is done using 'kmeans' clustering. However, the decision of which UAV to assign to each cluster is a very difficult one. The optimal solution to this problem would be to minimize the total combined distance of each UAV to its assigned cluster. This can be found with a variety of the methods. For example the use of a branch search algorithm. Unfortunately while a branch search algorithm will always find the minimum distance it is not time efficient. The method used was to create a table that contains all of the distances between each target and UAV. This table is then evaluated row by row to find the values that are the smallest for each row. These values are then passed on as the clusters is given to each UAV. This works well when each UAV is close to only a single cluster, but this is not always the case. In cases where one UAV is the closest to multiple clusters it is assigned the cluster that is closest to itself. While this process is faster then branch search it does not usually result in the minimum overall distance. That being said while branch search might be better and even possible with small numbers of UAVs it is not great when scaling the number of UAVs up. As such the approximate methods is used as when using large numbers of UAVs it becomes much more efficient. This could change depending on hardware setup, but that will be covered more in the future work section. The process of the cluster assignment is illustrated in Fig. 2.3.

One potential flaw of this target tracking algorithm is the uncertainty in the position of

the objects at which the cameras are trying to point. Error in the position estimates means that the cameras on the UAVs could end up pointing at nothing. While features such as ignoring group pointing when the covariance of an objects goes beyond a certain point certainly helps to mitigate this problem it is sometimes not enough. The simulated objects move in a grid like motion and change directions at "intersections". This creates error when there are no cameras pointing at an object that changes its direction. The estimator will continue to estimate the objects position as if it was still moving in the same direction. While the algorithm does try to look at the objects it is suppose to be observing fairly often these errors even when fairly modest can prevent the grouping for camera points from working properly. The algorithm could try to point at the space where several targets are suppose to be but if they are towards the periphery of the cameras view the error may mean they are actualy not in view. Conversely, all of the targets may be closer together but if this happens then these objects will likely be in view anyway. Looking at a location where there are not objects is not acceptable and as such the projected view of the camera is artificially narrowed by lowering the field of view parameter when calculating the projected camera frame. Lowering the field of view makes it more certain that the objects to be tracked are within the camera's image frame. The downside is that when the estimates are good potential opportunities to view objects could be lost. Though this is not a particularly dreadful thing as objects near the edge of the image frame are more likely to leave it then objects closer to the center. Which makes a difference as the camera will hold its position for two seconds and as such objects near the edge could easily leave the image frame. The reasoning behind the two

second hold time is that the gimbal takes time to move the camera into position. If this is not done the camera could only end up looking at an objects for a fraction of a second.

2.3.1 Kalman Gain Tuning

Tuning the Kalman gain matrices (Q and R) was accomplished through the use of genetic algorithms wherever possible. For the geolocation EKF the measurement standard's of deviation were available as they are taken from the GPS standard of deviation. In this case they were inherited from the work done in[2]. These standards of deviation are squared and then placed along the diagonal of a three by three matrix to form the R matrix (measurement noise). The process noise, Q, was tuned by a genetic algorithm. All genetic algorithms posses the same basic structure [7]. This can be broken down into the following process: create an initial population of values, evaluate population of solutions, use a set of selection criteria to find the best individuals of a population, 'breed' the top preforming members of the population to form a new population, modify random members of this new population with random values, repeat steps two through five over and over again until an acceptable solution is found then stop the loop. The major differences in genetic algorithms is due to how they breed their populations and their optimization criteria. The criteria that was used in this genetic algorithm was the average total error taking into account object position in north and east. The population members that gave the lowest average error were chosen to be bred. The crossover function process that was chosen to be used followed these steps: sort the population with the lowest error going to the top of the list, the top two solutions are passed on

to the next generation unchanged, new solutions are bred from the top twenty solutions. The geolocation Kalman gains are the population members. They are arranged into a vector of values with a length of five (the length of the diagonal of the Q matrix). The breeding was done by taking the two randomly selected population members and selecting a random subsection of the gain vectors. Then either one of two things would happen. The gains in the subsection of each of the chosen population members would be either averaged together or they would be switched. This process can be seen in Fig 2.4. In [2] the process noise matrix Q was created using values from user experience. These values are much lower then the values chosen by the genetic algorithm. This is most likely due to several factors. Higher process noise means that the uncertainly goes up much faster and remains at higher values even while updates are occurring. When the levels of uncertainly are higher the update step matters more. The evaluation criteria used the error in the positions of the objects being tracked. The environment was one where updates are not always happening regularly and as a sharp change in the objects' motion happens regularly it makes sense that the estimation process was not trusted by the algorithm. If the evaluation criteria had been covariance the genetic algorithm would likely have selected lower process noise values.

2.4 Simulated Environment Details and Setup

The program used for the simulation of the UAV's and ground targets was MATLAB's Simulink [6] as shown in Fig 2.6. The version of MATLAB used was R2013a. The

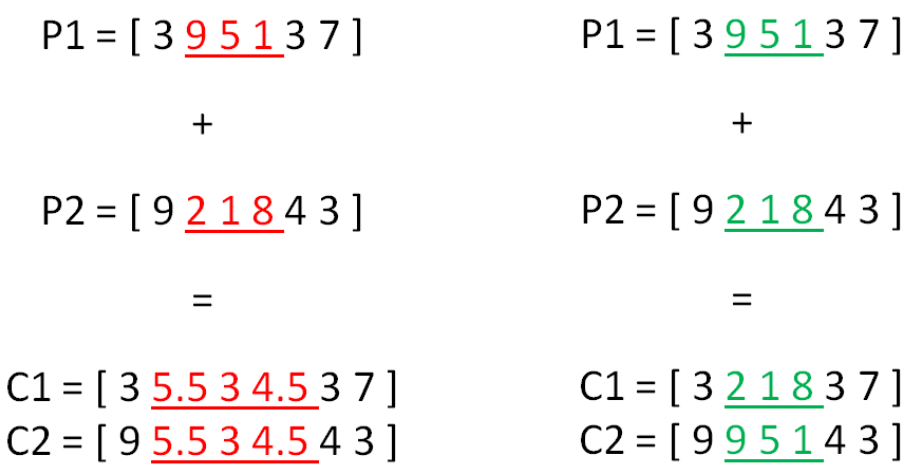

FIGURE 2.4: Genetic Algorithm Crossover Function.
The right hand side of the figure shows combining crossover method. Where a random section, in this case the red underlined section, is averaged with another population member to create two new population members. The left hand side of the figure shows the swapping crossover method. Where a random section, in this case the green underlined section, is swapped with that of another population member to create two new population members.

simulation is split into five subsystems which are as follows: ground target simulation, UAV and gimbaled camera simulation, target selection, plotting the UAVs and ground targets in a 3D plot, and a 2D plot of the estimated versus actual target positions. The first of these subsystems: the ground target simulation is very straight forward. It contains the MATLAB S-functions that are using the derived equations of motion for the ground targets to simulate their motion. The UAV and gimbaled camera subsection is quite a bit more complex then the ground target simulation subsection. As it contains not only the S-functions that are being used to simulate the motion of UAVs and gimbaled cameras, but it also contains the path following algorithms as well as the geolocation and sensor fusion programs. The target tracking step only contains the target tracking

algorithms. It is this section that tells the UAV cameras where to point and what points the UAVs should orbit around. The 3D plot subsection is quite straight forward as it is simply plotting the UAVs and ground targets positions in 3D space. The 2D plot of the actual position versus estimated position for the ground targets does just that. In addition it plots the uncertainty from the Kalman filter around each of the estimated positions. Pictured below in Fig 2.5 is a block diagram depicting the high level control loop for the system. Two visuals are created by the simulation for visualization purposes. A 3D plot that shows the simulated environment and a plot which shows the positions of the objects being tracked, their true position and uncertainty of the estimate. Fig 2.6 and Fig 2.7 show these two plots. Estimate uncertainly is taken from the geolocation EKF rather than the fusion EKF. This will be further covered in the future work section.

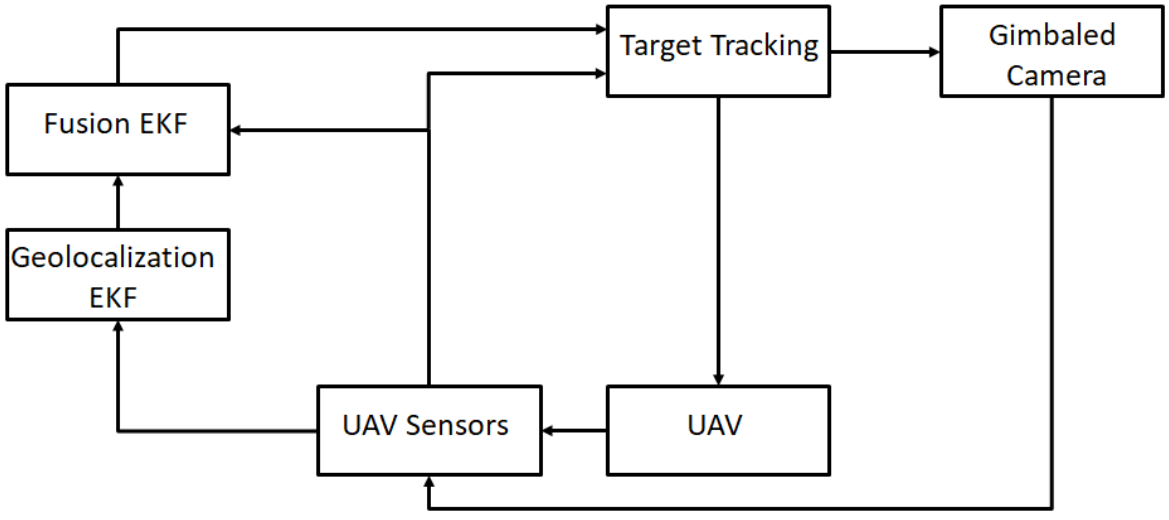

FIGURE 2.5: High Level Block Diagram of the System.

This simulator is based upon the simulator used in [2]. It has been heavily adapted

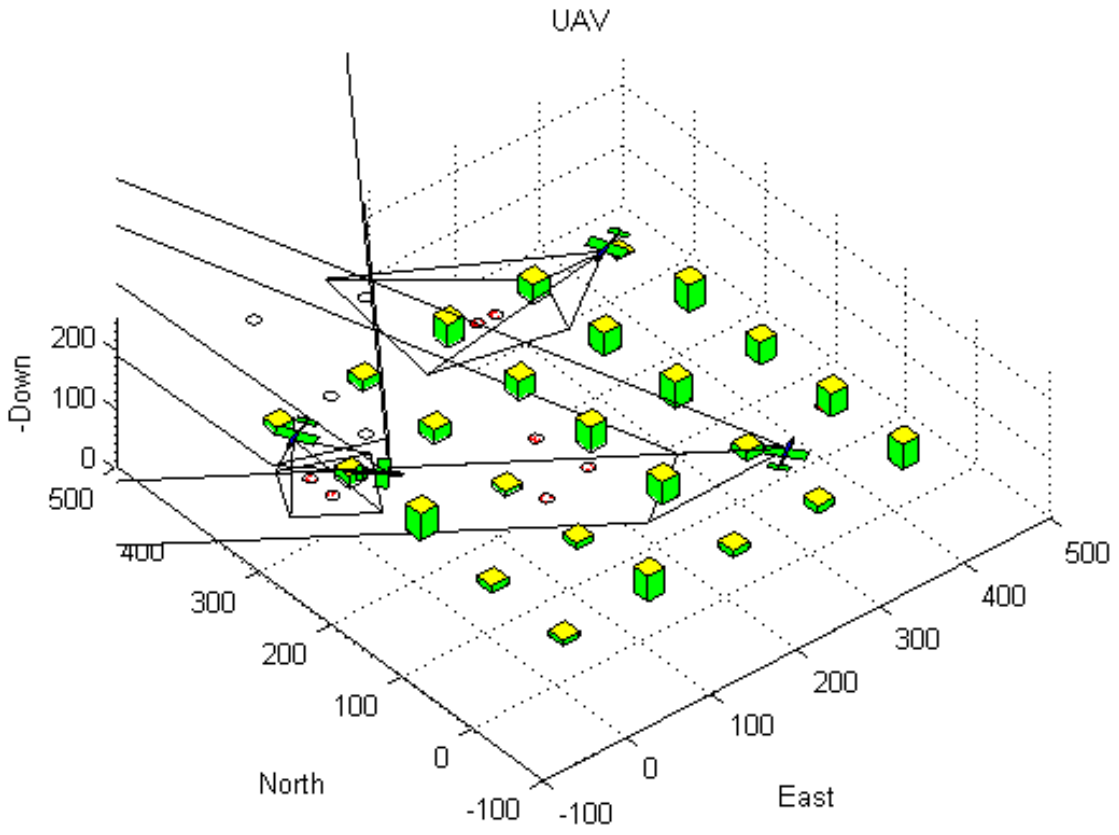

FIGURE 2.6: Simulated Environment.
This image shows the UAVs the ground objects being tracked as well as a visual representation of the UAV's cameras image fields. Simulated buildings are also shown but they have no effect.

to work with multiple UAVs as well as function using the algorithms presented in this work rather then those presented in [2]. Simulation run rate is set at ten hertz. This number was experimented with in an attempt to increase the accuracy of the EKFs. While increasing the hertz rate results in an increased number of update steps in the EKFs. Which does result in increased accuracy in the estimated object positions. It also increases the instability in the EKFs as they have been tuned to run at ten hertz and an increase hertz rate tends to throw them off. Noise was added to the simulated data coming from the guidance model and from the camera gimbal model. The values in

which noise was added are the following: positions of the UAVs, attitudes of the UAVs, ground speeds of the UAVs, course angles of the UAVs, and the angles of the camera gimbals. The process is quite tolerate of noise with the exception of UAV attitudes and camera gimbal angles. Even small deviations in these angles could lead to large changes in the estimated positions of the ground objects. Especially as the distance between the UAVs and the ground targets increases. Ideally the implementation of this process on actual hardware would be done with the highest accuracy gyroscopes possible. As such the noise added to the angle measurements had to be realistic but on the higher end of the accuracy spectrum. Gaussian white noise was chosen for the noise model. The variances chosen for each variable is as follows: UAV positions 10 meters, UAV attitude angles 0.00001745329 radians, UAV ground speed 1 meters per second, UAV course angle 0.08726646259 radians, camera gimbal angles 0.00001745329 radians. While the variance of the angles of the UAV attitudes and camera gimbals angles may seems small it correlates to a measurement error in degrees that can be more than a half of a degree. This simulator was used exclusively on this target tracking portion of this thesis. For the hardware portion of this thesis Gazebo [8] was used in conjunction with ROS (Robot Operating System) [10], the PixHawk firmware [11], MAVROS [12], and MAVLink [13]. The gazebo simulation used for the hardware simulation came from the makers of the PX4 firmware and the PixHawk [9]. These additional simulation tools will be discussed during the hardware chapter.

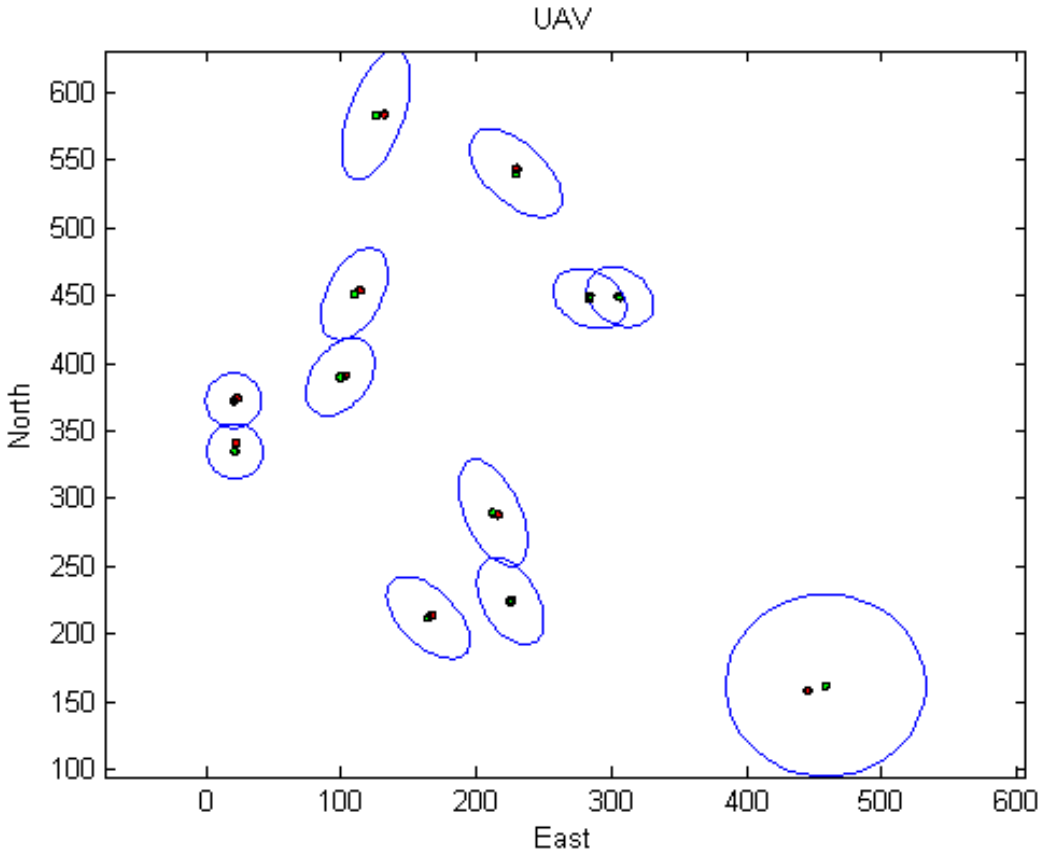

FIGURE 2.7: True and Estimated Position with uncertainty.
Both the estimated position, shown in green, and the true position, shown in red, are plotted in this diagram. The uncertainty is drawn around the each of the estimated positions.

Chapter 3

Results

3.1 Simulation

In this section the results of the MATLAB and Simulink simulations will be presented. All of the results included in this section were done using the noise parameters described in Section 2.4. The simulation run time was limited to one hundred and twenty seconds to limited the amount of data that is acquired as well as to keep the simulation run time down. The results that will be shown in this section are for four UAVs and twelve ground vehicles as well as the cameras on the UAVs. Unfortunately this increases the simulation run time for a single computer to quite a high level depending on the hardware used. On the primary computer use to perform many such simulates the time required was approximately four hundred seconds for four UAVs and twelve ground targets. Using the secondary computer used for simulation this time was pushed off quite significantly

as it has substantially worse computational ability. The operating system used on both computers was Windows 10. The hardware specifications of the primary simulation computer are as follows: processor Intel i7-7700 clocked at 3.6 GHz, graphics processor Nvidia GTX 1080, system memory 32 GB. The secondary computer specifications are as follows: processor Intel i7-4700HG clocked at 2.4 GHz, graphics processor Nvidia GTX 760M, system memory 16 GB.

3.1.1 Analysis of a Typical Run of the Simulator

This section will focus on analyzing a single run of simulation rather then an analysis of many consecutive runs. As the computational requirements of doing this are significantly lower then that of having to perform many consecutive simulations the computer being used was the secondary one. Even with just one simulation run there is a great deal of data to be examined as such not all of the data can be displayed without adding a significant amount of length of this thesis. As such the important information will be displayed while trying to avoid showing only biased portions of the data.

One of the most important factors that needs to be examined in an individual run of the simulation is the error. The purpose of the algorithm is after all to minimize the error in the position of all of the tracked objects. Getting this data on actual hardware is quite difficult as some sort of GPS, inertia navigation process, or a combination of the two would have to be used to get the position data of the objects. These methods

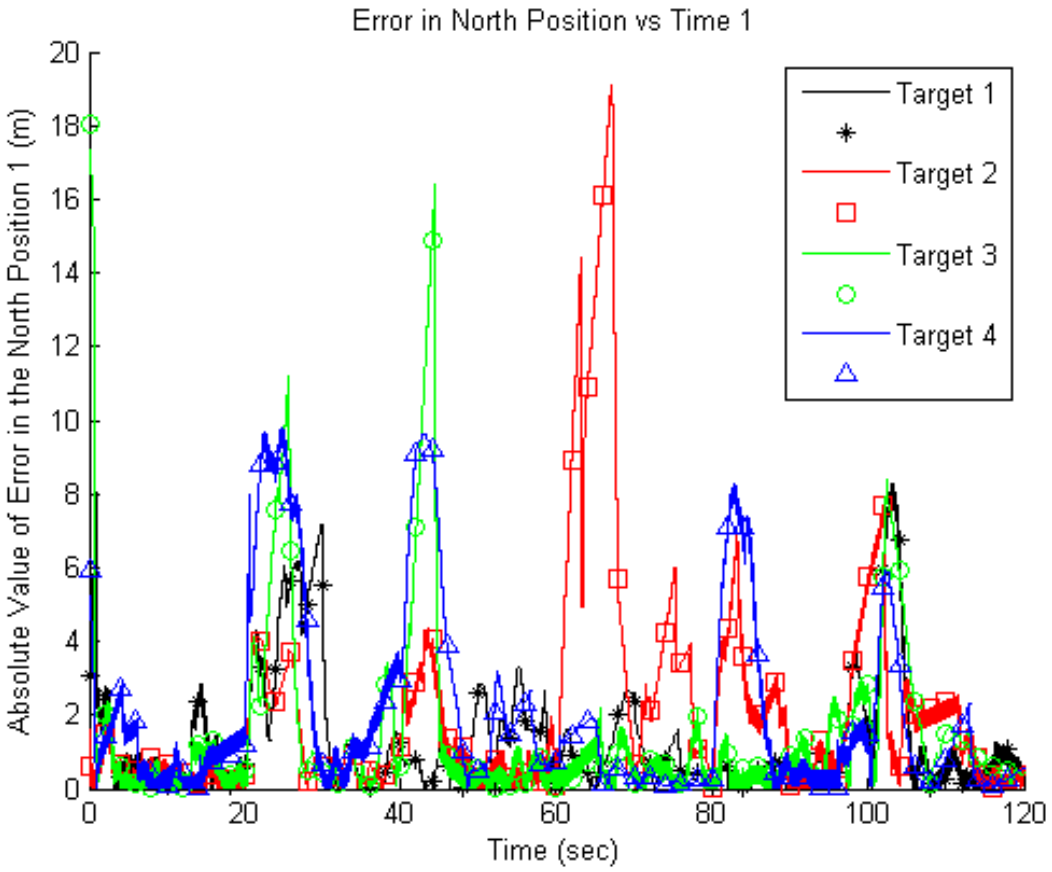

FIGURE 3.1: Absolute Values of the Error in the North Direction for Targets 1-4.

of acquiring the position of the objects are also subject to error. While a high degree of accuracy is possible [14]. These processes are expensive and/or require very precise starting data as is the case with [14]. As such the simulated environment used for this test is the ideal way to get accurate error values as there is no error in true position data. In Fig 3.1 the error of the first four targets are shown in the north direction. In Fig 3.3, and 3.5 the error in the north direction for the rest of the targets is shown. In Fig 3.2, Fig 3.4, and Fig 3.6 the error in the east direction for each of the twelve targets is shown. Each graph has the error for four targets.

As one can clearly see the error is kept quite low for most of the targets however the error

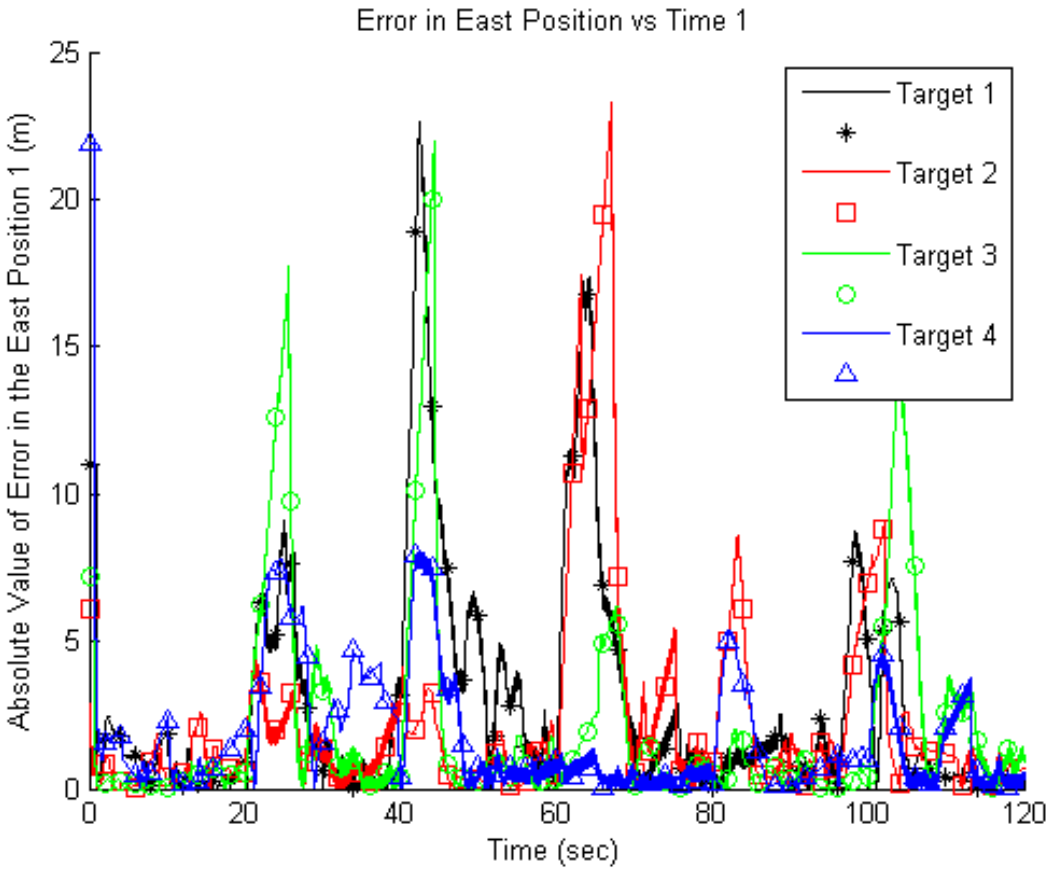

FIGURE 3.2: Absolute Values of the Error in the North Direction for Targets 1-4.

does occasionally increase above the average. Average error can be found in Fig 3.7 and can be seen to quite low. The average north error for all targets is 2.5405 and the average east error for all of the targets is 2.7729. While not the centimeter level accuracy of a properly set up RTK GPS this error is excellent. Especially considering that only four sensor platforms were used for all twelve of the targets. The accuracy could be further improved by introducing higher accuracy sensors on the UAVs. Better gyroscopes and GPSs would increase this accuracy even further. In particular the GPS used on board the UAVs could be substantially improved as the noise model for the GPS units had a variance of ten meters. Using RTK GPS units which potentially offer centimeter level

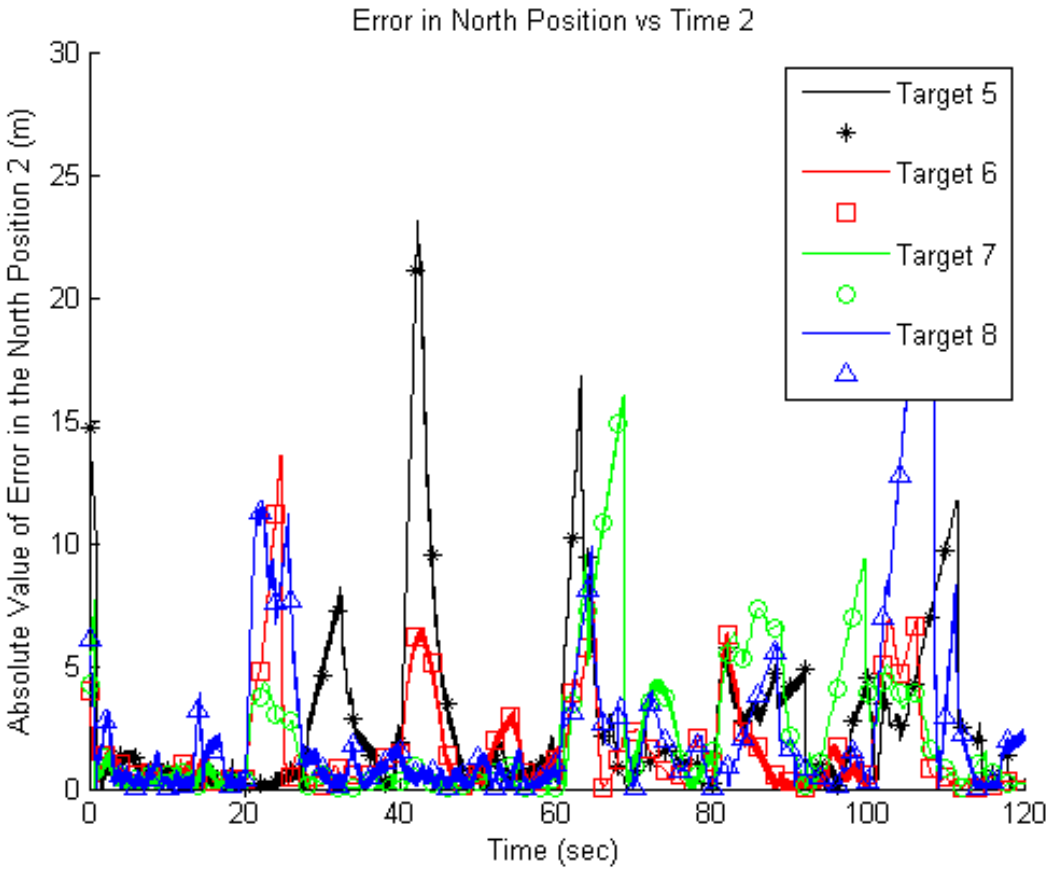

FIGURE 3.3: Absolute Values of the Error in the North Direction for Targets 5-8.

accuracy would be a vast improvement over what was modeled. The gyroscope noise was modeled to have a high degree of accuracy but there are higher accuracy gyroscopes that could be used. Gyroscopes are particularly important as even small variance in the gyroscope measurements could result in massive errors. One of the inherent downside of this form of position estimation is apparent from these plots in the form of higher position errors at times. This behavior is somewhat unavoidable with high number of targets compared to UAVs. Especially when the targets are spread over a wide area. Wide spread target distributions means the only options for this particular algorithm is to switch from target to target based on which target has the highest covariance at the

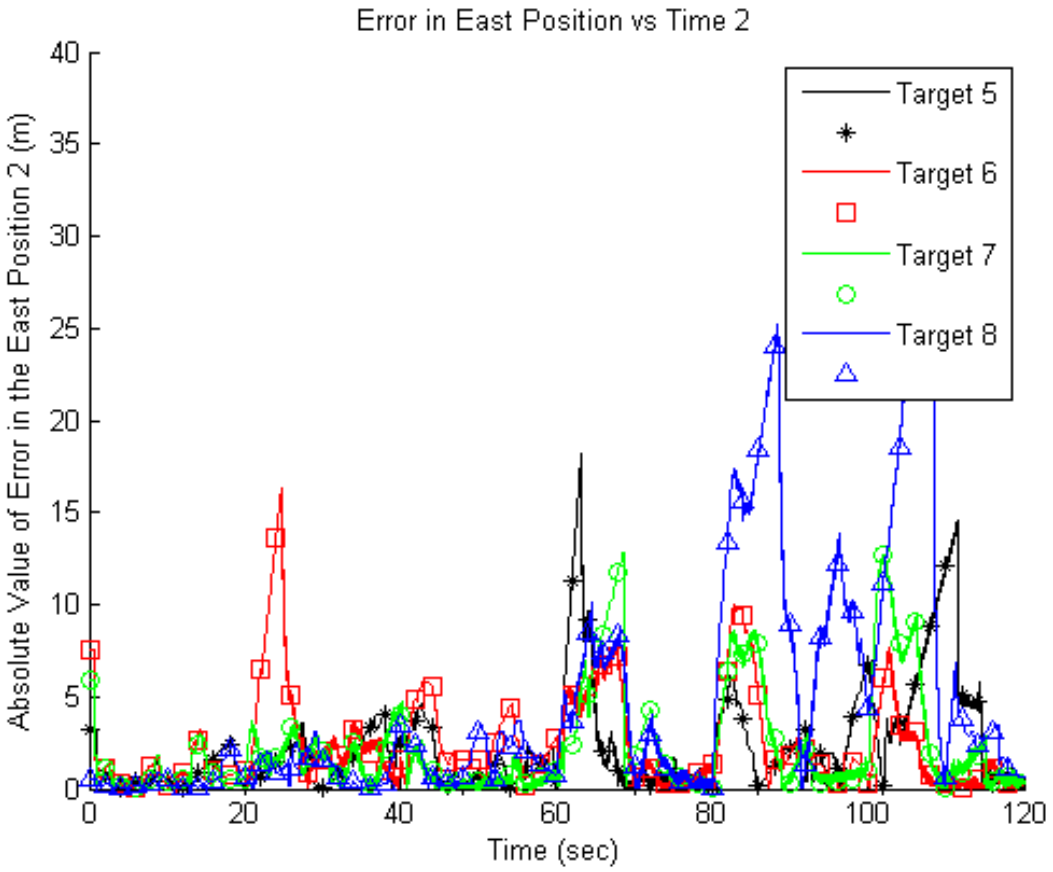

FIGURE 3.4: Absolute Values of the Error in the North Direction for Targets 5-8.

time. These situations can be nullified a bit if UAVs either are at a higher altitude or are farther away from the target. Due to the increased image frame size when the UAVs are farther away from the targets they are able to fit more targets in the image frame. However, this also makes target detection much more difficult. Additionally if the targets are far away in the north or east direction this also damages the estimates by making them much more uncertain. Gyroscope accuracy also matters as great deal due to small differences in angles leading to huge errors when coupled with large distances. Still the largest of these error spikes never exceeded forty meters and these spikes only last for a very short period of time. Also the average error never exceed forty meters amongst

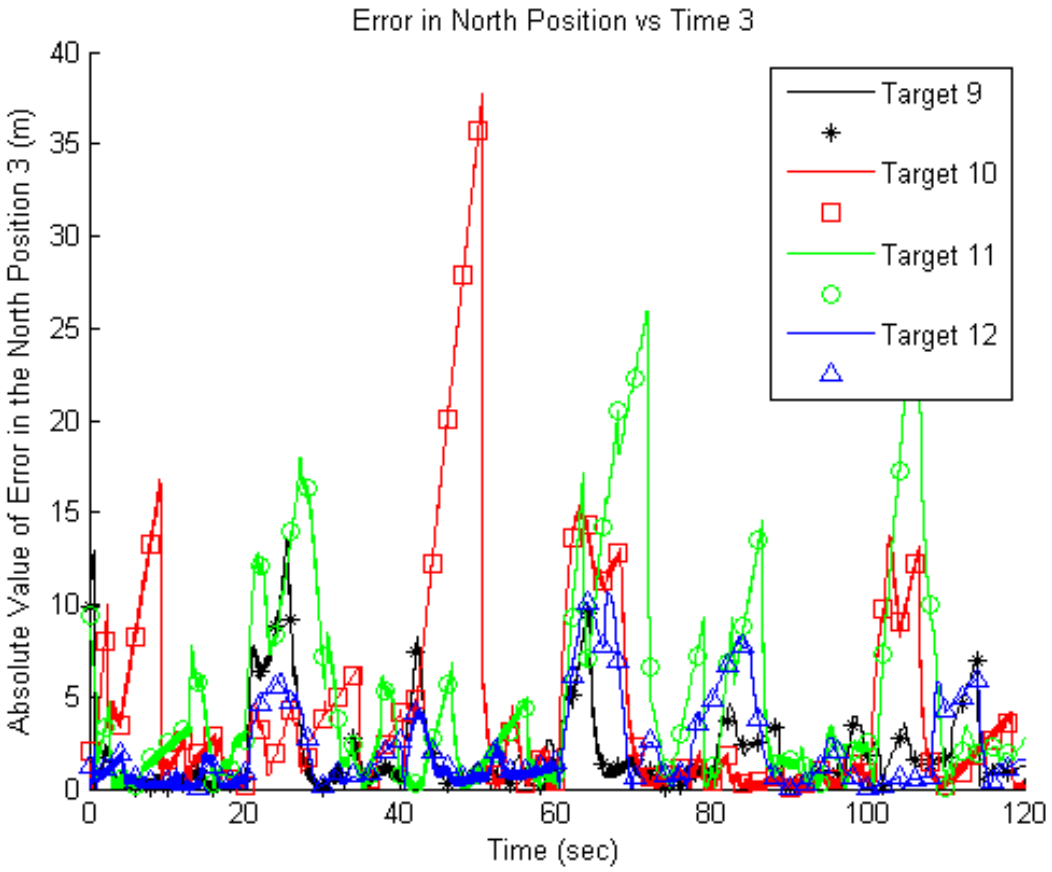

FIGURE 3.5: Absolute Values of the Error in the North Direction for Targets 9-12.

all twelve targets during the entire one hundred and twenty second run. This also goes to show the robustness of this method as though the error increased above 30 meters at times these estimates were able to recover.

Another interesting group of figures of merit are how covariance relates to error. This is done by using a three sigma bounds plot to show the uncertainly in the filter. Generally it can be seen that the error will track with the three sigma bounds expansions and contractions. Position estimates are given by both the geolocation EKF and the fusion EKF. However, the fusion EKF is where the position data is collected from as it takes

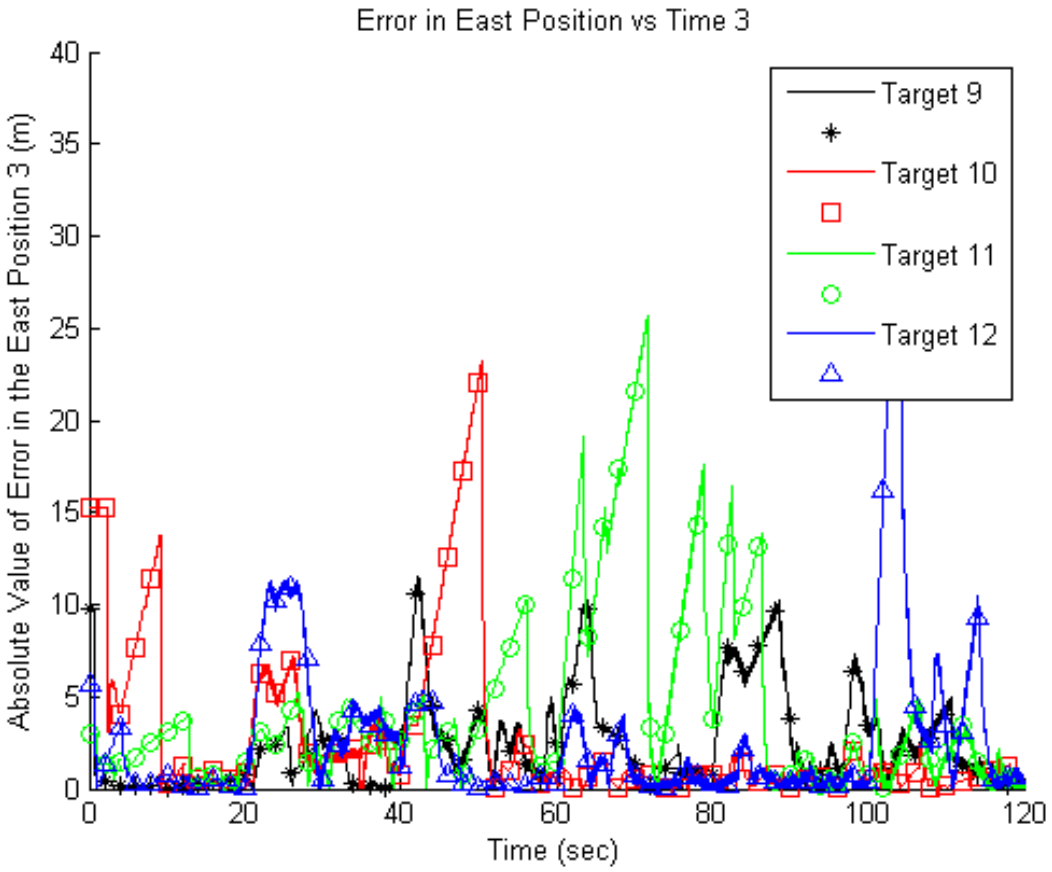

FIGURE 3.6: Absolute Values of the Error in the North Direction for Targets 9-12.

into account all of the geolocation EKFs. That being said it is interesting to see the correlation of the error not just as it relates to the fusion EKF but also the geolocation EKF. Looking at Fig 3.8 and Fig 3.9 one can begin to see how both sets of covariance have bearing on the position estimates. Neither seem to be correlated exactly one to one with three sigma bounds. This makes sense as the source of the updates in the fusion EKF is the geolocation EKF, but there are four geolocation EKFs. The fusion EKF only takes updates under certain conditions with the only consistent parameter being whether or not the geolocation EKF has had an update step. Which in turn makes these two filters inextricably linked. However, only the lowest geolocation EKF covariance is recorded,

	Target 1	Target 2	Target 3	Target 4	Target 5	Target 6
Average Error North:	1.3306	2.1509	1.5051	2.0693	2.7581	1.6608
Average Error East:	2.9788	2.3668	2.2111	1.8579	2.323	2.4305

	Target 7	Target 8	Target 9	Target 10	Target 11	Target 12
Average Error North:	2.1557	2.6522	1.927	4.6414	5.4327	2.2027
Average Error East:	2.2625	4.6959	2.4366	2.7491	4.2934	2.6693

Highest Error North:	5.4327
Highest Error East:	4.6959
Lowest Error North:	1.3306
Lowest Error East:	1.8579
Mean Error North:	2.5405
Mean Error East:	2.7729

FIGURE 3.7: Table of Error Values for a Single Simulation run
This simulation was performed with four UAVs and Twelve ground vehicles.

but multiple could be feeding into the fusion EKF filter. Additionally this may also be due to what data is used to perform measurement updates. The geolocation EKF uses the GPS position to perform the measurement updates. While the fusion EKF is using the geolocation EKF positions. Another reason could be due to the fusion EKF only receiving updates from the geolocation EKF when they have met the criteria of being considered to have a reasonable estimate, but this is not always true. Meaning the Fusion EKF may think that because it is receiving an update the uncertainly should go down, but if the estimate is poor it may be increasing. Potentially masking some of the covariance spikes that are occurring in the geolocation EKF. These three sigma bounds plots do not provide a complete picture by themselves. When looking at the real position plotted against the estimated position the picture of what causes these unexplaned increases in error becomes

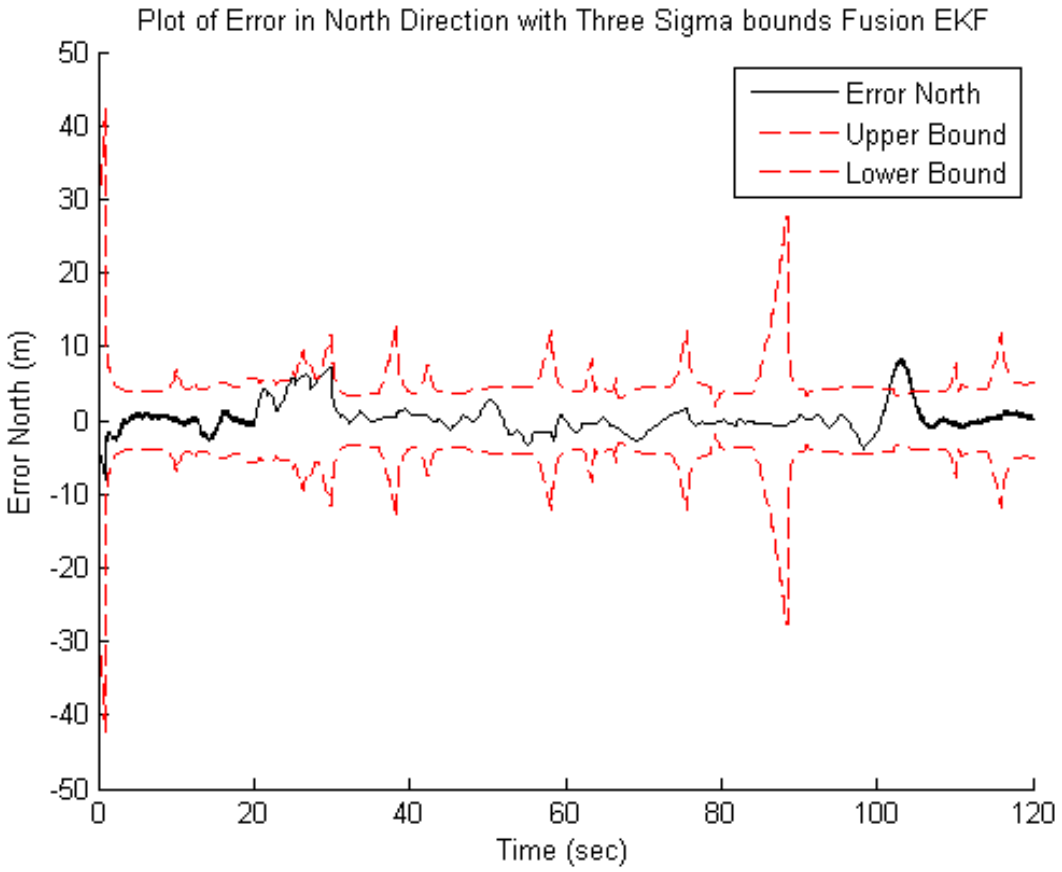

FIGURE 3.8: Three Sigma Bound Plot with error in the North Direction using Fusion EKF

more apparent. The increases a likely due to the targets changing their trajectories. At these instances error will go up due to the abrupt change in motion but covariance will stay relatively consistent. Still the error rarely leaves the three sigma bounds. Meaning that the uncertainly is usually more than the estimate. As covariance is used for decision making processes this means that these decisions can be seen as valid. Figures 3.8 and 3.10 refer to the Fusion EKF. While Figures 3.9 and 3.11 refer to the geolocation EKF.

The final two figures that will be analyzed for the single simulation session will be plots showing the estimated position versus the actual position. One can see the superior

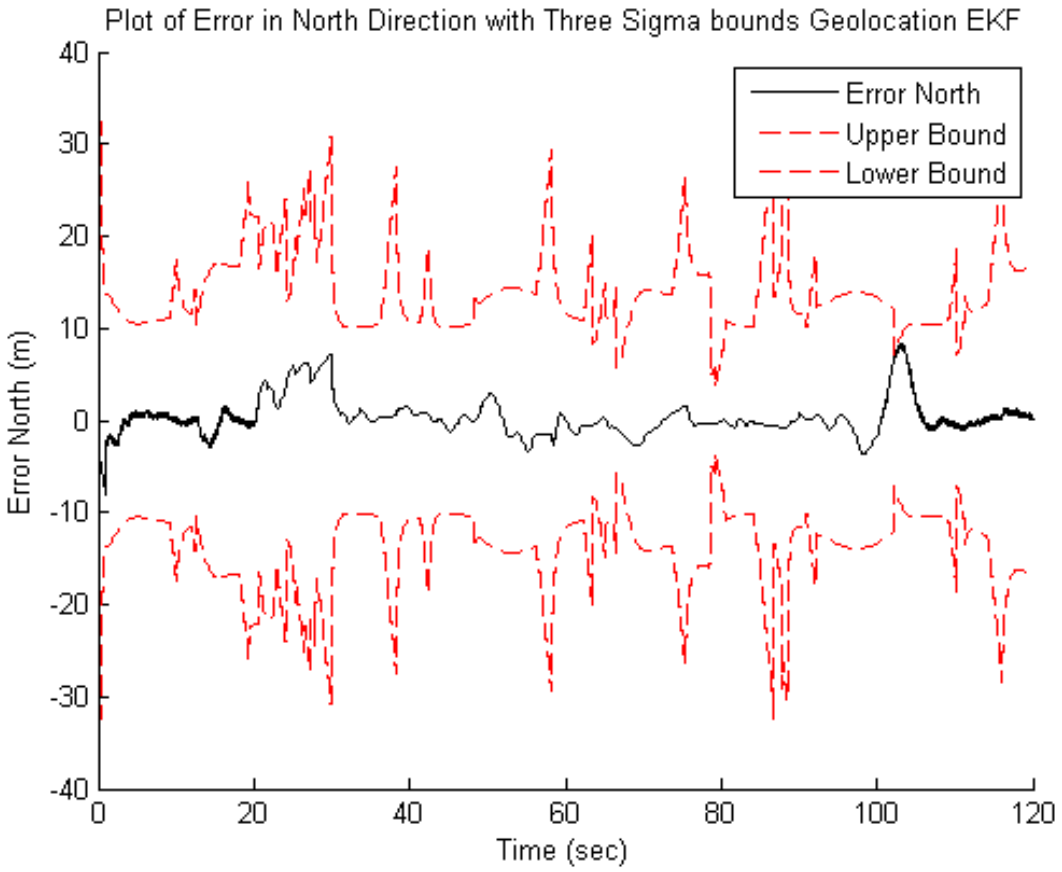

FIGURE 3.9: Three Sigma Bound Plot with error in the North Direction using Geolocation EKF

estimate is excellent as it tracks almost exactly on top of the true values for a large section of the targets motion. The poor estimate is much worse then its counter part. That being said, the estimate is only dreadful at times and at other times the estimates can be quite good. An interesting feature of both plots, that should be noted, is the tendency of the estimates to be much worse when the object changes it motion to head in another direction. The abrupt nature of these transitions is what cause the estimates to worsen. When the EKFs are tracking a particular motion they are trying to estimate where the targets will move when they are not receiving updates. The estimation step occurs on every iteration of the EKF even when an update step is run. If the target

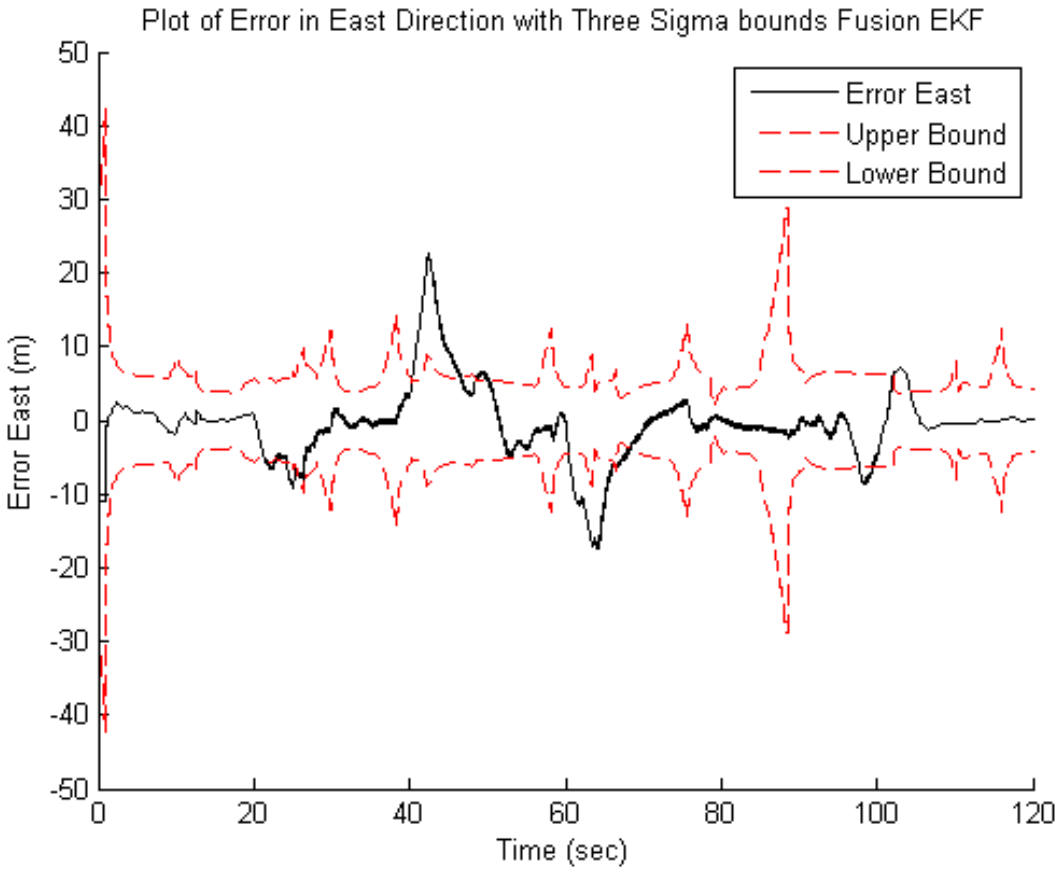

FIGURE 3.10: Three Sigma Bound Plot with error in the East Direction using Fusion EKF

is established as moving in a particular direction the EKF will assume that this is the direction in which the object will move. When the uncertain is low due to many updates occurring in a row this means that the filter will keep wanting to estimate the position following the original motion even when an update step is occurring. Thus, it can take multiple updates steps for the EKF to fully understand the transition of the established motion. This does lead to another potential problem as if a target decides to change trajectories while there are no update steps going on this can lead to much higher errors. This contributes significantly to the higher error spikes that can be seen in the first six figures. To counter act this problem the cameras must regularly switch what targets they

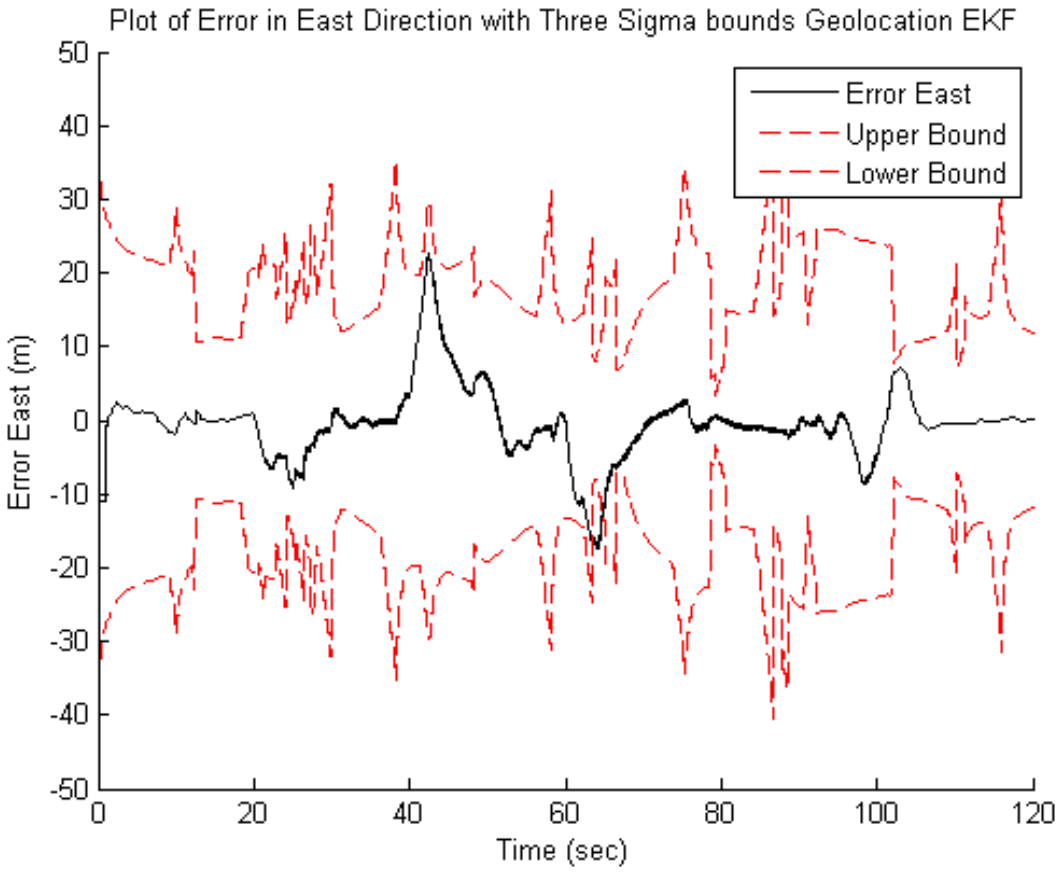

FIGURE 3.11: Three Sigma Bound Plot with error in the East Direction using Geolocation EKF

are looking at often enough that these changes of position are caught. When they are caught the estimated location usually quickly converges to the true position as both EKFs usually have higher covariance values so the measurement values have a great deal more potency. Another potential problem that occasionally occurs is related to the nature of the selection of where to point the camera. The algorithm naturally wants to point at the locations where the most targets can be found. In these types of areas the cost function adds the values of all of the targets into the unified cost value. This makes target rich areas very difficult to resist for the algorithm and this is by design as this minimizes the covariance of the system of targets. The downsides of this method is that targets by

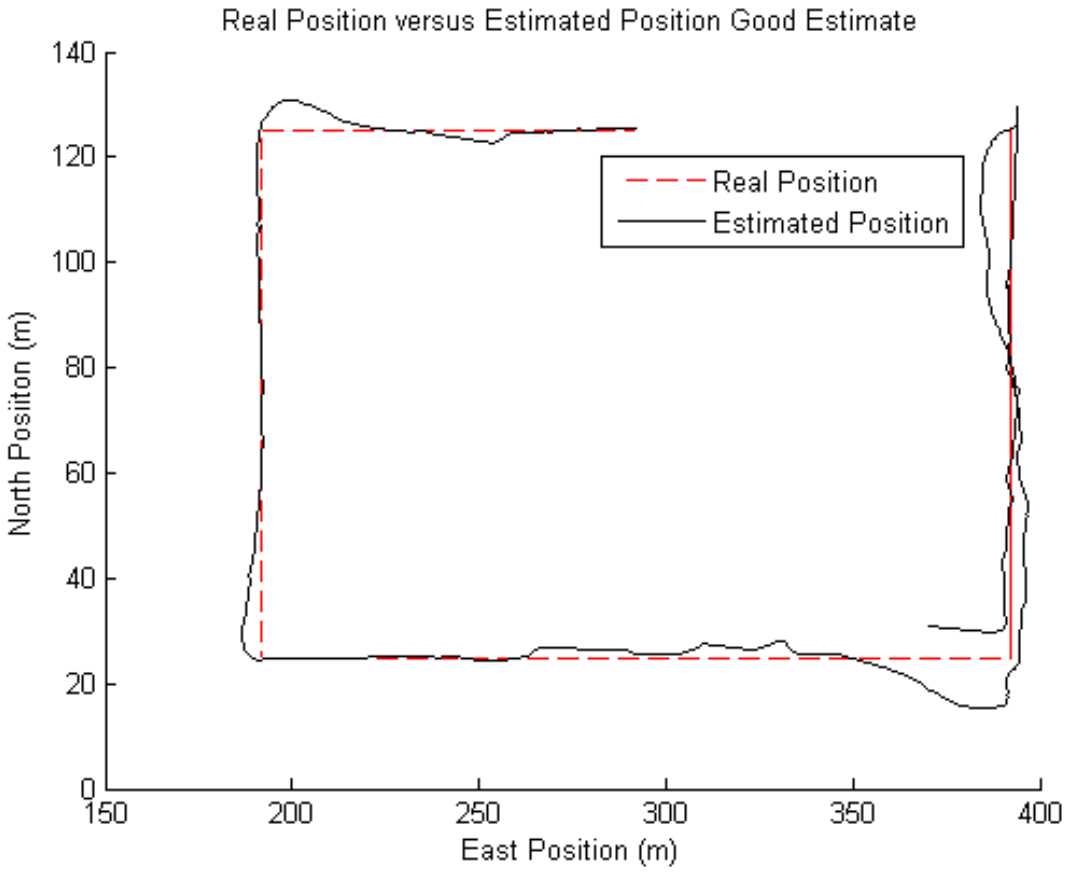

FIGURE 3.12: Example of a Target Trajectory with Good Estimate

themselves accumulate more error. This is why there is a trigger in the algorithm to look at individual targets if their covariance reaches above a certain level. Preventing targets from being difficult or impossible to find again. Figures 3.12 and 3.13 show the excellent and poor estimated positions versus true positions respectively.

3.1.2 Monte Carlo Analysis

Monte Carlo analysis involves running a simulation repeatedly with randomized initial conditions in order to gain some insight into how well a process is operating or how it

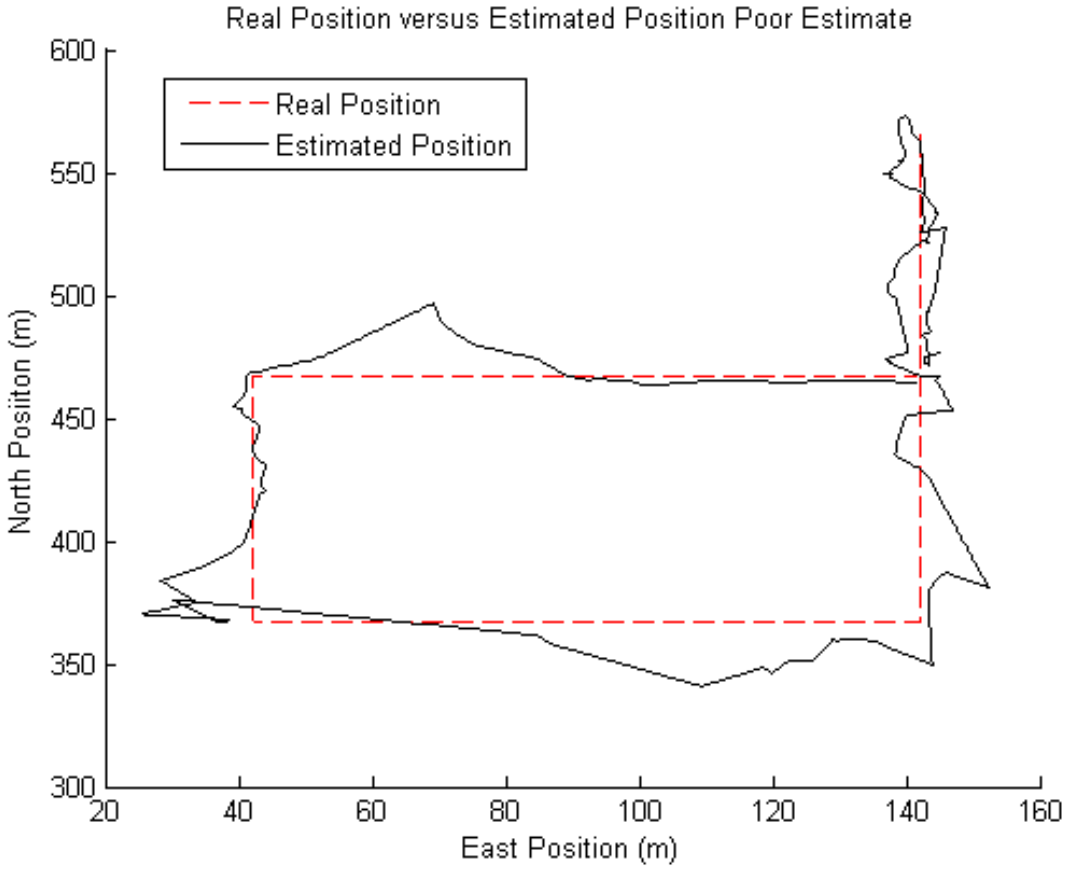

FIGURE 3.13: Example of a Target Trajectory with Poor Estimate

operates. Three cases were run varying the number of ground vehicles each time. Starting with eight and gaining four more vehicles each subsequent time the code was run. There were a total of three iterations and thus an ending total of sixteen ground vehicles. Each complete Monte Carlo simulation run took approximately ten to twelve hours to complete. The UAVs were not chosen for variance as ground vehicles were thought to have more effects on stress testing the system as it would provide more chances for ground vehicles to stray away from each other. More stray targets for the UAVs to track in theory should provide a greater challenge of the decision making algorithms. Increments of four UAVs were chosen as that represents increasing the ratio of UAVs to targets by one. Thus

making the scenarios tested twice, three times, and four times the amount of UAVs to targets. The output of each Monte Carlo simulation is the average error for each of the ground vehicles. Due to this it should be noted that in Fig 3.14, 3.15, 3.16 when average error is used what is meant is the average of the average error. While in Fig 3.17, 3.18, 3.19 what is meant is just average error. As these graphs are only meant to show the error over the course of the iterations of the Monte Carlo simulation.

Looking at Fig 3.14, 3.15, 3.16 one can see information of note about the performance of the decision making and tracking algorithms. One the immediate and extremely predictable trends that can be seen is that as the number of targets to UAVs increases the amount of error also increases. While this trend is true it should also be noted that the average error does not increase by much as the ratio of UAV to targets increases. In fact the increase from eight targets to sixteen is not even a one meter increase of average error. Additionally the maximum error values for a single target in one of the Monte Carlo run does not seem to follow this trend of increase as the number of targets increases. As the maximum average error in the eight target case is larger then in the twelve target case. This trend suggests that outliers in the data are not solely dependent on the number of targets. Instead this could mean that outliers have a significant dependency on target distribution as well as the number of targets. That being said the data trend does clearly show that the number of instances were the average error exceeded ten meters did increase from eight to sixteen UAVs. The increases in these high average error incidents is greater between twelve targets and sixteen targets which would suggest a nonlinear trend in this

	Target 1	Target 2	Target 3	Target 4	Target 5	Target 6	Target 7	Target 8
Avg. Error North	2.5207	3.0527	2.4977	2.3932	2.5745	2.5066	2.5573	3.032
Avg. Error East	2.5163	2.4531	2.2624	2.3977	2.4837	2.4761	2.6669	2.6173

Highest Avg. Error North	3.0527
Highest Avg. Error East	2.6669
Lowest Avg. Error North	2.3932
Lowest Avg. Error East	2.2624
Highest Error North	61.3273

Lowest Error North	0.6801
Lowest Error East	0.5752
Error Greater than Ten North	4
Error Greater than Ten East	3
Highest Error East	33.4453

FIGURE 3.14: Analysis of Eight Target Monte Carlo Simulation

aspect but there is not enough data to confirm this trend. Still the trend of increasing high error events is what was predicted and even in the sixteen target case the number of high error events is not extremely high. As only nine of sixteen hundred average north position estimates exceeded ten meters of average error and only six of twelve hundred in the east direction. Meaning that even in the case where the highest number of average errors over ten meters occurred not even one percent of targets experienced such an event. When target distribution is wide spread the UAV cameras must point at many different locations rather then just a single or a few locations. This invariably leads to higher error in the target position data which in turn increase the chance that a particular target will be lost. If a target is lost by the system it can be very challenging to find it again. Difficulty reacquiring a particular target invariably leads to that particular targets error increase even more.

Figures 3.17, 3.18, and 3.19 are included in this section to show an example of how the data set of average errors looks for each of the three scenarios. There is not a great deal to say about the data is this section that has not already been said. For the most part the

	Target 1	Target 2	Target 3	Target 4	Target 5	Target 6
Avg. Error North	2.846	2.8901	2.7329	3.116	2.9498	2.738
Avg. Error East	2.7437	3.0181	2.5279	2.6974	2.7063	2.9602

	Target 7	Target 8	Target 9	Target 10	Target 11	Target 12
Avg. Error North	2.8926	2.778	3.1709	3.4909	2.8101	3.1135
Avg. Error East	2.7315	3.0123	2.9638	2.8309	2.8446	2.9328

Highest Avg. Error North	3.4909	Lowest Error North	0.8795
Highest Avg. Error East	3.0181	Lowest Error East	0.6593
Lowest Avg. Error North	2.7329	Error Greater than Ten North	8
Lowest Avg. Error East	2.5279	Error Greater than Ten East	1
Highest Error North	47.708	Highest Error East	14.5284

FIGURE 3.15: Analysis of Twelve Target Monte Carlo Simulation

	Target 1	Target 2	Target 3	Target 4	Target 5	Target 6	Target 7	Target 8
Avg. Error North	2.8135	3.1301	2.7943	3.0432	3.1439	2.6405	3.184	3.3502
Avg. Error East	2.7103	3.1132	2.8286	2.8964	2.865	3.0087	3.005	3.3077

	Target 9	Target 10	Target 11	Target 12	Target 13	Target 14	Target 15	Target 16
Avg. Error North	3.0828	3.3205	3.0551	3.215	3.0083	5.1993	3.337	3.3646
Avg. Error East	2.9163	3.5829	3.1175	3.0731	3.123	3.5899	3.3864	3.2364

Highest Avg. Error North	5.1993	Lowest Error North	0.7975
Highest Avg. Error East	3.5899	Lowest Error East	0.8176
Lowest Avg. Error North	2.6405	Error Greater than Ten North	9
Lowest Avg. Error East	2.7103	Error Greater than Ten East	6
Highest Error North	139.5619	Highest Error East	24.2311

FIGURE 3.16: Analysis of Sixteen Target Monte Carlo Simulation

average error is kept quite low by the tracking and decision making algorithms. Though there are a few notable exception for which potential explanations have been provided.

FIGURE 3.17: Plots of Average Error Versus Monte Carlo Simulation Iteration Eight Target Scenario

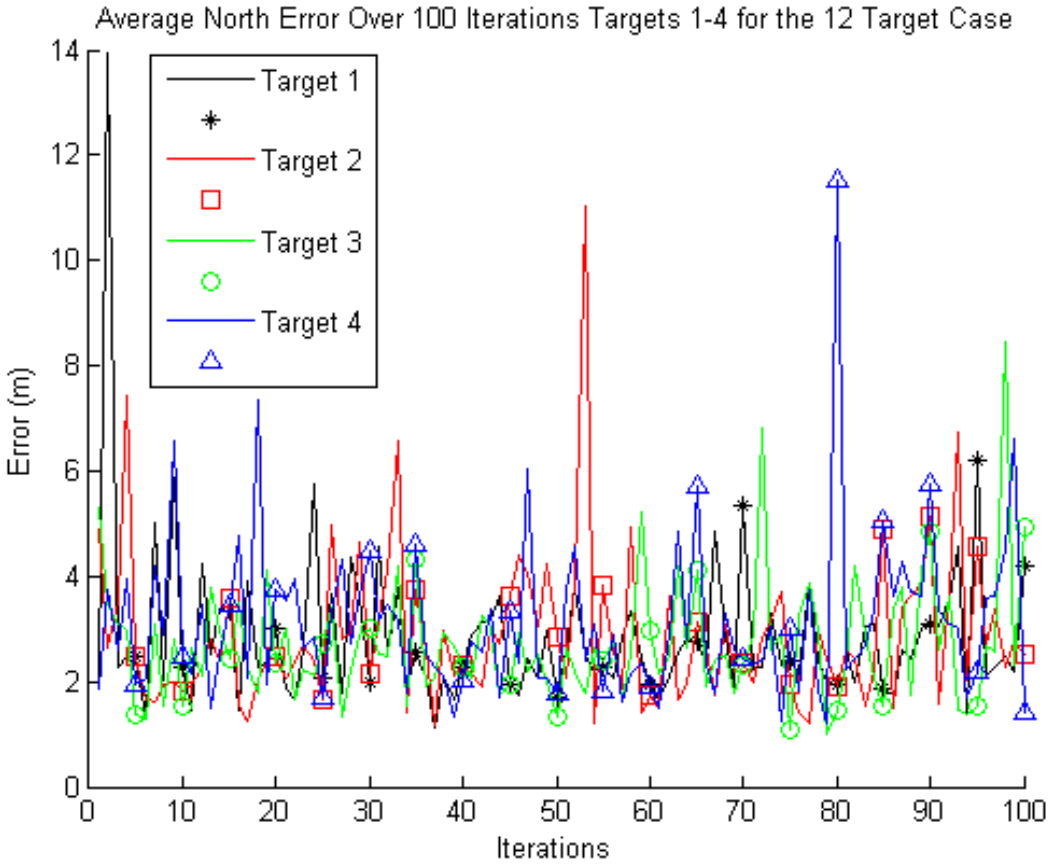

FIGURE 3.18: Plots of Average Error Versus Monte Carlo Simulation Iteration Twelve Target Scenario

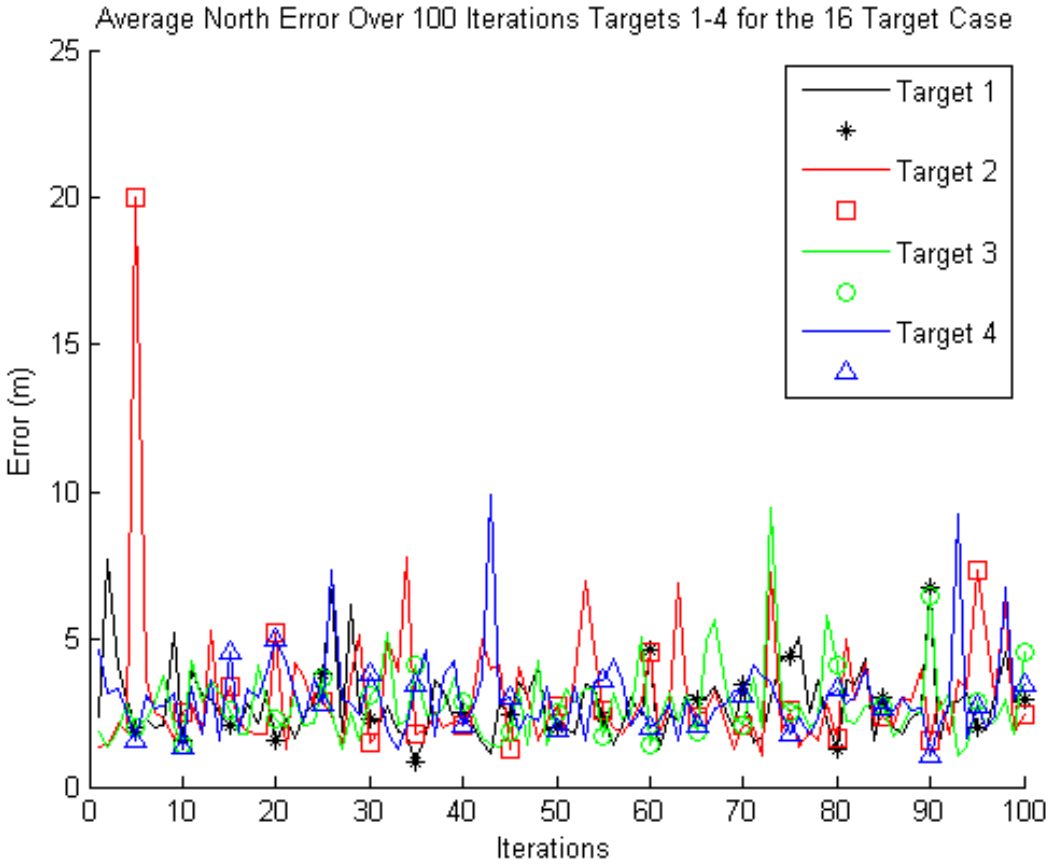

FIGURE 3.19: Plots of Average Error Versus Monte Carlo Simulation Iteration Sixteen Target Scenario

Chapter 4

Hardware

Two different hardware experiments were performed. One to test an aircraft platform and the other to test the viability of online geolocation. While both of these are important pieces in the target tracking methods described in the simulation sections, neither of these things alone is enough to get the full method working. These two things are quite important and together make up a significant aspect of the methods shown in simulation. Unfortunately, testing the full simulation method is complicated by a great deal of safely and current hardware limitations. The first of these problems is bandwidth requirements for multiple UAVs transmitting images and position data to each other and to the ground station. Testing only a single camera transmitting a low resolution image had a small but noticeable amount of latency. Scaling up this system would require the bandwidth to be similarly scaled up to avoid saturating the wireless communication networks bandwidth. Many object detection methods are improved with higher resolution images. Higher

resolution images obviously result in larger file sizes which further increases bandwidth requirements. To get around the bandwidth problem the image detection would have to take place on board the UAVs. This also presents problems as the hardware capable of performing the object detection is quite a bit larger then the on board computers that were used during this research. Thus, using superior hardware necessitates the increased size of the needed aircraft platform. In addition to the afore mentioned problems there is also the issue of flying multiple aircraft at once. This requires multiple safety pilots and spotters as well as additional autopilot measure put in place to avoid mid air collisions.

The autopilot that was chosen was PX4 running on the PixHawk 2.1 [11]. This firmware was chosen due to its prevalence. Which enables solutions to many problems to be found quickly. Additionally the PixHawk 2.1 is fantastic hardware that provides excellent results. Difficultly with one of the sensors on the PixHawk 2.1 was encountered in the form of a magnetometer that would not give correct compass readings. This was fixed relatively easily by simply turning off the affected sensor in the firmware parameters. PX4 provides in build tools that are able to control a fixed wing aircraft to travel predefined paths through the use of global waypoints. Unfortunately, lower levels of control are blocked off in fixed wing aircraft mode and are only usable through modifications to the source code. Two programs were created to issue waypoints to the PixHawk from the on board computer. One which used local waypoint to define where the UAV should go and then translated these to global waypoints. While the other issued only global waypoints. Only the program that issued commands solely through global waypoints was tested in the air. Both controllers were tested on the ground. The reason the controller

that only used global waypoints was the only one tested in the air was for the precision and simplicity it provides when planning where the UAV will go. The local waypoints controller defines its home position as the first GPS values it receives. This home position forms the basis of the local coordinate system. While this allows for the convenience of using local coordinate systems to plan missions it also changes the published waypoints based off where the UAV is first activated. Leading to deviations in the UAVs intended path. This can be fixed by setting the home position before hand. Ultimately as only a single waypoint was used in the flying mission it proved more convenient to simply assign the necessary waypoint. Fig 4.1 shows the simulated results of a multiple waypoint mission using the local coordinate based controller.

The simulator used to mimic the characteristics the PixHawk 2.1 and a fixed wing aircraft of similar design to the Strix Stratosurfer was Gazebo [8]. Gazebo is an open source simulator that can be used in conjunction with ROS. Through Gazebo and ROS the PX4 firmware used in the PixHawk 2.1 can be replicated quite closely. ROS is a communication frame work originally created for robotics applications but has been applied to UAVs as well. Its purpose is to send data back and forth to different programs which run the code necessary to control the aircraft. As well as position estimation filters and communications with the ground station. ROS is run as normal in the simulation but rather then interfacing with the actual hardware it interfaces with the simulated hardware in Gazebo.

The aircraft chosen for the test flights was the Strix Stratosurfer with the PixHawk 2.1 for the flight controller, and an Odroid XU4 for the on board computer. A modified Ubiquiti

FIGURE 4.1: Plot of Simulated Flight Test using Multiple Waypoints
This plot was created using QGroundControl for display purposes. QGroundControl can also be used to send missions to the PixHawk 2.1 flight controller. This was not done in this case. Instead a custom waypoint assignment program was used that translates local coordinate waypoints into global waypoints.

Bullet and a Ubiquiti Rocket were used for the communication network. Both models of Bullet and Rocket communicated in the 5.8 GHz spectrum. The antennas used for the Bullet and Rocket were Fat Shark 5.8 GHz omni-directional antennas. The modifications to the Ubiquiti bullet were the removal of the hard plastic casing and changing its antenna connector to the Fat Shark antennas connector type (SMA). The Strix Stratosurfer was chosen for its excellence internal capacity. As it uses a molded plastic fuselage it is more spacious than other aircraft of its class. The Odroid was used for its performance relative to size as well as low cost. The Ubiquiti Bullet and Rocket were used as they are reliable as well as have excellent bandwidth by using the 5.8 GHz communication band. Though it should be noted that using such a high frequency band does not allow for an

especially large communication range. The wireless network is required in order to get a reading from the on board computer to the ground station and back. ROS has a feature that allows it to be connected to many computers at once as long as there is some form of connection between them. As a wired connection is not practical on the fixed wing platform a wireless network is used. Mainly the requirements for each of the components used was to have an excellent size to performance ratio. As space is extremely limited on small fixed wing UAVs. A large blended body flying wing would be one of the better choices for applications of this research but this type of aircraft is not readily available in the current commercial market.

For the hardware test of the UAV only a single waypoint was issued to the flight controller. Primarily this was because the gains of the flight controller had not been tuned for this particular aircraft, but rather for a similar aircraft. So it was not known how the aircraft would respond. The waypoint that was issued was for a loiter waypoint with a loiter diameter of fifty meters. The loiter waypoint was chosen as only a single waypoint was issued, to put the flight controller to the test by having it maintain a circular trajectory, and the methods developed in Chapter two have the UAVs fly loiter waypoints as well. Two autonomous flights were flown. Each with the same waypoint. The results of the first flight test are shown in Figures 4.2 and 4.3.

Experiments were performed to test the capabilities of the current hardware to perform geolocalization in real time in addition to the flight testing. Objection detection was performed using color detection algorithms provided by OpenCV [15]. White was chosen as the color to be detected as it provided excellent contrast to the floor around the target.

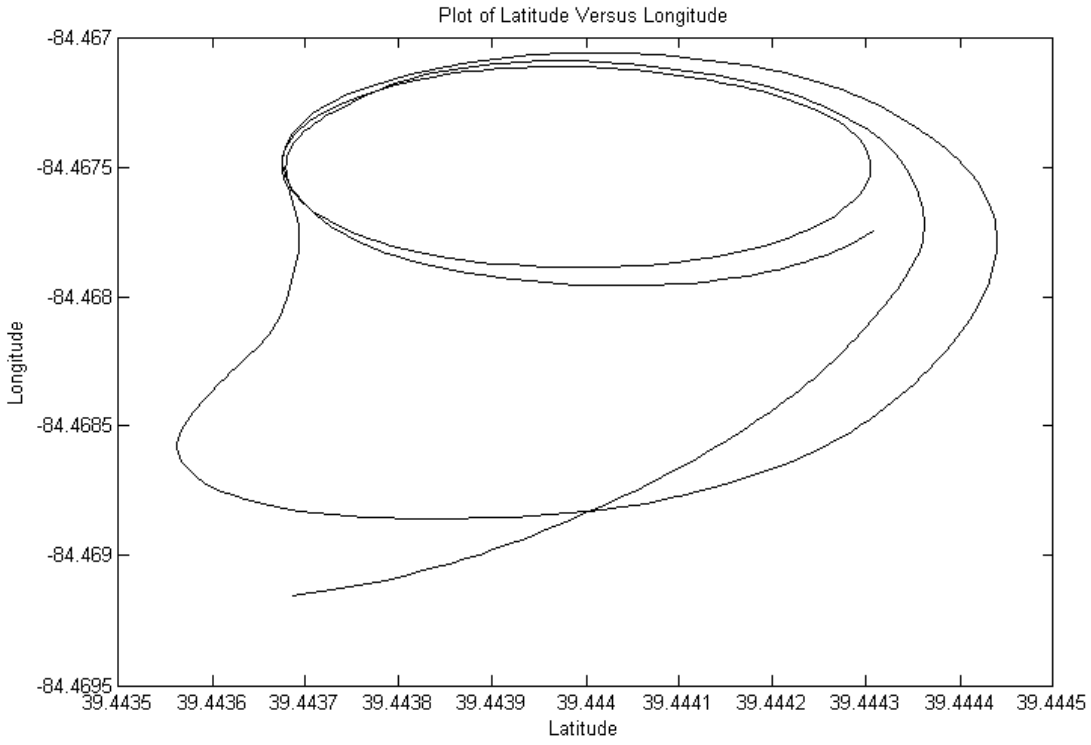

FIGURE 4.2: Plot of Latitude and Longitude from First Flight Test

First the detection was attempted on the Odroid XU4 computer but this proved to be to intensive for the computer to handle when coupled with the other tasks it was suppose to perform. Such as running an EKF with the same structure as the Fusion EKF mentioned in Chapter 2 as well the the process of geolocalization also described in Chapter 2. Fig 4.4 shows what the color detection algorithm is doing. Essentially the output of this code is the pixel location of the center like group of white pixels. The white screen was used as it is also a source of light which makes it easier for the camera to detect the white pixels. This was needed as color detection is very finicky and often either will not detect objects it should or detects objects is should not. This necessitates an other form of object detection or at least a substantial upgrade to the reliability of the color detection

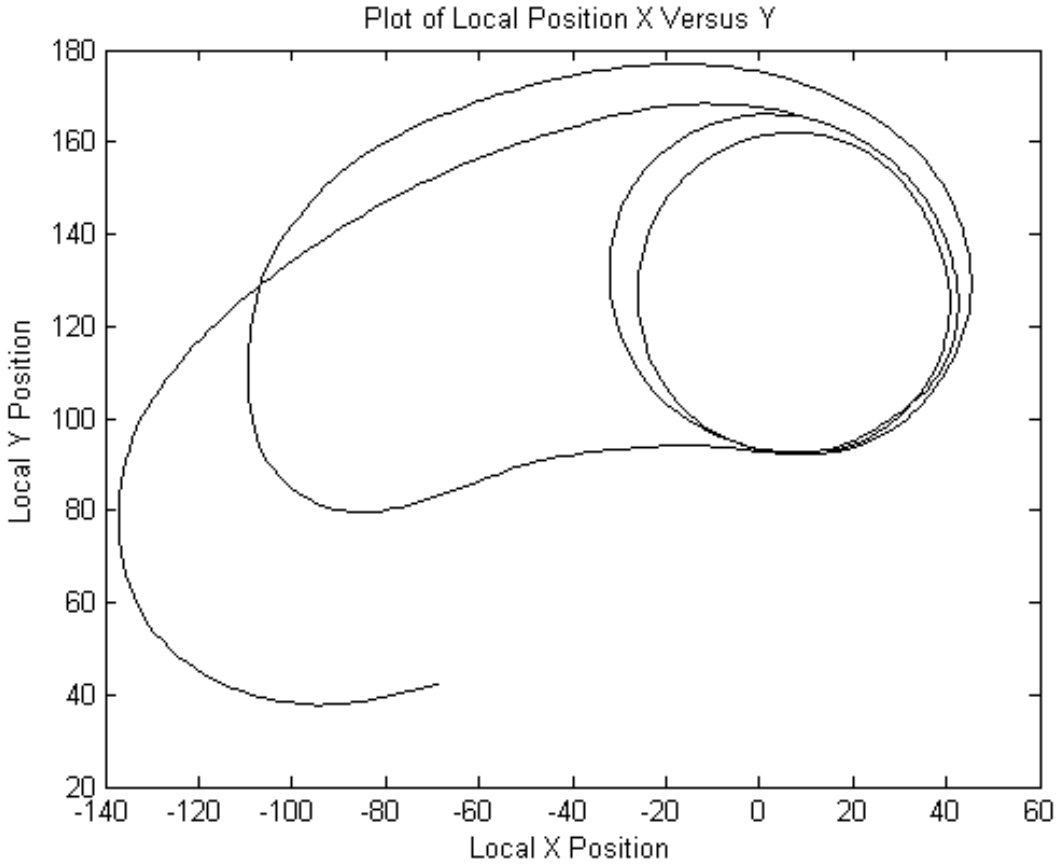

FIGURE 4.3: Plot of Relative Position in Meters X and Y from First Flight Test

used during this research. It should also be noted that the color detection python coded that was used is based off of the code written by [16]. To counter the inadequacy of the Odroid XU4 to perform all of the necessary function without substantial latency the object detection, geolocation, and EKF estimation were done on the ground station. To enable this ROS was was used to transfer the images from the Odroid to the ground station computer. This lessened the computational strain of the Odroid significantly and also greatly lowered the latency. Additionally as the position estimates are published to ROS it is available to the Odroid shortly after it is calculated by the on board computer. While not the most efficient solution as it relies on a signal to and from the the ground

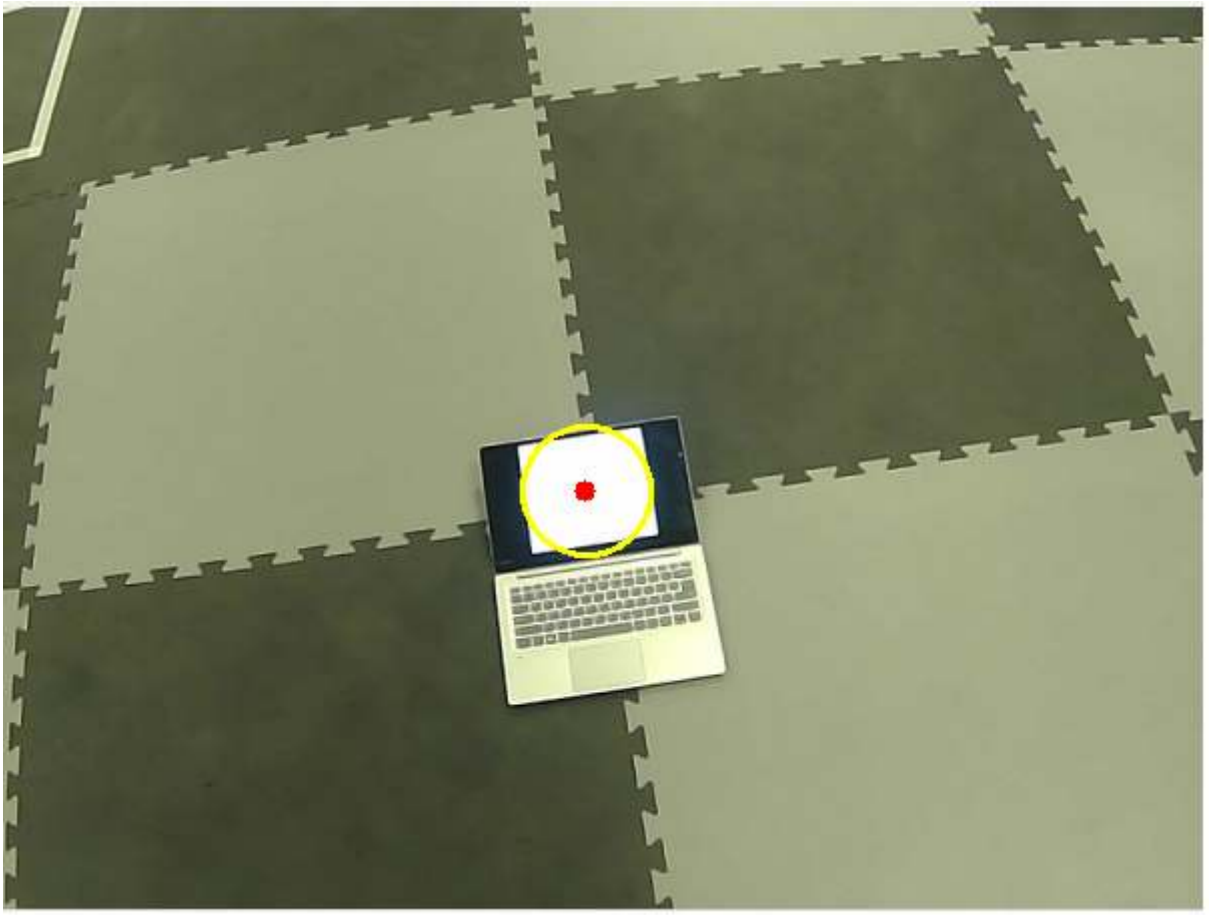

FIGURE 4.4: Color Detection Frame Showing Calculated Center

station. It still provides estimates faster than the Odroid doing everything by itself.

The experiment that was preformed to test if the geolocalization was working properly was done using the target from Fig 4.4. The camera was mounted to a thin but stiff board facing downwards. Also mounted on the board was an Odroid computer, a PixHawk 2.1, and a battery. It was assumed that the gimbal angles were fixed and did not change. While the angles taken from the PixHawk were used aircraft rotation matrix from geolocation method described in Chapter 2. This board was placed on top of the head of a person who's height is approximately 175 centimeters. From the persons height the height of the camera is known. By tilting his head up the camera was able to observe the target. Using

data from the PixHawk and the known parameters of the person's height the geolocation algorithm had enough information to provide a distance from the person to the target. This distance is approximately one meter. This distance comes from the tiles used in the room where this experiment was performed. Each tile is approximately one meter square. With this completed, the position in x and y from the person was 0.64 meters and -0.41 meters rounded to the second significant figure. Obviously this is not correct at all but when looking into the angles that were coming from the PixHawk it was determined there was a bias due to incorrect placement relative to the camera as well as an internal bias as well. It should be noted this PixHawk was not used for the flight test but was a back up unit. As the unit used for the flight test was ensured to be properly calibrated. With these sources of error considered the bias turned out to be 2.5 radians in the yaw axis as well as flipping the sign on pitch axis. With these corrections the estimates turned out to be 0.73 meters in the x direction and -0.05 meters in the y direction. This is much more inline with the expected results. There is still a hefty error but given that the geolocation equations are very sensitive to any forms of angle noise it is not surprising to see this amount of error. Particularly when considering the corrections required to fix the bias in the yaw axis.

Chapter 5

Future Work

Problems as complex as the one this research attempts to solve have many aspects. As such there is still pieces of this problem that have not yet been explored in earnest. For instance there is still the significant challenge of object recognition. Only one of the most basic form of image recognition was explored on actual hardware in the form of color recognition. Many problems with this form of objection recognition exist and it is quite clear that these issues must either be addressed or a new form of object recognition must be used or developed. It is not an understatement to say that this is a significant challenge. The best solution for this problem more than likely lies in the use of deep neural networks. Another issue with image recognition lies with the difficulty in planning for what types of targets the system needs to be able to recognize. A neural network or system of neural networks can only be trained to recognize so many features. However, the number of targets that the target tracking method developed during this research can

be used for is essentially infinite. All that matters to this algorithm is that the target can be recognized.

Camera resolution is another issue that will need to be addressed. Higher resolutions will make object recognition less difficult but put more strain on the system. Particularly when it comes to image transfer. The hardware solution presented in this paper uses a wireless network to transmit the images from the computer on board the UAV to a more powerful computer on the ground. As the size of the images increases this puts increasingly more strain on the bandwidth of the wireless network. Eventually this will cause a great deal of latency in the image transmission. Speed is very important as the sooner the updated positions estimates are sent to the UAVs the better the results of the algorithm will be. Faster frame rates also means more updates in the geolocation EKFs also leading to better estimates of the targets' positions. Optimally the object detection will be done on board the UAVs as this will substantially reduce latency, but this would require significantly larger UAVs. Additionally this would lead to increased UAV unit cost. It could also complicate how easily this type of research can be preformed. Additional UAV size, and powerful on board computing could end up pushing the weight, of the UAVs which will need to be used, past fifty-five pounds. Adding complex and less flexible governing laws to already difficult research. Undoubtedly, there are probably no aircraft that will have the size, and performance requirements for this specific use case necessitating another time consuming and difficult task of designing such a platform or heavily modifying and existing one. There are essentially no simple and/or cost effect methods of working around this problem.

While the algorithms developed over the course of this research provide excellent results there are many potential methods to boost performance to even higher levels. All of the position estimation is done through the use of EKFs but other filters can potentially provide superior performance. For instance particle filters should provide better position estimates given enough particles at the cost of increase computational cost. There are also many parameters that could be adjusted to get superior performance in the various algorithms themselves. For example the parameter that sets the length of time each UAV's camera spends looking at a given location. Right now this parameter is static as is the position to look at but these things could be made to be more dynamic. Not only can the time spent looking at each cluster of points be optimized or even made to be dynamic depending on the covariance of other targets but the focal point could be made dynamic within the same group of points. These changes would take some time to be implemented but if done efficiently could decrease the error of the system of targets. Of course this would also increase the computational cost and this factor as always would need to be weighed against a potential boost in performance. Neural networks could also potential be used to attempt to minimize the error of the system of targets. This would be very challenging though as one would need to determine the optimal place for each UAV's camera to point at each time instance of the simulation in order to train the neural network. Getting these perfect pointing locations is not impossible but it would require an iterative process. Overall it may not be practical based off of the computational cost of training such a network. Geolocation could be changed to running just the algorithm and not the EKF that is currently being run. This change would reduce the computational

cost of the sensor fusion and could potential eliminate some of the remaining sources of estimate instability or at least lessen them. The geolocation EKFs were kept in this version as they provide a convenient source of data for the cost functions in the decision making algorithms.

Chapter 6

Conclusion

Ultimately, the purpose of this research was to provide a method of tracking many objects with limited UAV sensing platforms. In this respect, the methods created over the course of this research have shown incredible potential to provide such a solution. While also maintaining a great deal of robustness with respect to the amount of noise in sensor measurements as well as in preventing the EKF measurements for becoming unstable. There is still work that needs to be done to take this research and fully implemented it on hardware. This will also be a substantial challenge as the technologies needed to implement this tracking method on small scale UAVs may not quite be possible. Regardless the potential of algorithmic approach that was created for this research is significant as there are a great deal of potential applications. Ranging from military, to law enforcement, and even traffic management. Additionally even if these methods prove to be to difficult to run a small scale UAVs. Larger scale platforms could easily be designed to still maintain

a relatively small size but be able to preform the very intensive object detection methods that are required for any such approach to function.

Bibliography

[1] Sinha, A., Kirubarajan, T., and Bar-Shalom, Y., Autonomous Ground Target Tracking by Multiple Cooperative UAVs, Aerospace Conference IEEE, 2005.

[2] Sharma, R., and Pack, D., "Cooperative Sensor Resource Management for Multi Target Geolocalization using Small Fixed-wing Unmanned Aerial Vehicles, AIAA Guidance, Navigation, and Control (GNC) Conference, 2013.

[3] Barber, D. B., Redding, J. D., McLain, T. W., Beard, R. W., and Taylor, C. N., "Vision-based Target Geolocation using a Fixed-wing Miniature Air Vehicle, Journal of Intelligent and Robotic Systems, 2006.

[4] Beard, R. W., and McLain, T. W., *Small Unmanned Aircraft: Theory and Practice,* Princeton University Press, Princeton, 2012.

[5] Lloyd, S. P., "Least Squares Quantization in PCM, IEEE Transactions on Information Theory Vol. IT-28, No. 2, 1982.

[6] "MATLAB", *MathWorks*, November 27, 2018. [Online]. Available: https://www.mathworks.com/ [Accessed: November 27, 2018].

[7] Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press. ISBN 9780585030944.

[8] "Gazebo," *gazebosim.org*. [Online]. Available: http://gazebosim.org/. [Accessed: November 27, 2018].

[9] "Gazebo Simulation," *dev.px4.io*. [Online]. Available: https://dev.px4.io/en /simulation/gazebo.html. [Accessed: November 27, 2018].

[10] "ROS," ros.org. [Online]. Available: http://www.ros.org/. [Accessed: Novemeber 27, 2018].

[11] L. Meier, D. Honegger, and M. Pollefeys, PX4: a node-based multithreaded open source robotics framework for deeply embedded platforms, in the *2015 IEEE Conference on Robotics and Automation (ICRA), May 26-30, 2015, Seattle, WA*. IEEE, 2015.

[12] "MAVROS", *ROS.org*, March 3, 2018. [Online]. Available: http://wiki.ros.org /mavros. [Accessed: November 27, 2018].

[13] "MAVLink Micro Air Vehicle Communication Protocol," *qgroundcontrol.com*. [Online]. Available: http://qgroundcontrol.org/mavlink/start. [Accessed: November 27, 2018].

[14] Wanninger, Lambert. "Introduction to Network RTK". [Online]. Available: http://www.wasoft.de/e/iagwg451/intro/introduction.html. IAG Working Group 4.5.1. [Accessed: November 27, 2018].

[15] Pulli, Kari; Baksheev, Anatoly; Kornyakov, Kirill; Eruhimov, Victor (1 April 2012). "Realtime Computer Vision with OpenCV". Queue. pp. 40:4040:56. doi:10.1145/2181796.2206309.

[16] Rosebrock, Adrian. "Ball Tracking with OpenCV". [Online]. Available: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/. [Accessed: November 27, 2018].