ESTIMATING CHARACTERISTICS OF A MANEUVERING REENTRY VEHICLE OBSERVED BY MULTIPLE SENSORS

THESIS

Evan M. Brooks

AFIT/GA/ENY/10-M02

AFIT/GA/ENY/10-M02

ESTIMATING CHARACTERISTICS OF A MANEUVERING REENTRY VEHICLE OBSERVED BY MULTIPLE SENSORS

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Astronautical Engineering

Evan M. Brooks, BS

March 2010

AFIT/GA/ENY/10-M02

ESTIMATING CHARACTERISTICS OF A MANEUVERING REENTRY VEHICLE OBSERVED BY MULTIPLE SENSORS

Evan M. Brooks, BS

Approved:

Dr. Richard Cobb (Chairman)	Date
Lt. Col. Frederick Harmon, USAF (Member)	Date
Lt. Col. Eric Swenson, USAF (Member)	Date

AFIT/GA/ENY/10-M02

Abstract

Post flight analysis of ballistic missile reentry vehicles is an area of focus for the U.S. Government, especially for those involved in ballistic missile defense. Typically, this analysis incorporates either a model-driven least squares filter or a data-following Kalman filter. The research performed here developed a filter that attempts to integrate the strengths of both filters. A least squares filter operates on observation data collected during exoatmospheric free flight and a Kalman filter is used to analyze data collected lower in the atmosphere, where potential maneuvers could be performed. Additionally, the filter was written to incorporate data from multiple sensors.

Using this hybrid filter, different scenarios are investigated to determine the potential benefits of adding additional collectors, increasing the data rate of collecting sensors, and investigating the effects of different collector geometry on the accuracy of results.

Results show that the filter successfully transitions from the least squares to Kalman filter, using the final values of the free flight propagation for the Kalman filter's initial state. Using this filter to investigate different collection scenarios, it was determined that the best results are achieved when multiple collectors are used, the data collection rate of the collectors is increased, and collectors are positioned perpendicular to the reentry vehicle heading.

Acknowledgments

I would like to express my sincere appreciation to my faculty advisor, Dr. Richard Cobb, for his guidance and support throughout the course of this thesis effort. The insight and experience was certainly appreciated. I would, also, like to thank my friend and coworker, Paul Precoda, who convinced me to add a smoother pass to my code and showed me the light at the end of the tunnel. Finally, without the love and support of my wife I may never have made it to this point.

Evan M. Brooks

Table of Contents

Page

Abstract....iv

Table of Contents....vi

List of Figures....viii

I. Introduction....1

Motivation....1

Background....2

Research Focus and Problem Statement....2

Methodology....3

Assumptions/Limitations....4

Preview....5

II. Literature Review....6

Chapter Overview....6

Relevant Research....6

Summary....8

III. Methodology....9

Chapter Overview....9

Filter Phases....9

Phase 0 – The Initial Guess.....*9*

Phase 1 – Free Flight Batch Least Squares Filter.....*11*

Phase 2a – Least Squares Sliding Window Discontinuous Filter.....*19*

Phase 2b – Kalman Filter Forward Pass.....*21*

Phase 2c – Backward Smoother Pass.....*29*

Phase 3 – Wrap-up and parameter computation. *30*

Summary 31

IV. Analysis and Results 32

Chapter Overview 32

Results of Simulation Scenarios 32

Case 1 – Filter Performance. *32*

Case 2 – Filter Performance Improvements Through Additional Sensors. *44*

Case 3 – Filter Performance Improvements Through Increased Data Rate. *52*

Case 4 – Filter Performance Improvements Through Collection Geometry. *61*

Investigative Questions Answered 68

Summary 69

V. Conclusions and Recommendations 70

Chapter Overview 70

Conclusions of Research 70

Significance of Research 71

Recommendations for Action 71

Recommendations for Future Research 72

Summary 73

Appendix 74

Coordinate Frames 74

kaliper.m 77

read_data.m 103

run_kaliper_cases.m 105

Bibliography 110

List of Figures

Page

Figure 1 – Nonlinear least squares flowchart 12

Figure 2 – Least squares sliding window size 20

Figure 3 – The Kalman filter flowchart 21

Figure 4 – Single sensor collection geometry 33

Figure 5 – Single sensor observation residuals 34

Figure 6 – Trajectory results with primary sensor 36

Figure 7 – Estimated ballistic coefficient 37

Figure 8 – Effects of smoother pass on estimates 38

Figure 9 – Ballistic coefficient estimated with and without smoother pass. 39

Figure 10 – Trajectory results from scenario 2, maneuvering target 41

Figure 11 – Trajectory results from scenario 3, maneuvering target 43

Figure 12 – Additional sensor collection geometries 45

Figure 13 – Trajectory results with addition of sensor 1 47

Figure 14 – Trajectory results with addition of sensor 2 49

Figure 15 – Trajectory results with addition of sensor 3 51

Figure 16 – Trajectory results with primary sensor at increased data rate 54

Figure 17 – Trajectory results with addition of sensor 1 at increased rate 56

Figure 18 – Trajectory results with addition of sensor 2 at increased rate 58

Figure 19 – Trajectory results with addition of sensor 3 at increased rate 60

Figure 20 – Trajectory results from sensor 1 63

Figure 21 – Trajectory results from sensor 2 65

Figure 22 – Trajectory results from sensor 3 67

Figure 23 – Earth-Centered Inertial (ECI) coordinate frame...74

Figure 24 – Earth-Centered Earth-Fixed (ECEF) coordinate frame 75

Figure 25 – Latitude/Longitude coordinate frame 75

Figure 26 – South, East, Up (SEZ) coordinate frame 76

List of Tables

Page

Table 1. Random noise applied to simulated observations..5

Table 2. Trajectory error values with primary sensor 36

Table 3. Reentry scenarios...39

Table 4. Trajectory errors from scenario 2, maneuvering target 41

Table 5. Trajectory errors from scenario 3, maneuvering target 43

Table 6. Trajectory errors due to LSSW..44

Table 7. Observing sensors...44

Table 8. Combined trajectory errors with addition of sensor 1 47

Table 9. Combined trajectory errors with addition of sensor 2 49

Table 10. Combined trajectory errors with addition of sensor 3 51

Table 11. Trajectory errors with primary sensor at increased data rate 54

Table 12. Trajectory errors with addition of sensor 1 at increased rate 56

Table 13. Trajectory errors with addition of sensor 2 at increased rate 58

Table 14. Trajectory errors with addition of sensor 3 at increased rate 60

Table 15. Trajectory errors from sensor 1 63

Table 16. Trajectory errors from sensor 2 65

Table 17. Trajectory errors from sensor 3 67

ESTIMATING CHARACTERISTICS OF A MANEUVERING REENTRY VEHICLE OBSERVED BY MULTIPLE SENSORS

I. Introduction

Motivation

Ballistic missiles are among the most advanced technology being currently developed for the purpose of conducting war. From the relatively small scale missiles in the arsenals of India and Pakistan to the massive intercontinental ballistic missiles (ICBMs) whose silos and mobile launchers dot the remote landscapes of Russia and the United States, ballistic missile technology exists today on a massive scale and will be a major component in future conflicts. Additionally, the proliferation of missile technology to countries that lack the technical manufacturing expertise necessary to produce ballistic missiles themselves is a reality. For these reasons, the understanding of the operational capabilities of ballistic missiles and their reentry vehicles (RVs) is currently one of our nation's highest priorities [14].

To meet these priorities, the United States has deployed a wide range of sensor technologies throughout the world. Combining the data collected by these sensors into a coherent assessment of a missile system's capabilities has long been the mission of intelligence agencies. Almost always, this will require data analysis in the form of modeling and simulation to determine key characteristics of the missile's RV.

Background

The RV parameter most commonly estimated is the ballistic coefficient. The ballistic coefficient can have varying definitions but is generally a ratio between the RV's mass and the product of its coefficient of drag and wetted surface area of the form [9]

$$\beta = \frac{m}{C_D S} \tag{1}$$

where

β	= *ballistic coefficient*
m	= *mass*
C_D	= *coefficient of drag*
S	= *wetted surface area*

In this form, an RV-like object with a large mass to surface area ratio will be referred to as a high-beta object ($5000 \lesssim \beta \lesssim 15{,}000 \, \frac{kg}{m^2}$); however, some formulations can define the term to be the inverse of this representation leading to RV-like objects being characterized by ballistic coefficients that are positive fractions much less than one. In the intelligence community, the formulation with mass in the numerator, as in Equation (1), is most commonly used and will be adopted for this thesis.

Research Focus and Problem Statement

While mass and wetted area can largely be considered constant for an RV flying a purely ballistic trajectory, the coefficient of drag cannot be. Changes in velocity and atmospheric density will lead to variations in the coefficient of drag, and thus, the ballistic coefficient. Estimating the characteristic profile of the ballistic coefficient as it changes throughout the reentry for non-maneuvering RVs is one of the goals of the data analysis.

For maneuvering RVs, the ability to determine the accelerations that deviate from a purely ballistic reentry becomes the focus of analysis. This thesis will investigate methods of characterizing the magnitude and direction of these sensed accelerations.

A further challenge in the estimation problem is combining observation data from multiple sensors that collect data on the same target. The analytical problem that is most commonly investigated involves the exploitation of single-source radar collections of azimuth, elevation, range, and potentially range rate. As infrared sensors, or other sensors that lack the ability to determine range, increasingly monitor reentries, incorporating data from this sensor type into the estimation problem is an additional goal of this thesis. In summary, the goal of this thesis is:

Estimate characteristics of reentry vehicles, including the ballistic coefficient and non-gravitational accelerations, using observation data collected by multiple sensors.

Methodology

Data analyses of reentering objects typically employ either the method of least squares [1,6,7] or a Kalman filter [3,8,9]. The decision to use one of these filters over the other is usually determined by the ability to model the vehicle's dynamics in a predetermined model. In a situation where a vehicle is only acted on by the forces of gravity or by constant sensed accelerations, a least squares filter with a model of these dynamics will rapidly determine the best solution [1,6]. Alternatively, a target experiencing unpredictable accelerations that could vary throughout its flight is better filtered by a Kalman filter which can adapt to changing dynamics [3,8,9].

In this thesis, an attempt will be made to merge these two filters into a single algorithm. During periods of purely ballistic flight, the accurate and fast solution of the least squares filter will be utilized. When the target has descended lower into the atmosphere, where there is a potential for maneuvering, the adaptable Kalman filter will be incorporated to estimate non-gravitational accelerations. Through the marriage of these two methods, the strengths of both filters will be exercised.

Assumptions/Limitations

The preliminary implementation of this algorithm will assume separate zones between the regions of purely ballistic flight and potential maneuverability at a predetermined altitude. However, this assumption could be invalidated by the firing of thrusters or an upper stage rocket engine that could be included in the collected data, and thus a future version of this algorithm could allow for a more detailed breakdown of regions of ballistic and maneuvering flight to compensate for this. Potentially, allowing the Kalman filtering portion of the filter to activate during these regions could handle any such maneuvers.

Alternatively, there could be situations where a vehicle’s performance in the lower atmosphere is well known and could be modeled. By employing the Kalman filter during these segments instead of a model driven least squares filter, some accuracy could potentially be sacrificed. This circumstance is unlikely due to the unpredictable nature of reentry vehicles and would be unlikely to arise.

For all cases investigated, sensors measuring range, azimuth, and elevation (RAE) or azimuth and elevation (AE) were incorporated. Random noise was applied to simulated observation data according to Table 1. These values were specified in an attempt to be representative of these categories of sensors without being specific to any actual sensor in the real world. Having a priori knowledge of these values is assumed.

Table 1. Random noise applied to simulated observations.

Standard Deviation	RAE Sensor	AE Sensor
σ_{RANGE}	2 m	--
$\sigma_{AZIMUTH}$	.02 degrees	.03 degrees
$\sigma_{ELEVATION}$	.015 degrees	.03 degrees

Preview

In the following pages, Chapter II will review the published literature covering previous research on this topic, Chapter III will provide a detailed methodology for the data filter's operation, Chapter IV discusses the results of this methodology, and Chapter V provides conclusions and offers recommendations for future work. After the main body of the paper, the appendix includes MATLAB code written as a part of this research.

II. Literature Review

Chapter Overview

The purpose of this chapter is to provide an overview of relevant research that has been conducted in the past ten years pertaining to this topic.

Relevant Research

The research into nonlinear estimation has a long history, arguably dating back well before the time of Gauss's development of least squares to the methods of averaging. The advances in the field have largely mirrored the advances in computing technology, as described in *Nonlinear Filters: Beyond the Kalman Filter* [3]. Gauss developed a method to make a single pass through all the available observations at once. With the advances in the computing technology, engineers increasingly turned to sequential filters, such as the unscented Kalman filter, to perform real-time estimation. As advances in computing progress, even more exotic methods have begun to gain in popularity, such as the particle filter. The particle filter is based off of Monte Carlo sampling being used to investigate the state space [3,15].

Some research has been performed at the Air Force Institute of Technology (AFIT) by Holmes [6] and Bittle [1] into the parameter identification of reentry vehicles. The research of Holmes and Bittle primarily focused on identifying characteristics of reentry vehicles that were in either flying purely ballistic or performing a constant maneuver which could be determined from the vehicle's bank angle [1,6]. Both of these problems lend themselves to the batch processing of the method of least squares, but would not be able to handle the issue of a ballistic coefficient changing as a function of velocity and altitude. In order to account for these uncertain dynamics, this thesis will move beyond the batch processing of the method of least squares into a sequential filter, like the Kalman filter. As will be shown, to maintain the maximum accuracy the solutions of both a Kalman filter and a nonlinear least squares filter will be combined into a hybrid solution.

Some work that has been done on hybrid filters, or modified filters [7,9] mostly focused on the problem of RV interception by an Anti-Ballistic Missile (ABM). Jackson and Farbman developed an interesting application of the least squares method where data is processed in small batches instead of as one large data set. This method allowed the filter to respond to changing dynamics. Jackson and Farbman's approach could be of applicability in this problem, but they relied heavily on curve fitting data without regard to continuity of dynamics between adjacent states. Also, accelerations were modeled as unknowns in three directions, so further processing would need to take place to determine the sources of accelerations and what portion of the entire acceleration was due to drag.

Lee and Liu adopted a more hybrid approach, combining a least squares and Kalman filters [9]. Lee and Liu recognized that a Kalman filter is better suited to a target with large changes in dynamics, but that when such a filter is applied to better behaved vehicles (slowly changing EOMs) results can degrade greatly. To counter this behavior, Lee and Liu ran a Kalman filter with a basic state vector through portions of the flight identified as being nearly ballistic by the companion least squares filter and then switched modes to a more dynamic state vector for the Kalman filter when the companion least squares filter identifies potential non-ballistic behavior. This thesis will attempt to utilize the methods of Jackson and Farbman to estimate an initial guess for the state and further refine this estimate using a Kalman filter similar to that derived by Lee and Liu.

Summary

A lot of research is being done on new methods of data filtering, including particle filters and other exotic filtering techniques. This thesis will investigate a combination of the more classical least squares and Kalman filters to take advantage of their individual strengths. This method is different than what was developed by Lee and Liu, where the companion least squares filter was only used to modify the makeup of the Kalman filter's state vector. A method of performing least squares filtering on a sliding window of non-ballistic flight will be explored to provide an initial guess at the state for the Kalman filter to include in its computations. This least squares sliding window method was pulled from the literature, where it was used to generate the final estimate, rather than acting as preprocessing for another filter. Furthermore, whereas all of the reviewed literature assumed a single collector and sometimes simplified observation models, the algorithm developed here accommodates an unlimited number of sensors and a variety of data types.

III. Methodology

Chapter Overview

The purpose of this chapter is to overview the specific methods and decisions that were implemented for the filters included in this thesis. A breakdown of the different filtering phases as well as assumptions and engineering decisions will be described.

Filter Phases

Phase 0 – The Initial Guess.

The phase of flight where the target is exoatmospheric is analyzed by a batch least squares filter fitting the observations to an oblate Earth gravity model. This phase is implemented for all observations that are determined to be at an altitude greater than 120 km. The 120 km value is used as a common number with METAL* for consistency. If the data does not include any observations above this cutoff altitude, the algorithm will instead consider the highest altitude observations. In this case, the algorithm will select 10% of the entire data set that occurs at the highest altitudes to fit a state vector to.

With the free flight data identified, the initial hurdle for the least squares filter is determining an initial guess to feed the batch filter. For this thesis, an initial state was computed using Equation (1) [7], the same equation that will be used for the least squares sliding window. For this application, this portion of data is taken as the window of interest and a single computation is done. The equation describing the state's

* The Mathematical and Engineering Trajectory Analysis Library is a library of mostly MATLAB functions that was developed at the National Air and Space Intelligence Center that contains tools to accomplish common trajectory analysis tasks, such as coordinate frame conversions, state vector propagation, and data manipulation. [2]

propagation with constant snap (fourth-derivative of position) is presented in Equation (2) [7].

$$(x)_{meas} = (x_n) + \left({x_n}^{(1)}\right)\Delta t + \left({x_n}^{(2)}\right)\frac{1}{2}\Delta t^2 + \left({x_n}^{(3)}\right)\frac{1}{6}\Delta t^3 + \left({x_n}^{(4)}\right)\frac{1}{24}\Delta t^4 \qquad (2)$$

where

$(x)_{meas}$	= *observed positions*
(x_n)	= *initial state to be computed*
${x_n}^{(i)}$	= *ith inertial derivative of the state with respect to time*
Δt	= *time difference between observations and initial state*

Equation (2) can be used to separately solve for the X, Y, and Z components of position and velocity for the initial state estimate. The procedure for computing the initial X position and velocity is now described, with the understanding that the procedure is the same for the Y and Z components.

In order to compute the position and velocity components of the state, a system of equations of the form $A\vec{x} = \vec{b}$ is desired so that the least squares solution can be directly solved for. In this application, the A matrix has as many rows as observation points and five columns matching the five states defined in Equation (2). The A matrix defines the coefficients of the derivatives defined by Equation (2) as shown below:

$$A = \begin{bmatrix} 1 & \Delta t_1 & \frac{1}{2}{\Delta t_1}^2 & \frac{1}{6}{\Delta t_1}^3 & \frac{1}{24}{\Delta t_1}^4 \\ 1 & \Delta t_2 & \frac{1}{2}{\Delta t_2}^2 & \frac{1}{6}{\Delta t_2}^3 & \frac{1}{24}{\Delta t_2}^4 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \qquad (3)$$

The $\vec{b}$ vector is constructed as the X position components of observations that had been previously identified as being exoatmospheric transformed to an ECI reference frame. The $\vec{b}$ vector has as many rows as observations and is a single column. With this information defined, the system of equations is fully defined as:

$$\begin{bmatrix} 1 & \Delta t_1 & \frac{1}{2}\Delta t_1{}^2 & \frac{1}{6}\Delta t_1{}^3 & \frac{1}{24}\Delta t_1{}^4 \\ 1 & \Delta t_2 & \frac{1}{2}\Delta t_2{}^2 & \frac{1}{6}\Delta t_2{}^3 & \frac{1}{24}\Delta t_2{}^4 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} x_n \\ x_n{}^{(1)} \\ x_n{}^{(2)} \\ x_n{}^{(3)} \\ x_n{}^{(4)} \end{bmatrix} = \begin{bmatrix} x_{meas\ 1} \\ x_{meas\ 2} \\ \vdots \end{bmatrix} \quad (4)$$

The ideal solution to this system, when A is square and nonsingular, is $\vec{x} = A^{-1}\vec{b}$ however this thesis will take advantage of the Singular Value Decomposition (SVD) to perform the inversion, which will have the advantage of computing the minimum norm solution when the A matrix is not invertible [11]. This calculation is done easily with MATLAB using the `pinv()` command to compute the pseudoinverse of A. Since the pseudoinverse equals the inverse in the case where A is invertible, this method is appropriate for all A.

Once this procedure has been performed on the X, Y, and Z elements of the transformed observations, the initial guess of the state is taken as the position and velocity components of the individual solutions. With this initial guess, the batch least squares filter is triggered.

Phase 1 – Free Flight Batch Least Squares Filter.

The batch least squares filter operates iteratively, improving upon the solution until further computation cannot achieve a better result. The least squares filter relies on the computation of partial derivatives to compute state updates. In the classical implementation of the filter, these derivatives are derived analytically [13], but with today's computers these derivatives can easily be estimated numerically using a finite difference method. This thesis will implement numerical partial derivatives for computing the state updates in the same manner as does portions of METAL[2]. With this decision, the least squares algorithm will proceed as described in Figure 1 [13].

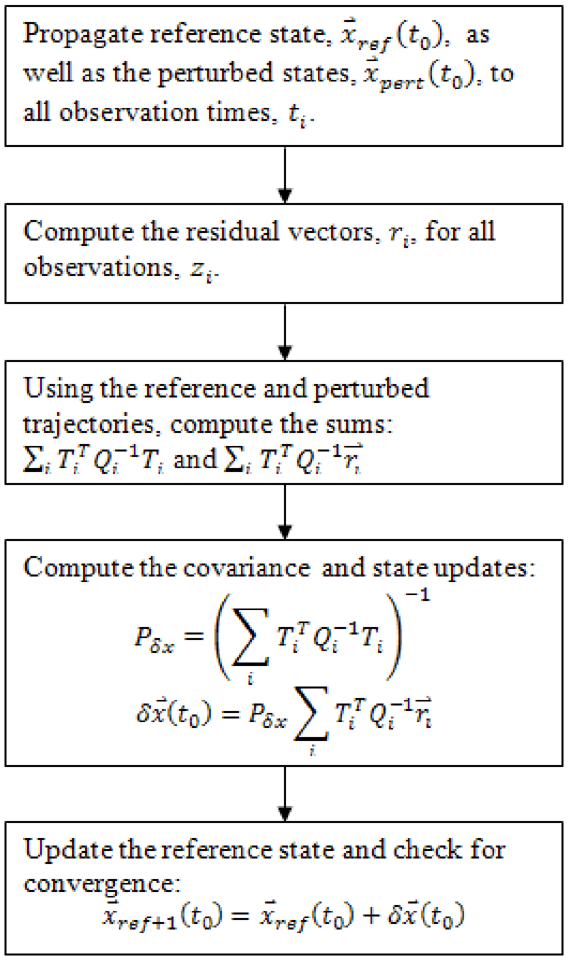

Propagate reference state, $\vec{x}_{ref}(t_0)$, as well as the perturbed states, $\vec{x}_{pert}(t_0)$, to all observation times, t_i.
Compute the residual vectors, r_i, for all observations, z_i.
Using the reference and perturbed trajectories, compute the sums: $\sum_i T_i^T Q_i^{-1} T_i$ and $\sum_i T_i^T Q_i^{-1} \vec{r}_i$
Compute the covariance and state updates: $P_{\delta x} = \left(\sum_i T_i^T Q_i^{-1} T_i\right)^{-1}$ $\delta\vec{x}(t_0) = P_{\delta x} \sum_i T_i^T Q_i^{-1} \vec{r}_i$
Update the reference state and check for convergence: $\vec{x}_{ref+1}(t_0) = \vec{x}_{ref}(t_0) + \delta\vec{x}(t_0)$

Figure 1 – Nonlinear least squares flowchart

In MATLAB, the current guess at the initial state is taken as $\vec{x}_{ref}(t_0)$. The elements of the state used throughout the algorithm are:

$$\vec{x} = \begin{bmatrix} x_{ECI} \\ y_{ECI} \\ z_{ECI} \\ \dot{x}_{ECI} \\ \dot{y}_{ECI} \\ \dot{z}_{ECI} \\ \ddot{x}_{S_{ECI}} \\ \ddot{y}_{S_{ECI}} \\ \ddot{z}_{S_{ECI}} \end{bmatrix} \tag{5}$$

where

x_{ECI}	= *Earth-Centered Inertial X Position*
y_{ECI}	= *Earth-Centered Inertial Y Position*
z_{ECI}	= *Earth-Centered Inertial Z Position*
$\dot{x}_{ECI}$	= *Earth-Centered Inertial X Velocity*
$\dot{y}_{ECI}$	= *Earth-Centered Inertial Y Velocity*
$\dot{z}_{ECI}$	= *Earth-Centered Inertial Z Velocity*
$\ddot{x}_{S_{ECI}}$	= *Earth-Centered Inertial X Sensed Acceleration*
$\ddot{y}_{S_{ECI}}$	= *Earth-Centered Inertial Y Sensed Acceleration*
$\ddot{z}_{S_{ECI}}$	= *Earth-Centered Inertial Z Sensed Acceleration*

Note that the accelerations in Equation (5) are sensed accelerations. Sensed accelerations differ from the total accelerations in that they do not incorporate gravitational acceleration. Sensed accelerations account for accelerations that are the result of other body forces that would result in a maneuver away from a ballistic trajectory.

This state is then perturbed individually in the X, Y, and Z positions and velocities to create six perturbed states denoted $\vec{x}_{pert}(t_0)$. The magnitudes of these perturbations are arbitrarily chosen. For this thesis, perturbations of 1 m in position and 1 $\frac{m}{sec}$ in velocity are implemented as arbitrary values.

With the initial state and perturbed states defined, they are then propagated to all observation times. This is accomplished by basic equations of motion (EOM) defined in Equations (6-11).

$$\frac{d}{dt}x = \dot{x} \tag{6}$$

$$\frac{d}{dt}y = \dot{y} \tag{7}$$

$$\frac{d}{dt}z = \dot{z} \tag{8}$$

$$\frac{d}{dt}\dot{x} = \ddot{x} = g_x \tag{9}$$

$$\frac{d}{dt}\dot{y} = \ddot{y} = g_y \tag{10}$$

$$\frac{d}{dt}\dot{z} = \ddot{z} = g_z \tag{11}$$

where

g_x, g_y, g_z = *x,y,and z ECI components of gravitational acceleration*

During this phase of flight, it is assumed that there is little to no atmosphere and therefore there are no external body forces and hence no sensed accelerations. The only changes in velocity are due to the gravitational acceleration. With the current state estimate and its perturbations propagated, the next step is the computation of the data residuals.

Computing data residuals is merely an exercise in reference frame transformations. These functions have been fully implemented in the METAL library, and were used in this research [2]. The propagated states are transformed from Earth-Centered Inertial (ECI) frame to a sensor-specific Range, Azimuth, and Elevation (RAE) frame. With this transformation complete, the residuals are calculated from Equation (12).

$$\vec{r} = \vec{z} - \vec{x}_{ref\,RAE} \tag{12}$$

where

$\vec{r}$	= *matrix of residuals*
$\vec{z}$	= *matrix of observations*
$\vec{x}_{ref\,RAE}$	= *propagated reference state in RAE frame*

With the residuals computed, statistical editing of outliers can be performed. With a priori knowledge of a sensor's performance[†], residuals outside of an arbitrary number of standard deviations can be removed. The algorithm produced for this thesis allows for the number of standard deviations to be input by the user, with the default settings deleting residuals more than three standard deviations from the computed reference trajectory.

With any deleting complete, the residuals matrix must be reshaped into a column vector to be used in the sum $\sum_i T_i^T Q_i^{-1} \vec{r}_i$. Note that this action is easily accomplished in MATLAB with the `reshape()` function.

[†] See Table 1 for error values used in the simulated observation data generated for this research.

The Q_i matrix required for the two sums is constructed from the a priori knowledge of random errors in the sensor observations. The matrix is diagonal and of the form:

$$Q = \begin{bmatrix} \frac{1}{{\sigma_R}^2} & 0 & 0 & 0 & 0 & \dots \\ 0 & \frac{1}{{\sigma_A}^2} & 0 & 0 & 0 & \dots \\ 0 & 0 & \frac{1}{{\sigma_E}^2} & 0 & 0 & \dots \\ 0 & 0 & 0 & \frac{1}{{\sigma_R}^2} & 0 & \dots \\ 0 & 0 & 0 & 0 & \frac{1}{{\sigma_A}^2} & \ddots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \end{bmatrix} \tag{13}$$

where

Q	= *covariance matrix*
σ_R	= *sensor standard deviation in range measurements*
σ_A	= *sensor standard deviation in azimuth measurements*
σ_E	= *sensor standard deviation in elevation measurements*

The Q matrix in Equation (13) has three of the standard deviations repeated enough times to make the matrix dimensions equal to the number of observations multiplied by the number of observation data types, three in the case of a sensor with range, azimuth, and elevation measurements. For sensors lacking a range component in the measurement, the range standard deviation is omitted and the Q matrix is reduced in size.

The last element of the sum $\sum_i T_i^T Q_i^{-1} \vec{r}_i$ needed is T, the observation matrix, computed with the perturbed trajectories. This matrix is defined as the propagation of the partials of the observation relationships with respect to the different state vector components, as shown numerically in Equation (14).

$$T = \tag{14}$$

$$\begin{bmatrix} \frac{R_{ref\,1} - R_{pert-x\,1}}{x_{pert}} & \frac{R_{ref\,1} - R_{pert-y\,1}}{y_{pert}} & \frac{R_{ref\,1} - R_{pert-z\,1}}{z_{pert}} & \frac{R_{ref\,1} - R_{pert-\dot{x}\,1}}{\dot{x}_{pert}} & \frac{R_{ref\,1} - R_{pert-\dot{y}\,1}}{\dot{y}_{pert}} & \frac{R_{ref\,1} - R_{pert-\dot{z}\,1}}{\dot{z}_{pert}} \\ \frac{A_{ref\,1} - A_{pert-x\,1}}{x_{pert}} & \frac{A_{ref\,1} - A_{pert-y\,1}}{y_{pert}} & \frac{A_{ref\,1} - A_{pert-z\,1}}{z_{pert}} & \frac{A_{ref\,1} - A_{pert-\dot{x}\,1}}{\dot{x}_{pert}} & \frac{A_{ref\,1} - A_{pert-\dot{y}\,1}}{\dot{y}_{pert}} & \frac{A_{ref\,1} - A_{pert-\dot{z}\,1}}{\dot{z}_{pert}} \\ \frac{E_{ref\,1} - E_{pert-x\,1}}{x_{pert}} & \frac{E_{ref\,1} - E_{pert-y\,1}}{y_{pert}} & \frac{E_{ref\,1} - E_{pert-z\,1}}{z_{pert}} & \frac{E_{ref\,1} - E_{pert-\dot{x}\,1}}{\dot{x}_{pert}} & \frac{E_{ref\,1} - E_{pert-\dot{y}\,1}}{\dot{y}_{pert}} & \frac{E_{ref\,1} - E_{pert-\dot{z}\,1}}{\dot{z}_{pert}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

where

T	= *observation matrix*
$R_{ref\,1}$	= *first range value of the propagated reference trajectory*
$A_{ref\,1}$	= *first azimuth value of the propagated reference trajectory*
$E_{ref\,1}$	= *first elevation value of the propagated reference trajectory*
$R_{pert-x\,1}$	= *first range value of the x-perturbed trajectory*
$A_{pert-x\,1}$	= *first azimuth value of the x-perturbed trajectory*
$E_{pert-x\,1}$	= *first elevation value of the x-perturbed trajectory*
x_{pert}	= *magnitude of perturbation to reference state in x direction*

With all elements of the sums $\sum_i T_i^T Q_i^{-1} T_i$ and $\sum_i T_i^T Q_i^{-1} \vec{r}_i$ now computed, these values are now incorporated into the sums and the updates are computed. From **Figure 1**, these updates are

$$P_{\delta x} = \left(\sum_i T_i^T Q_i^{-1} T_i\right)^{-1} \tag{15}$$

$$\delta\vec{x}(t_0) = P_{\delta x} \sum_i T_i^T Q^{-1} \vec{r_i} \tag{16}$$

where

$P_{\delta x}$	= *updated covariance matrix*
$\delta\vec{x}(t_0)$	= *update to the reference state*

The final step in the batch least squares phase is to check for convergence and update the reference state. The test for convergence can be accomplished in a couple of ways, either by checking if the update to the state vector lies within the updated covariance matrix uncertainty [13] or by checking to see if a defined cost function has stopped improving from one iteration to the next [2]. Both of these methods were investigated, and in the end it was decided that the cost function method typically led to better results as the covariance method would typically indicate convergence before a good fit had been achieved.

The cost function computed for this thesis takes the form:

$$cost = \sum_{SENSORS} \frac{\sqrt{\sum_{OBSERVATIONS} (residual)^2}}{\sigma_{SENSOR}} \tag{17}$$

where

$cost$	= *cost function used to test for convergence*
$residual$	= *observation residual from reference trajectory*
σ_{SENSOR}	= *standard deviation of sensor observation*

This cost function normalizes the data residuals by the sensor's standard deviations and gives proper weight to measurements that are more precise than others. The change in cost function between iterations is computed in each iteration after the first. Once this change drops below a value defined by the user, the filter is stopped. The default convergence criterion is a change of less than .01% of the cost function.

Once the free flight batch least squares filter has converged, the final state is passed to the next phase of the algorithm, where the sensed accelerations of the target will be considered variable, and need to be estimated.

Phase 2a – Least Squares Sliding Window Discontinuous Filter.

The next phase of the algorithm is divided into three subsections: 2a, 2b, and 2c, which together filter those observations deep enough into the atmosphere that the target can experience non-zero sensed accelerations. The first of these subsections in the least squares sliding window filter. This filter is based off of the paper by Jackson and Farbman, *Trajectory Reconstruction with a Least Squares Sliding Window (LSSW) Filter* [7]. This subsection is a rapid computation of an approximate state at every observation that contains a range, azimuth, and elevation measurement. This approximation can be used in the subsequent subsections of phase two as an initial guess of the state at those times.

This section relies on multiple solutions to Equation (2) over different portions of the observations. Taken as a whole, these solutions are the initial guesses used later. Other than computing solutions to Equation (2) multiple times, this portion of the algorithm must determine which portions (windows) of the observations to include in each run. The method used to determine this window size is similar to that performed by

Jackson and Farbman. The window size begins at a defined minimum value, grows to a nominal value that is used throughout the majority of the observations, and then grows again as the window approaches the end of the available data. Three values for minimum window size, window size, and maximum window size are optional user inputs with default values 5, 30, and 40, respectively. An example of a window size varying by the position in the observations is presented in Figure 2.

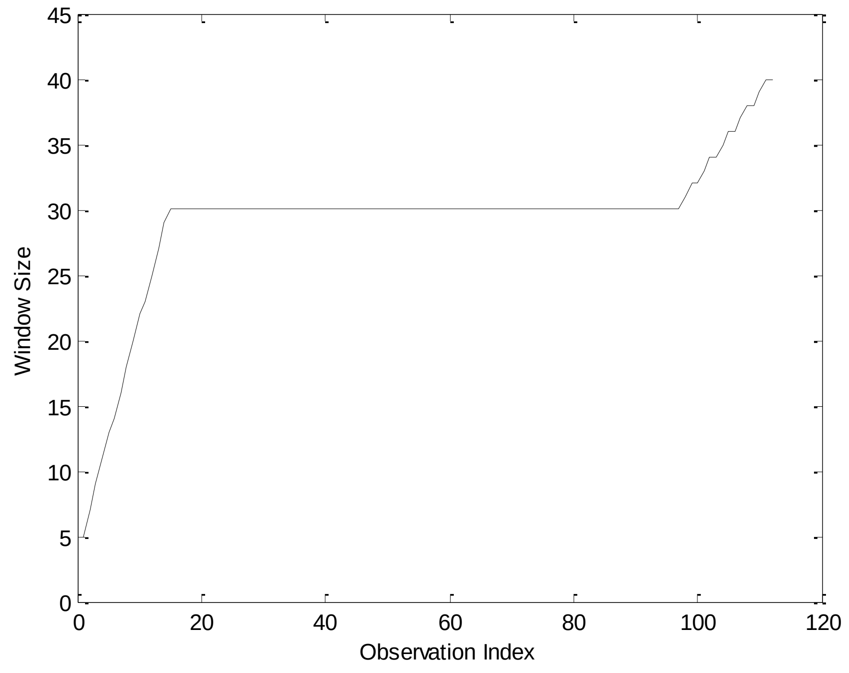


Figure 2 – Least squares sliding window size

After some trial, it was decided to limit the use of these results in later sections of phase two to just the position estimates. The velocity and acceleration estimates were found to have extreme noise, often returning estimates with mean and random errors large enough that it was unclear if there was any correlation to the truth values. With the positions estimates determined, the next subsection of phase two is commenced.

Phase 2b – Kalman Filter Forward Pass.

The main subsection of phase two runs a Kalman filter. The Kalman filter is executed in a manner described in **Figure 3** [13].

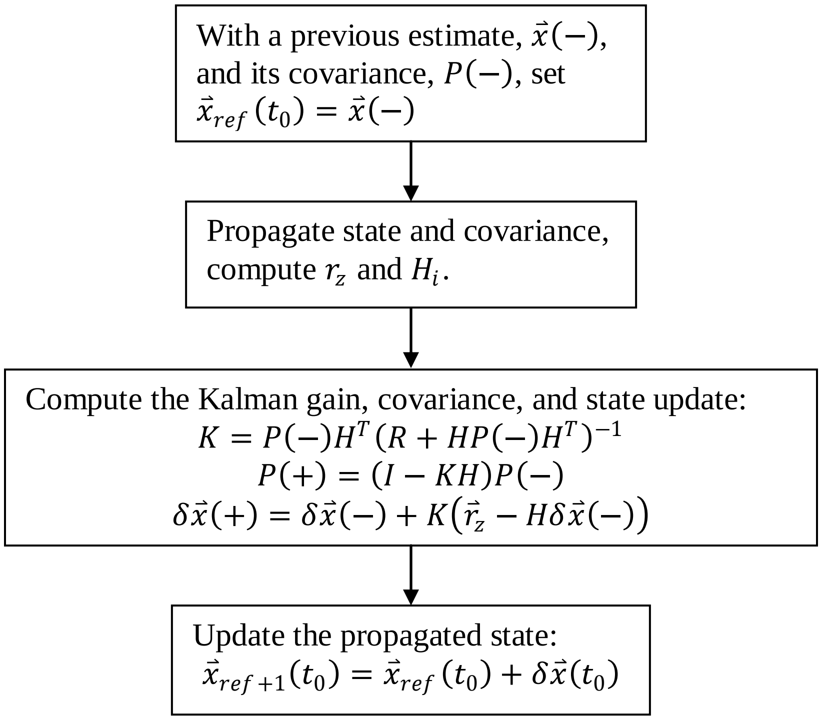

With a previous estimate, $\vec{x}(-)$, and its covariance, $P(-)$, set $\vec{x}_{ref}(t_0) = \vec{x}(-)$
Propagate state and covariance, compute r_z and H_i.
Compute the Kalman gain, covariance, and state update:
$K = P(-)H^T(R + HP(-)H^T)^{-1}$
$P(+) = (I - KH)P(-)$
$\delta\vec{x}(+) = \delta\vec{x}(-) + K\left(\vec{r}_z - H\delta\vec{x}(-)\right)$
Update the propagated state:
$\vec{x}_{ref+1}(t_0) = \vec{x}_{ref}(t_0) + \delta\vec{x}(t_0)$

Figure 3 – The Kalman filter flowchart

The state is propagated using easily defined equations of motion, rather than the state transition matrix. These equations of motion are

$$\frac{d}{dt}x = \dot{x} \tag{18}$$

$$\frac{d}{dt}y = \dot{y} \tag{19}$$

$$\frac{d}{dt}z = \dot{z} \tag{20}$$

$$\frac{d}{dt}\dot{x} = \ddot{x} = g_x + \ddot{x}_S \tag{21}$$

$$\frac{d}{dt}\dot{y} = \ddot{y} = g_y + \ddot{y}_S \tag{22}$$

$$\frac{d}{dt}\dot{z} = \ddot{z} = g_z + \ddot{z}_S \tag{23}$$

where

$\ddot{x}, \ddot{y}, \ddot{z}$	= *total accelerations in the x, y, and z ECI directions*
$\ddot{x}_S, \ddot{y}_S, \ddot{z}_S$	= *sensed accelerations (due to body forces other than gravity) in the x, y, and z ECI directions*

Equations (18-23) differ from Equations (6-11) that were used in the free flight batch least squares filter in that the sensed accelerations are allowed to be non-zero. This method of propagating the state is more accurate than the approximation given by the state transition matrix and is implemented for state propagation. The state transition matrix must still be computed, however, in order to propagate the state covariance. The state transition matrix is computed from

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} + F\,\Delta t \tag{24}$$

where

Φ = *state transition matrix*

F = *state distribution matrix*

Δt = *difference in time between adjacent observations*

The state distribution matrix, F, is the Jacobian of the state dynamics defined in Equations (18-23) where the gravitational acceleration is replaced with a simple approximation with respect to the elements of the state, defined in Equation (5).

$$\ddot{x} = \ddot{x}_S + g_x = \ddot{x}_S - \frac{\mu x}{(x^2+y^2+z^2)^{3/2}} \tag{25}$$

$$\ddot{y} = \ddot{y}_S + g_y = \ddot{y}_S - \frac{\mu y}{(x^2+y^2+z^2)^{3/2}} \tag{26}$$

$$\ddot{z} = \ddot{z}_S + g_z = \ddot{z}_S - \frac{\mu z}{(x^2+y^2+z^2)^{3/2}} \tag{27}$$

where

μ = *Earth's gravitational parameter* $\approx 3.986005 \cdot 10^5 \frac{km^3}{s^2}$

With these values substituted for the gravitational acceleration, the state distribution matrix is computed by Equation (28).

$$F = \tag{28}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -\mu R^{-3}(1-3x^2R^{-2}) & 3\mu xyR^{-5} & 3\mu xzR^{-5} & 0 & 0 & 0 & 1 & 0 & 0 \\ 3\mu xyR^{-5} & -\mu R^{-3}(1-3y^2R^{-2}) & 3\mu yzR^{-5} & 0 & 0 & 0 & 0 & 1 & 0 \\ 3\mu xzR^{-5} & 3\mu yzR^{-5} & -\mu R^{-3}(1-3z^2R^{-2}) & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where

R = $\sqrt{(x^2+y^2+z^2)}$ *for ease of computation*

With the state distribution matrix, F, computed, the state transition matrix, Φ, can be determined. The state transition matrix can then be used to propagate the covariance matrix to the current time with Equation (29).

$$P_{t_n}(-) = \Phi\, P_{t_{n-1}}(+)\, \Phi^{\mathrm{T}} + \mathrm{Q} \tag{29}$$

where

$P_{t_n}(-)$ = *initial covariance matrix at time* t_n

$P_{t_{n-1}}(+)$ = *updated covariance matrix at time* t_{n-1}

Q = *noise applied to covariance propagation*

A few different methods of computing Q, the covariance propagation noise, were investigated. In the end, the method employed by the Kinematics And Dynamics Reconstruction Environment (KADRE)[4] was implemented due to its incorporation of the time step. Due to its explicit dependency on the time step from one data point to the next, the filter was better behaved when compared to other methods, including a constant Q. The computation of Q was performed with Equation (30).

$$Q = \sum_{i=0}^{n} \sum_{j=0}^{m} \frac{1}{j!k!} F^i \, Q_0 \, (F^T)^j \, \Delta t^{1+i+j} \, \frac{1}{1+i+j} \tag{30}$$

where

F	= *state distribution matrix*
Q_0	= *covariance scaling matrix*
Δt	= *elapsed time from previous data point*

The scaling matrix, Q_0, was implemented as user-defined variables that form the diagonal matrix shown in Equation (31).

$$Q_0 = \begin{bmatrix} q_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_v & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_v & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_v & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q_a & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_a & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_a \end{bmatrix} \tag{31}$$

where

q_p	= *position covariance scaling constant*
q_v	= *velocity covariance scaling constant*
q_a	= *acceleration covariance scaling constant*

The covariance scaling constants q_p, q_v, and q_a are implemented with default values of 0, 0, and 0.001, respectively. These values were arbitrarily selected after some experimentation.

With the state and covariance propagated to the current data point, the next step involves the computation of the observation matrix, H, which is a linearization of the relationship between the state elements and the observation variables. This matrix can be used to convert both the covariance and state variables to the observation variables reference frame. As the transformation from ECI state components to RAE observation variables is easily handled with the METAL library, the exact transformation can be used. The observation matrix must still be used to transform the propagated covariance matrix. The observation matrix is computed from Equations (32-47).

$$H = \begin{bmatrix} \frac{\partial R}{\partial x} & \frac{\partial R}{\partial y} & \frac{\partial R}{\partial z} & \frac{\partial R}{\partial \dot{x}} & \frac{\partial R}{\partial \dot{y}} & \frac{\partial R}{\partial \dot{z}} & \frac{\partial R}{\partial \ddot{x}} & \frac{\partial R}{\partial \ddot{y}} & \frac{\partial R}{\partial \ddot{z}} \\ \frac{\partial Az}{\partial x} & \frac{\partial Az}{\partial y} & \frac{\partial Az}{\partial z} & \frac{\partial Az}{\partial \dot{x}} & \frac{\partial Az}{\partial \dot{y}} & \frac{\partial Az}{\partial \dot{z}} & \frac{\partial Az}{\partial \ddot{x}} & \frac{\partial Az}{\partial \ddot{y}} & \frac{\partial Az}{\partial \ddot{z}} \\ \frac{\partial El}{\partial x} & \frac{\partial El}{\partial y} & \frac{\partial El}{\partial z} & \frac{\partial El}{\partial \dot{x}} & \frac{\partial El}{\partial \dot{y}} & \frac{\partial El}{\partial \dot{z}} & \frac{\partial El}{\partial \ddot{x}} & \frac{\partial El}{\partial \ddot{y}} & \frac{\partial El}{\partial \ddot{z}} \end{bmatrix} \tag{32}$$

$$\frac{\partial R/Az/El}{\partial \dot{x}} = \frac{\partial R/Az/El}{\partial \dot{x}} = \frac{\partial R/Az/El}{\partial \dot{z}} = 0 \tag{33}$$

$$\frac{\partial R/Az/El}{\partial \ddot{x}} = \frac{\partial R/Az/El}{\partial \ddot{y}} = \frac{\partial R/Az/El}{\partial \ddot{z}} = 0 \tag{34}$$

$$\frac{\partial R}{\partial x} = \frac{x - x_0}{R} \tag{35}$$

$$\frac{\partial R}{\partial y} = \frac{y - y_0}{R} \tag{36}$$

$$\frac{\partial R}{\partial z} = \frac{z - z_0}{R} \tag{37}$$

$$\frac{\partial Az}{\partial x} = \frac{(-\sin(\theta)\cos(\omega t) - \cos(\theta)\sin(\omega t))g_{Az} - (-\sin(\varphi)\cos(\theta)\cos(\omega t) + \sin(\varphi)\sin(\theta)\sin(\omega t))f_{Az}}{\left(1 + \left(\frac{f_{Az}}{g_{Az}}\right)^2\right) g_{Az}{}^2} \tag{38}$$

$$\frac{\partial Az}{\partial y} = \frac{(-sin(\theta)sin(\omega t)+cos(\theta)cos(\omega t))g_{Az} - (-sin(\varphi)cos(\theta)sin(\omega t) - sin(\varphi)sin(\theta)cos(\omega t))f_{Az}}{\left(1+\left(\frac{f_{Az}}{g_{Az}}\right)^2\right)g_{Az}^2} \tag{39}$$

$$\frac{\partial Az}{\partial z} = \frac{-cos(\varphi)f_{Az}}{\left(1+\left(\frac{f_{Az}}{g_{Az}}\right)^2\right)g_{Az}^2} \tag{40}$$

$$f_{Az} = -sin(\theta)(cos(\omega t)x + sin(\omega t)y - X_{0_{ECEF}}) + cos(\theta)(-sin(\omega t)x + cos(\omega t)y - Y_{0_{ECEF}}) \tag{41}$$

$$g_{Az} = -sin(\varphi)cos(\theta)(cos(\omega t)x + sin(\omega t)y - X_{0_{ECEF}}) - sin(\varphi)cos(\theta)(-sin(\omega t)x + cos(\omega t)y - Y_{0_{ECEF}}) + cos(\varphi)(z - Z_{0_{ECEF}}) \tag{42}$$

$$\frac{\partial El}{\partial x} = \frac{(cos(\varphi)cos(\theta)cos(\omega t) - cos(\varphi)sin(\theta)sin(\omega t))g_{El} - (x - X_0)\frac{f_{El}}{g_{El}}}{\sqrt{\left(1-\left(\frac{f_{El}}{g_{El}}\right)^2\right)}\, g_{El}^2} \tag{43}$$

$$\frac{\partial El}{\partial y} = \frac{(cos(\varphi)cos(\theta)sin(\omega t) + cos(\varphi)sin(\theta)cos(\omega t))g_{El} - (y - Y_0)\frac{f_{El}}{g_{El}}}{\sqrt{\left(1-\left(\frac{f_{El}}{g_{El}}\right)^2\right)}\, g_{El}^2} \tag{44}$$

$$\frac{\partial El}{\partial z} = \frac{sin(\varphi)g_{El} - (z - Z_0)\frac{f_{El}}{g_{El}}}{\sqrt{\left(1-\left(\frac{f_{El}}{g_{El}}\right)^2\right)}\, g_{El}^2} \tag{45}$$

$$f_{El} = cos(\varphi)cos(\theta)(cos(\omega t)x + sin(\omega t)y - X_{0_{ECEF}}) + cos(\varphi)sin(\theta)(-sin(\omega t)x + cos(\omega t)y - Y_{0_{ECEF}}) + sin(\varphi)(z - Z_{0_{ECEF}}) \tag{46}$$

$$g_{El} = \sqrt{(x^2 + y^2 + z^2)} \tag{47}$$

where

φ	= *sensor geocentric latitude*
θ	= *sensor longitude*
ω	= *Earth angular velocity*
X_0, Y_0, Z_0	= *sensor ECI position at observation*
$X_{0_{ECEF}}, Y_{0_{ECEF}}, Z_{0_{ECEF}}$	= *sensor ECEF position*

This fully defines the H matrix for range, azimuth, and elevation observations. If the solution from the LSSW subsection is included in the Kalman filter, those partial derivatives must be included in H. Since the LSSW solution is just the state vector positions, those partials are easily added as shown in Equation (48).

$$H = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{48}$$

The final matrix that must be computed before determining the Kalman gain, *K*, is the measurement noise matrix, *R*. For a sensor collecting range, azimuth, and elevation data, the R matrix is constructed by Equation (49). If the LSSW solution is also included, its apparent noise is included as shown in Equation (50). The initial implementation of the algorithm uses a LSSW error standard deviation that is independent of direction. This could be modified in subsequent work.

$$R = \begin{bmatrix} \sigma_R & 0 & 0 \\ 0 & \sigma_{Az} & 0 \\ 0 & 0 & \sigma_{El} \end{bmatrix} \tag{49}$$

$$R = \begin{bmatrix} \sigma_R & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{Az} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{El} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{LSSW} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{LSSW} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{LSSW} \end{bmatrix} \tag{50}$$

where

R = *measurement noise matrix*

σ_R = *sensor range measurement standard deviation*

σ_{Az}	*= sensor azimuth measurement standard deviation*
σ_{El}	*= sensor elevation measurement standard deviation*
σ_{LSSW}	*= LSSW error standard deviation*

With $P_{t_n}(-)$, *H*, and *R*, the Kalman gain, *K*; the updated covariance, $P_{t_n}(+)$; and the state update, $\delta\vec{x}$, are computed from Equations (51-53), respectively.

$$K = P(-)H^T(R + HP(-)H^T)^{-1} \tag{51}$$

$$P_{t_n}(+) = (I - KH)P_{t_n}(-) \tag{52}$$

$$\delta\vec{x} = K(\vec{z} - H\vec{x}) \tag{53}$$

where

I	*= identity matrix*
$\vec{z}$	*= vector of observations*
$\vec{x}$	*= 9-element vector of the current state*

This completes the Kalman filter subsection. This series of equations is carried out at every data point until the end of the data set. As the computation takes place, a time history of the state and covariance are saved for use in the final subsection of phase two, the backward smoother.

Phase 2c – Backward Smoother Pass.

The final subsection of phase two involves a backward traveling smoother pass using the time history of results from the Kalman filter. This section of code was implemented in the manner described by the KADRE engineering description [4]. Starting at the final time, the filter consists of Equations (54-56).

$$C = P_{t_n}(+)\Phi^T P_{t_{n+1}}(-)^{-1} \tag{54}$$

$$P_{smoot\,h_{t_n}} = P_{t_n}(+) + C\left(P_{smoot\,h_{t_{n+1}}} - P_{t_{n+1}}(-)\right)C^T \tag{55}$$

$$\vec{x}_{smoot\,h_{t_n}} = \vec{x}_{t_n} + \left(C\left(\vec{x}_{smoot\,h_{t_{n+1}}}{}^T - \vec{x}_{t_{n+1}}(-)^T\right)\right)^T \tag{56}$$

where

C = *C matrix*

$\vec{x}_{t_{n+1}}(-)$ = *pre-update state vector from Kalman filter*

Once the smoother pass is complete, the filtering phases of the code are complete. Additional code is included to perform analysis on the results, including calculations of altitude, ballistic coefficient, and Mach number. These calculations makeup the final phase of the algorithm.

Phase 3 – Wrap-up and parameter computation.

The final phase of the algorithm performs calculations that can be useful for the analysis of results. With the complete time history of the state available, these calculations are performed rapidly and included in the output. Key among these parameters for this research is the ballistic coefficient. With the time history of sensed accelerations, the ballistic coefficient is computed from Equations (57) and (58).

$$V_a = \sqrt{(\dot{x} + \omega\, y)^2 + (\dot{y} - \omega\, x)^2 + \dot{z}^2} \qquad (57)$$

$$\beta = \frac{\frac{1}{2}\rho\, {V_a}^2}{\sqrt{{\ddot{x}_S}^2 + {\ddot{y}_S}^2 + {\ddot{z}_S}^2}} \qquad (58)$$

where

V_a = *Air relative velocity magnitude*

ω = *Earth's angular velocity about the pole*

ρ = *altitude dependent atmospheric density*

In addition to ballistic coefficient, parameters such as altitude, Mach number, and the position of the target in latitude and longitude are computed for easier analysis.

Summary

This chapter described the methodology used in the algorithm to filter observation data. The algorithm operates in two primary filtering phases with a third wrap-up phase. The first phase filters exoatmospheric observations with a batch least squares filter that assumes there are no sensed accelerations other than gravity. The second phase filters endoatmospheric observations with a least squares sliding window filter, a Kalman filter forward pass, and a backward running smoother. The final, wrap-up phase, calculates parameters that can be useful for further analysis.

IV. Analysis and Results

Chapter Overview

This chapter reviews the analysis that was performed with the filtering algorithm created following the methodology detailed in Chapter III. The primary goals of the analysis performed were to validate the filter's performance in reconstructing an observed reentering target, to investigate the benefits to accuracy of additional sensors observing the same target, to investigate the benefits to accuracy of an increased rate of data collection against targets, and to investigate the benefits to accuracy of different sensor collection geometries. In each case where the accuracy of the filter is to be investigated, the filter will be tested against both non-maneuvering and maneuvering targets.

Results of Simulation Scenarios

Case 1 – Filter Performance.

The initial goal in reviewing the filter results was to ensure that the filter was successfully filtering the collected data. To verify filter performance, the residual errors between the observed data and the reconstructed trajectory were analyzed to ensure that whenever possible they had a nearly zero mean error, an apparent random scattering in error about the mean error, and a low standard deviation in error.

In order to assess the performance of the algorithm, a test case was constructed with a single target reentering which is observed from an altitude of 800 km to an altitude near impact. The observing sensor collected range, azimuth, and elevation data at a rate of 2 Hz. An overview of the collection geometry is presented in Figure 4. The target is initially acquired by the sensor over the State of Maine and is tracked to its impact in the central United States.

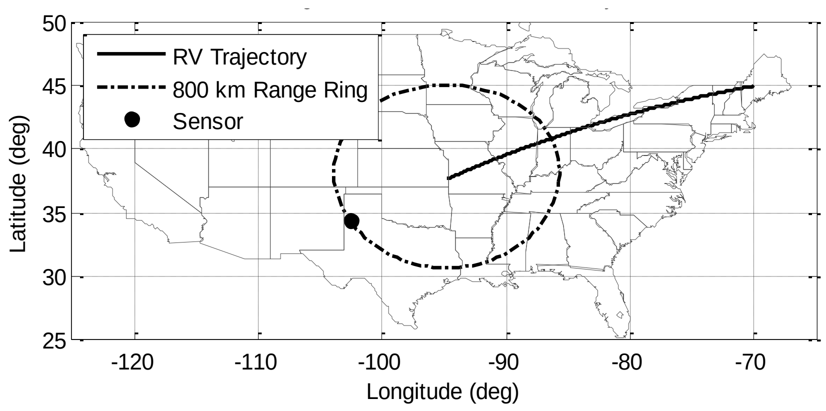


Figure 4 – Single sensor collection geometry

The residual errors between the observed data and the reconstructed state vectors are presented in Figure 5. The residuals shows the characteristics of near zero mean error, random scattering about the mean error, and a low standard deviation of error. For this case, the mean error was -0.0032 m in range, -0.00016 deg in azimuth, and 0.0037 deg in elevation. The residual scattering appears to be nearly random, the error standard deviation was 1.9 m in range, 0.016 deg in azimuth, and 0.015 deg in elevation.

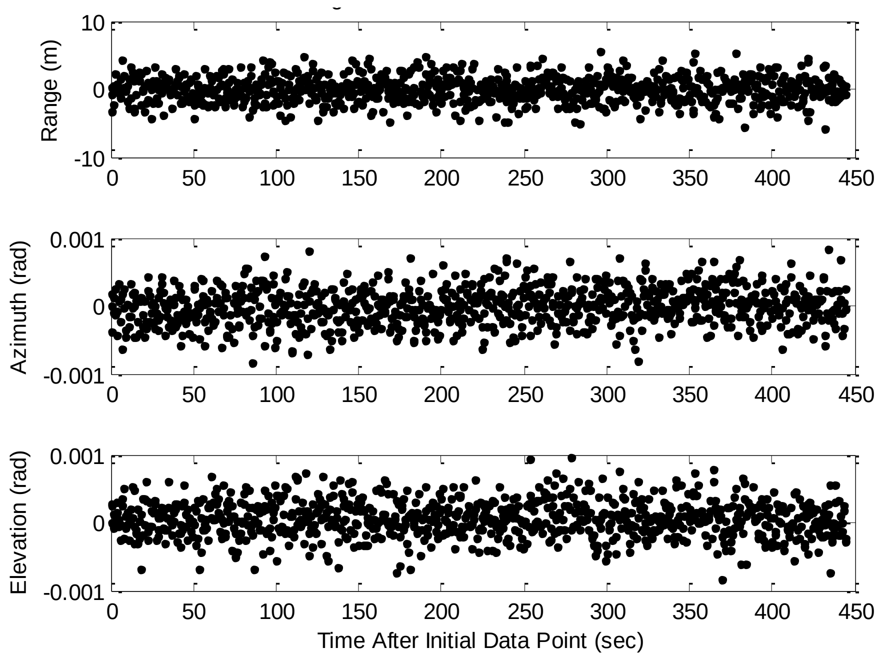
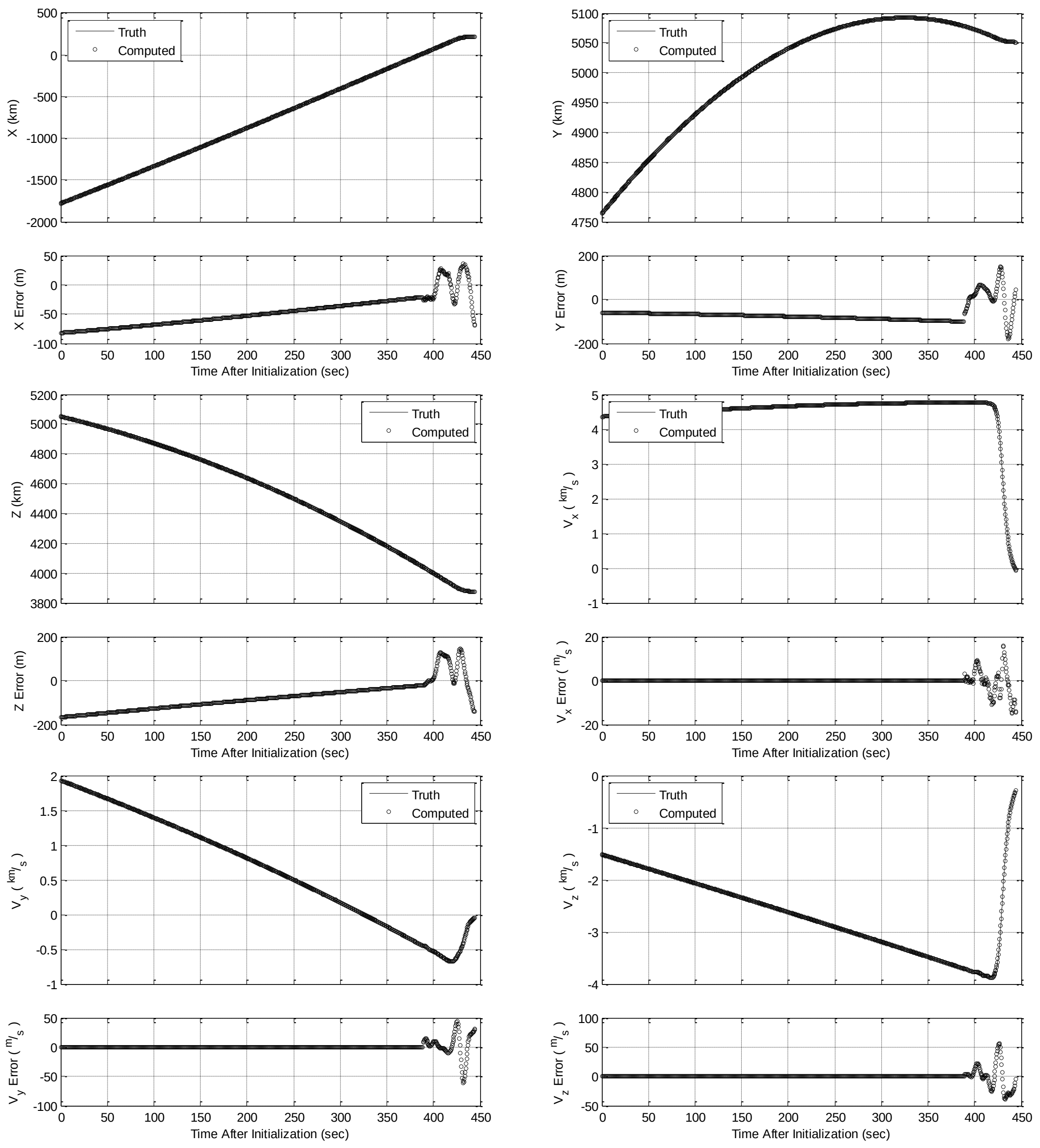


Figure 5 – Single sensor observation residuals

The results presented in Figure 5 were typical of other scenarios that were filtered. With these results satisfying the criteria for the filter performance that were being investigated, the subsequent focuses of analysis were examined. The errors in the trajectory estimate from the truth trajectory are presented in Figure 6[‡] and Table 2 as reference for analysis performed in the subsequent sections.

[‡] Acceleration units of $\frac{km}{sec^2}$ and $\frac{m}{sec^2}$ are used when plotting acceleration values and errors, respectively. In axes labels these units are labeled $^{km}/_{ss}$ and $^{m}/_{ss}$ for clarity when displayed in the MATLAB font.



Truth
Computed
X (km)
X Error (m)
Y (km)
Y Error (m)
Z (km)
Z Error (m)
V_x ($^{km}/_{s}$)
V_x Error ($^{m}/_{s}$)
V_y ($^{km}/_{s}$)
V_y Error ($^{m}/_{s}$)
V_z ($^{km}/_{s}$)
V_z Error ($^{m}/_{s}$)
Time After Initialization (sec)

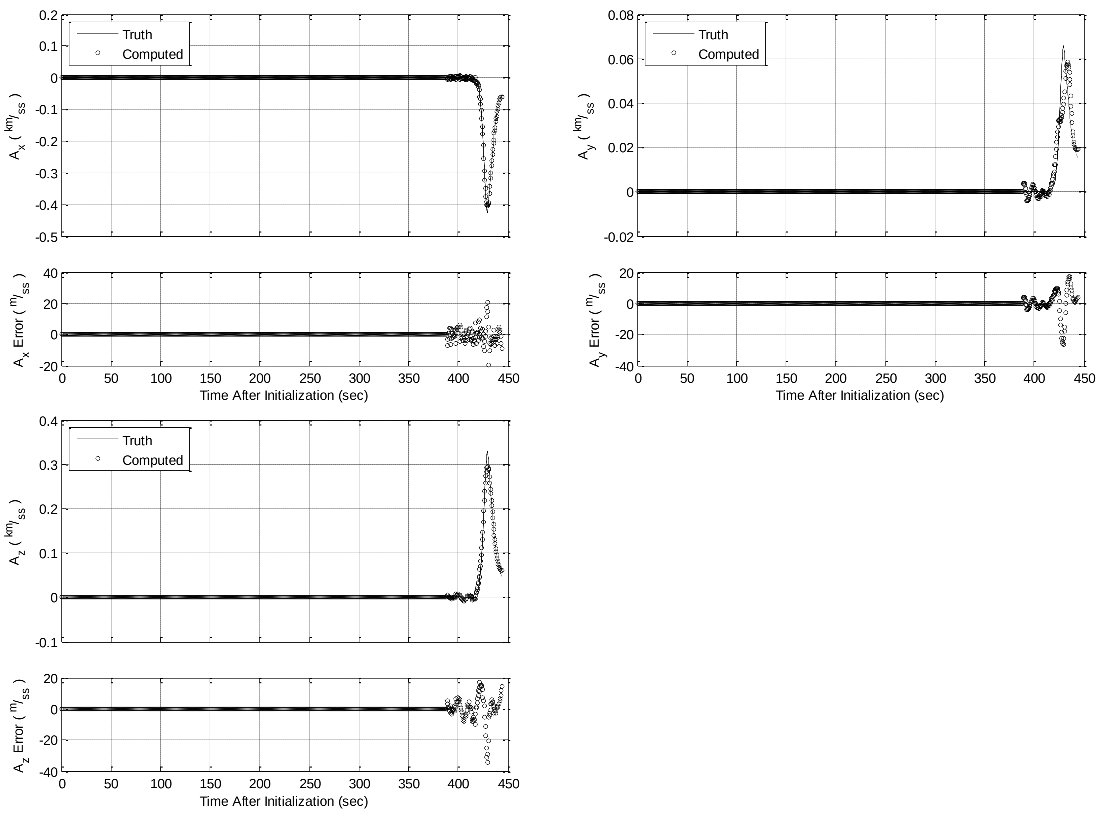


Figure 6 – Trajectory results with primary sensor

Table 2. Trajectory error values with primary sensor

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
53.1	78.5	99.3	2.4	7.7	8.1	1.9	2.9	3.1
Error RMS (m):		79.3	Error RMS $\left(\frac{m}{s}\right)$:		6.6	Error RMS $\left(\frac{m}{s^2}\right)$:		2.7

The transition of the filter from least squares to the Kalman filter can be easily observed in the plots of trajectory errors from the truth reference, presented in Figure 6. The least squares filter achieves a random scattering of error with a much smaller standard deviation than the Kalman filter does. Despite this fact, it is also noted that the Kalman filter appears to do a better job of minimizing the mean error. This is especially noticeable in the position errors from truth.

Regardless of these errors, the estimated trajectory is very close to the truth values. With these estimates determined, the ballistic coefficient can be computed from the sensed acceleration values with Equations (57) and (58). An example of ballistic coefficient values computed with this method is in Figure 7.

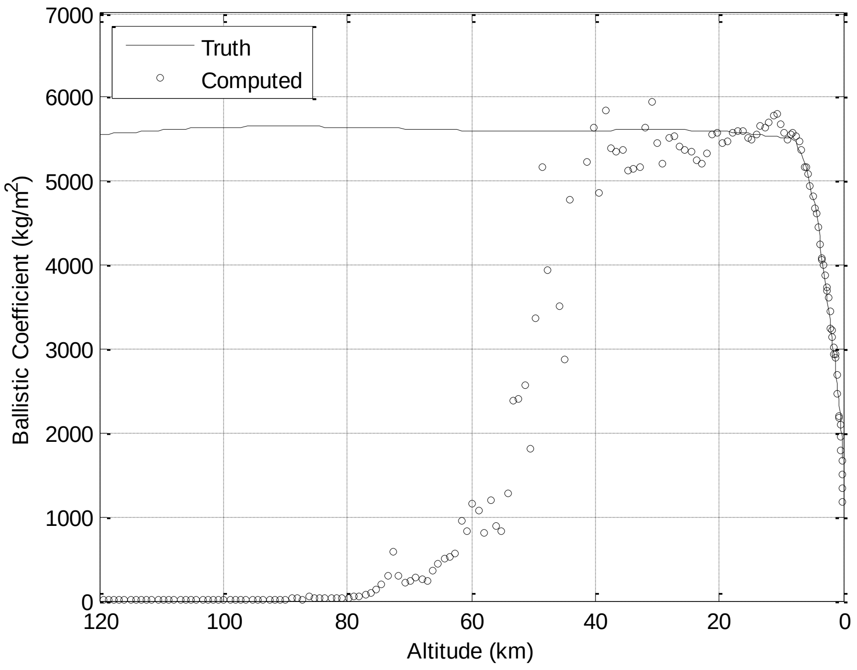


Figure 7 – Estimated ballistic coefficient

The computed ballistic coefficient in Figure 7 is typical of an RV's ballistic coefficient derived from measurements. At higher altitudes, the ballistic coefficient is largely unobservable and it climbs towards its actual value as the target descends in altitude before impact. The results obtained were close enough to the truth values to conclude that the ballistic coefficient was being properly computed.

Although initially implemented as optional, the backward smoothing pass, discussed in Chapter III - Phase 2c, was eventually deemed to be necessary for optimal results. A comparison of acceleration estimates computed with and without the smoother pass enabled appears in Figure 8. While the unsmoothed values appear to be close to the truth, their mean error has a standard deviation of 6.9 $\left(\frac{m}{s^2}\right)$ compared to 1.9 $\left(\frac{m}{s^2}\right)$ for the smoothed estimates.

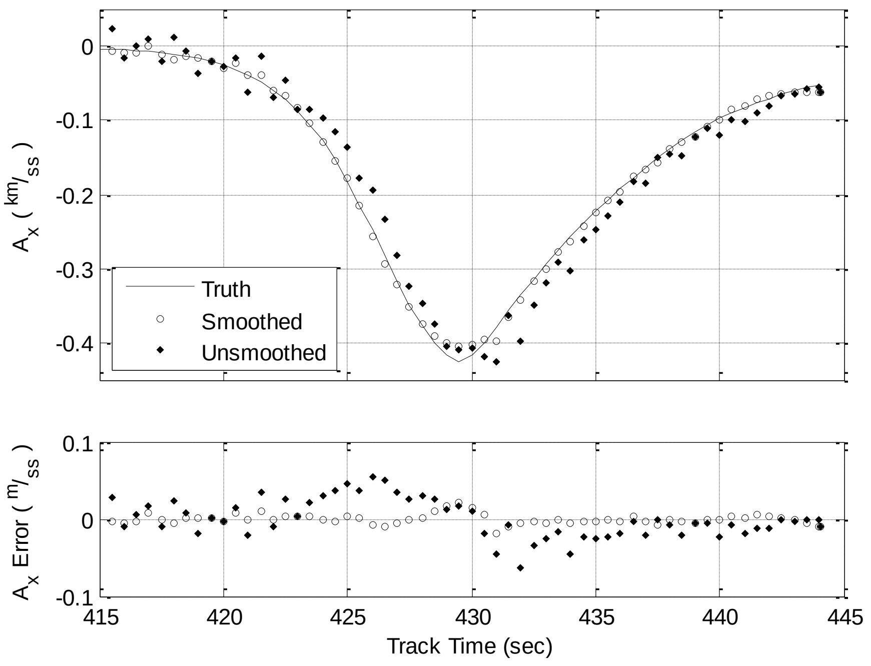


Figure 8 – Effects of smoother pass on estimates

The benefits of the smoother pass are further revealed when these accelerations are transformed into the corresponding ballistic coefficient estimates. The ballistic coefficient estimates from the smoothed and unsmoothed estimates appear in Figure 9. This zoomed view highlights the errors throughout the unsmoothed results. Whereas the smoothed accelerations converge to the truth value, the unsmoothed accelerations overshoot the truth and then overcompensate to a ballistic coefficient that is less than the truth. For these reasons, the backward smoother pass was deemed integral to achieving the best results.

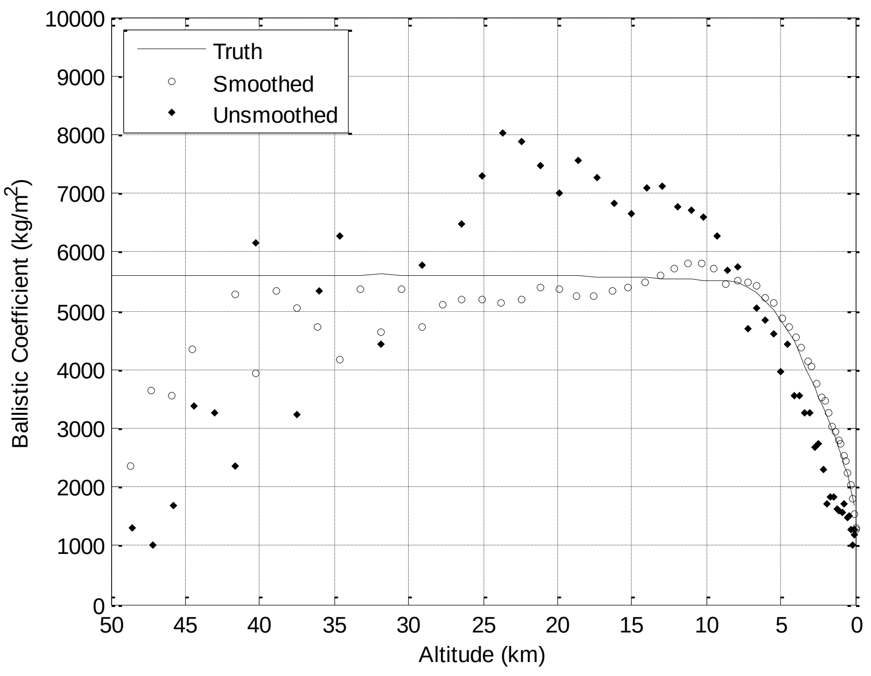


Figure 9 – Ballistic coefficient estimated with and without smoother pass.

To further validate the filter performance, scenarios with maneuvering targets were constructed to observe the filter's ability to model non-ballistic accelerations. The maneuvers performed by the target are summarized in Table 3.

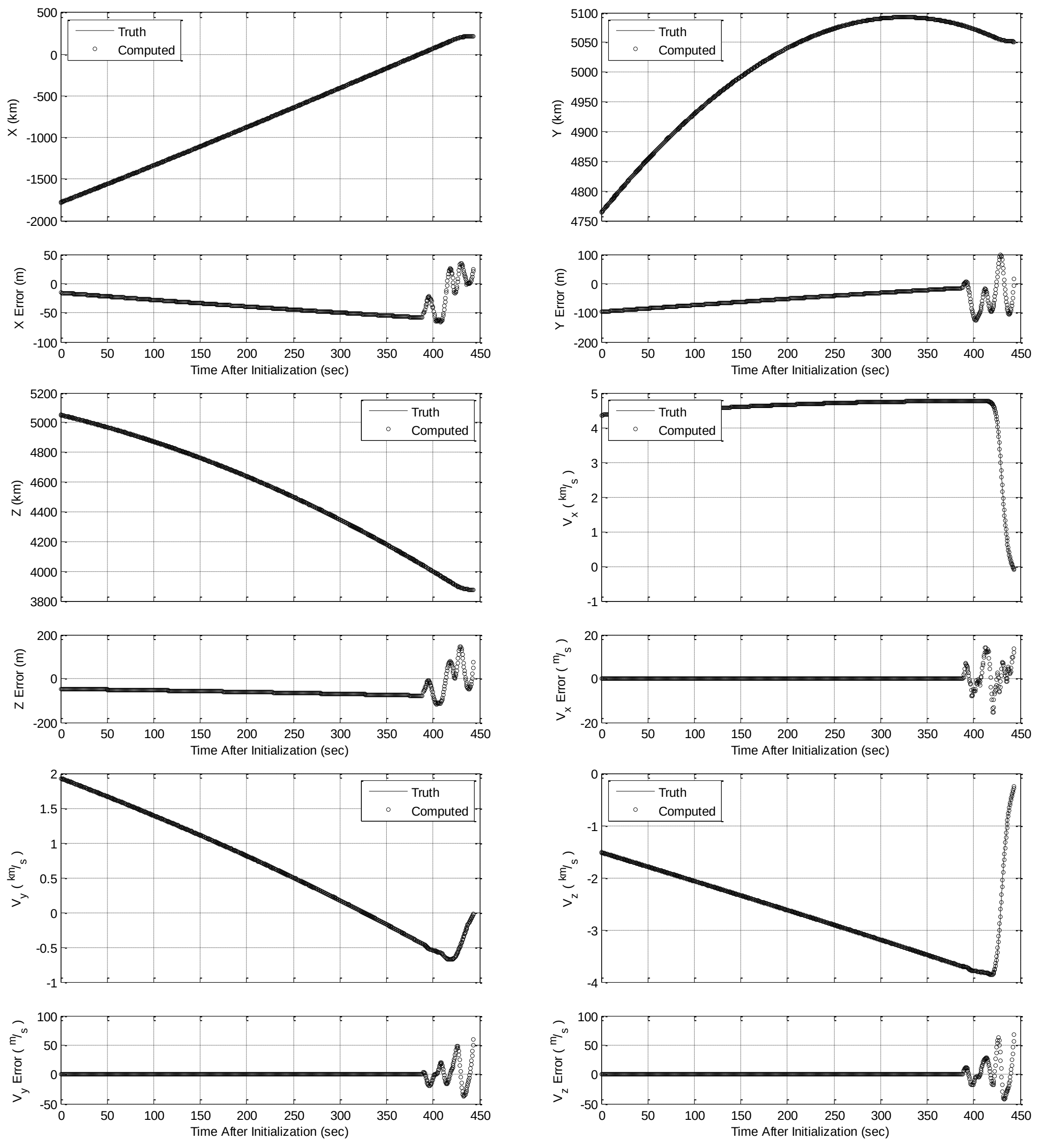
Table 3. Reentry scenarios[§]

SCENARIO	REENTRY MANEUVERS
1	None
2	Below 40 km alt: $\ddot{x} = \ddot{x}_{grav} - .000980665(alt - 40km)$
3	Below 40 km alt: $\ddot{x} = \ddot{x}_{grav} - .000980665(alt - 40km)$ Below 20 km alt: $\ddot{y} = \ddot{y}_{grav} - 2\big(.000980665(alt - 20km)\big)$

Both scenarios 2 and 3 involved accelerations in the *x* direction that ramp up from zero as altitude decreased. Scenario 3 added a level of complexity with a second

[§] These reentry scenarios were arbitrarily defined maneuver in the *y* direction that begins at a lower altitude. This maneuver ramped up more rapidly than the maneuver in the *x* direction.

These scenarios were run through the filter with the same observing sensor. The results showed that the filter was capable of modeling maneuvering accelerations and had similar errors to the non-maneuvering case. Error plots are presented in Figure 10 and Figure 11 while a summary of the fits are in Table 4 and Table 5.



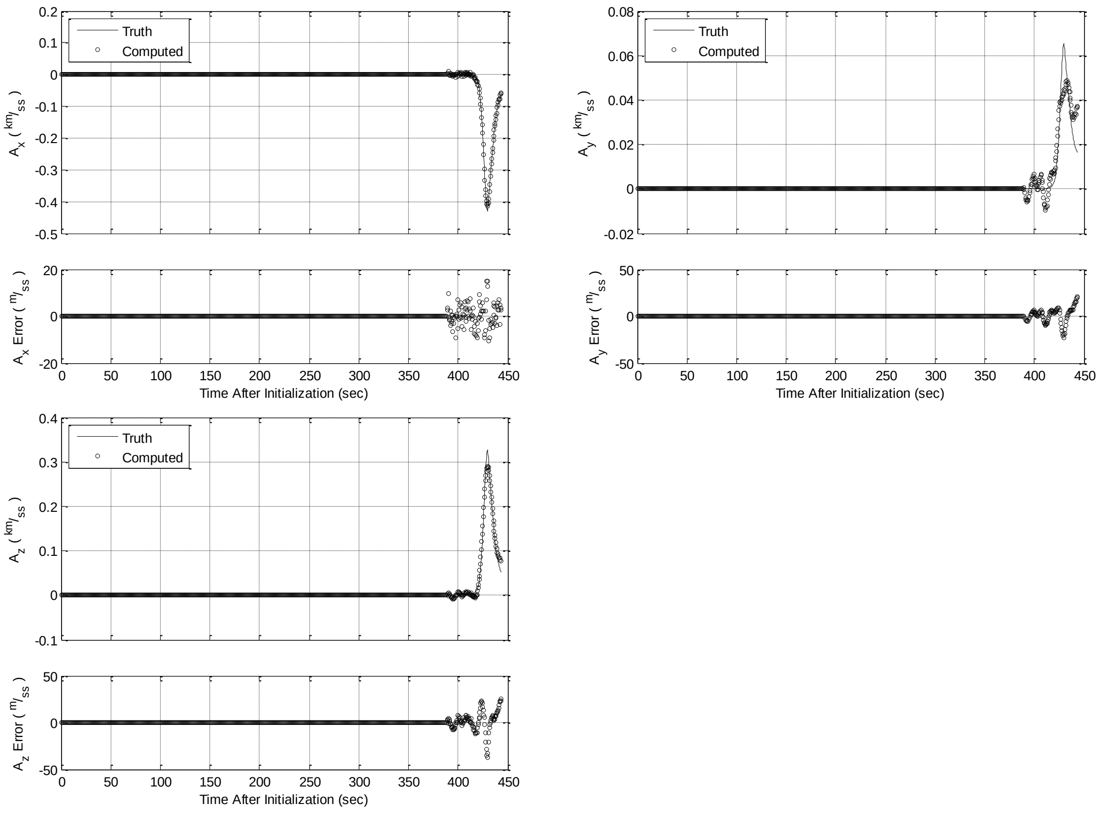
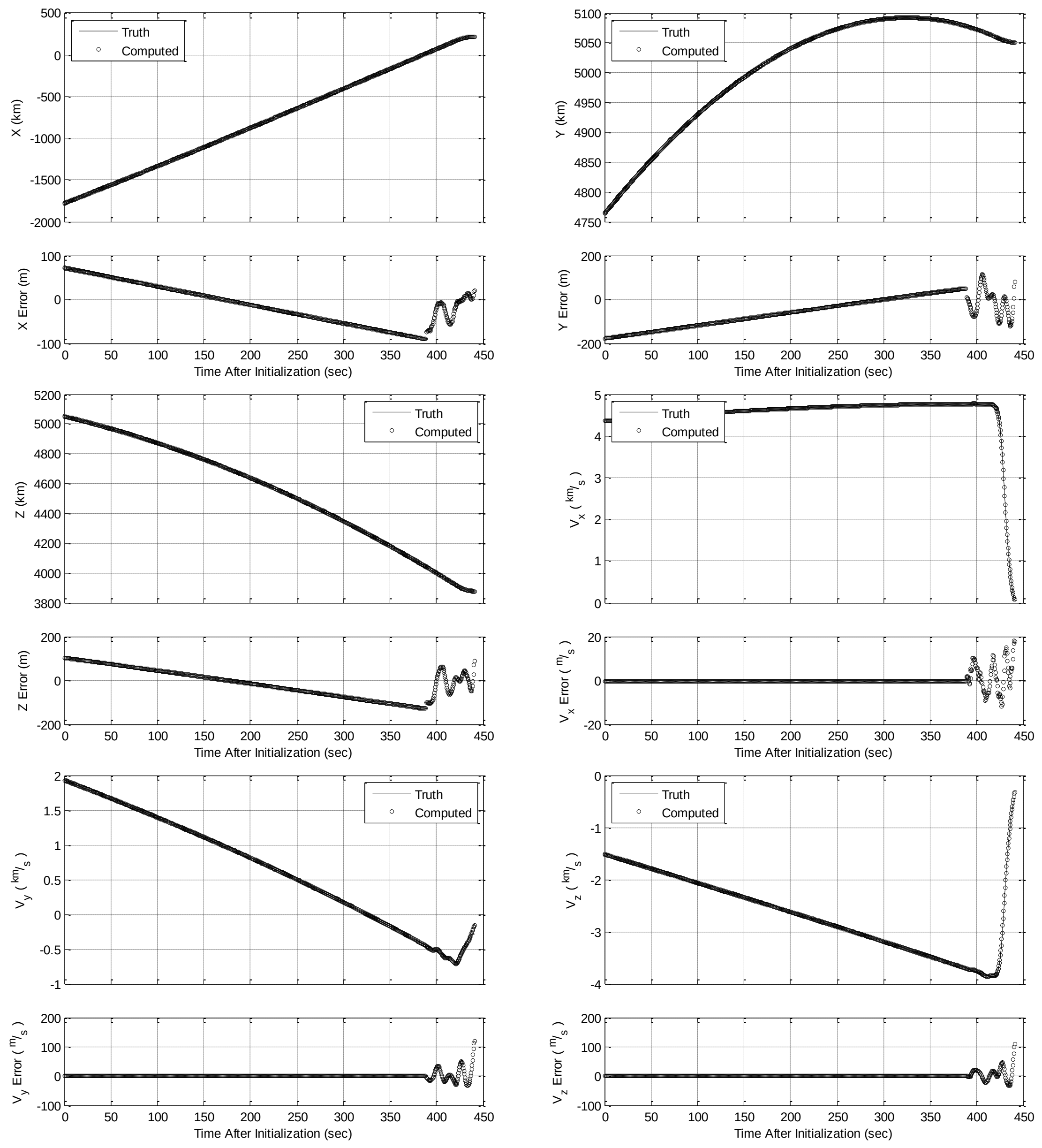


Figure 10 – Trajectory results from scenario 2, maneuvering target

Table 4. Trajectory errors from scenario 2, maneuvering target

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
39.6	61.1	63.5	2.4	7.5	8.9	1.8	2.9	4.0


Truth
Computed
X (km)
X Error (m)
Y (km)
Y Error (m)
Z (km)
Z Error (m)
V_x ($^{km}/_s$)
V_x Error ($^{m}/_s$)
V_y ($^{km}/_s$)
V_y Error ($^{m}/_s$)
V_z ($^{km}/_s$)
V_z Error ($^{m}/_s$)
Time After Initialization (sec)

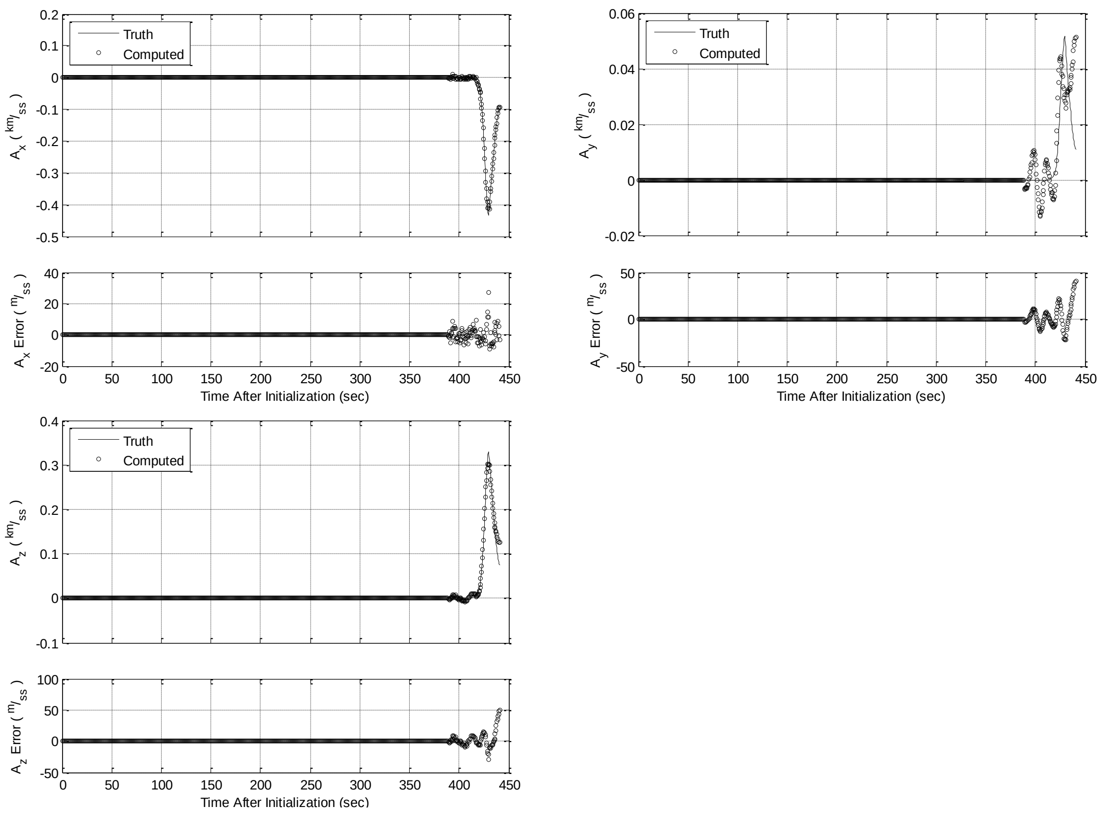


Figure 11 – Trajectory results from scenario 3, maneuvering target

Table 5. Trajectory errors from scenario 3, maneuvering target

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
46.9	88.9	66.7	2.5	10.1	8.7	1.8	4.6	4.5

The last discovery made that affected the error of the estimated trajectory pertained to the least squares sliding window portion of the algorithm. The section of the filter was intended to provide the Kalman filter with an initial guess for the position of the target during the maneuvering portion of the data. After analysis of the estimates computed with the LSSW compared to those computed without the LSSW filter active, it was determined that the LSSW failed to improve the trajectory results in every scenario.

As an example of this behavior, the results of the baseline scenario are presented in Table 6 both with and without the LSSW active.

Table 6. Trajectory errors due to LSSW

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
56.9	148.7	139.9	2.7	11.4	9.6	1.9	3.8	3.4
Error RMS (m):		122.4	Error RMS $\left(\frac{m}{s}\right)$:		8.7	Error RMS $\left(\frac{m}{s^2}\right)$:		3.1
Baseline Error:		79.3	Baseline Error:		6.6	Baseline Error:		2.7
Difference:		+54.4%	Difference:		+31.8%	Difference:		+14.8%

Case 2 – Filter Performance Improvements Through Additional Sensors.

In order to investigate performance improvements that could result from filtering data from multiple sensors, three additional sensor locations were defined. The three additional sensor locations, all equidistant from the impact point, are described in Table 7 and displayed graphically in Figure 12. These sensors incorporate the same random observation noise as described in Table 1.

Table 7. Observing sensors

SENSOR	CHARACTERISTIC
1	Approximately Along Reentry Azimuth, 400km from Impact
2	Approximately 45^o to Reentry Azimuth, 400km from Impact
3	Approximately 90^o to Reentry Azimuth, 400km from Impact

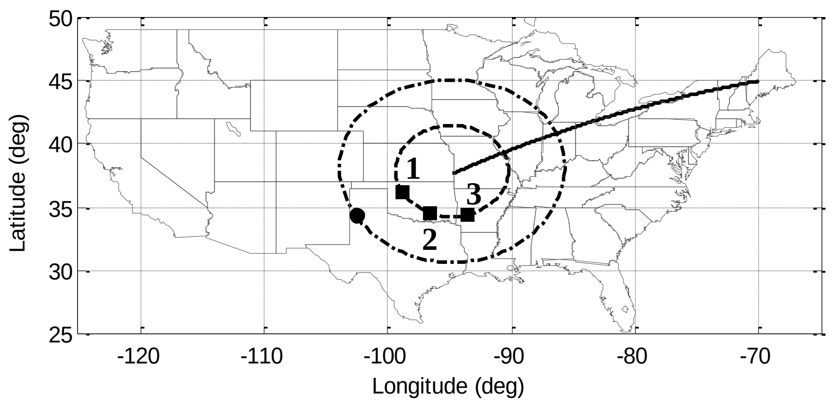
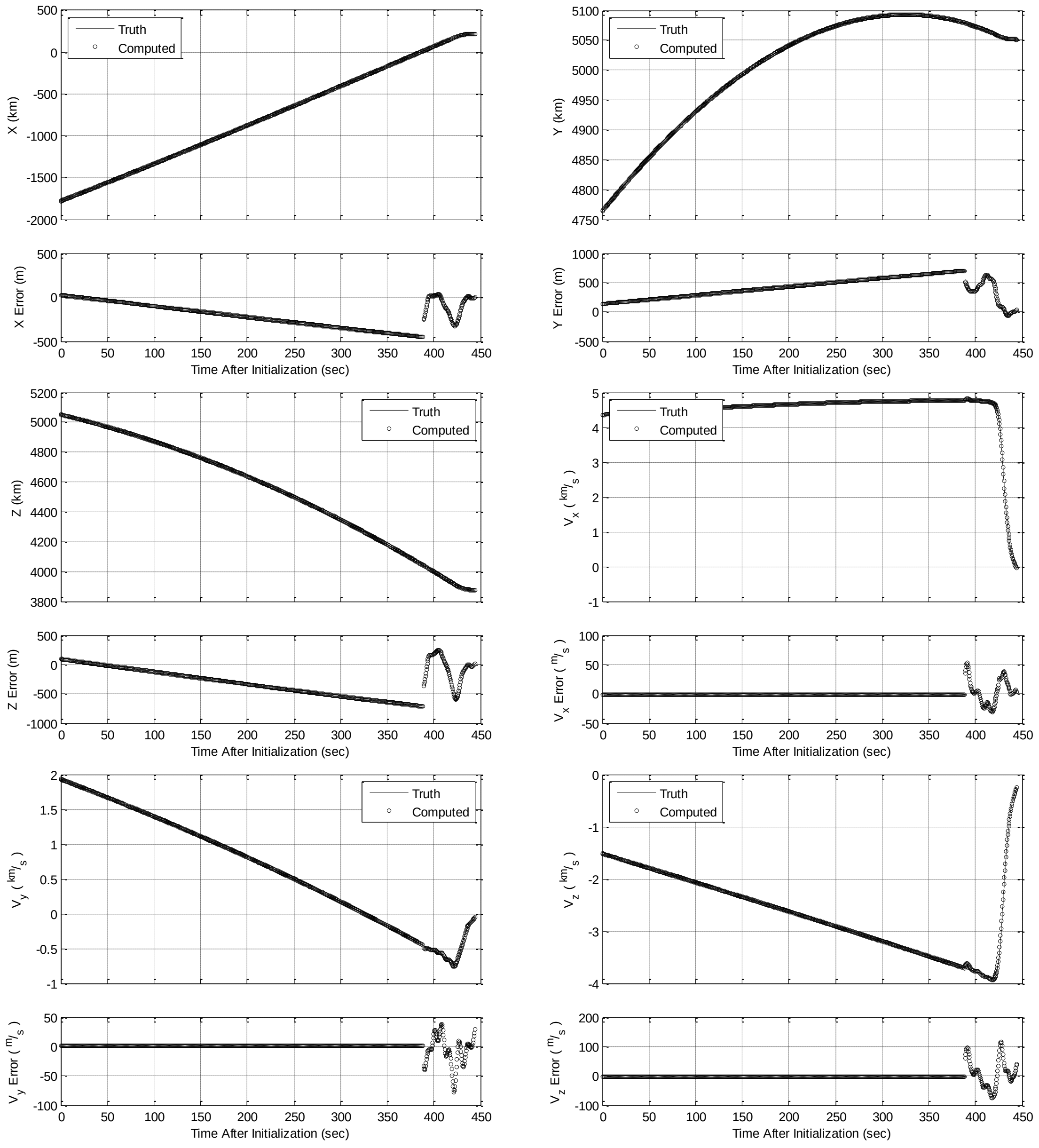


Figure 12 – Additional sensor collection geometries

The results of incorporating data from an additional sensor at location 1 are presented in Figure 13 and a summary of the trajectory errors from the truth are presented in Table 8. In general, the addition of a second sensor along the reentry azimuth has no positive effect on the results of the trajectory fit, in this scenario. In fact, the results are worse than those achieved with the primary sensor alone. It is unclear why the results are as degraded as they are, but it is assumed that adding a sensor at location 1 adds little observability to the problem beyond what the primary sensor already provides.



Truth
Computed
X (km)
X Error (m)
Y (km)
Y Error (m)
Z (km)
Z Error (m)
V_x ($^{km}/_s$)
V_x Error ($^{m}/_s$)
V_y ($^{km}/_s$)
V_y Error ($^{m}/_s$)
V_z ($^{km}/_s$)
V_z Error ($^{m}/_s$)
Time After Initialization (sec)

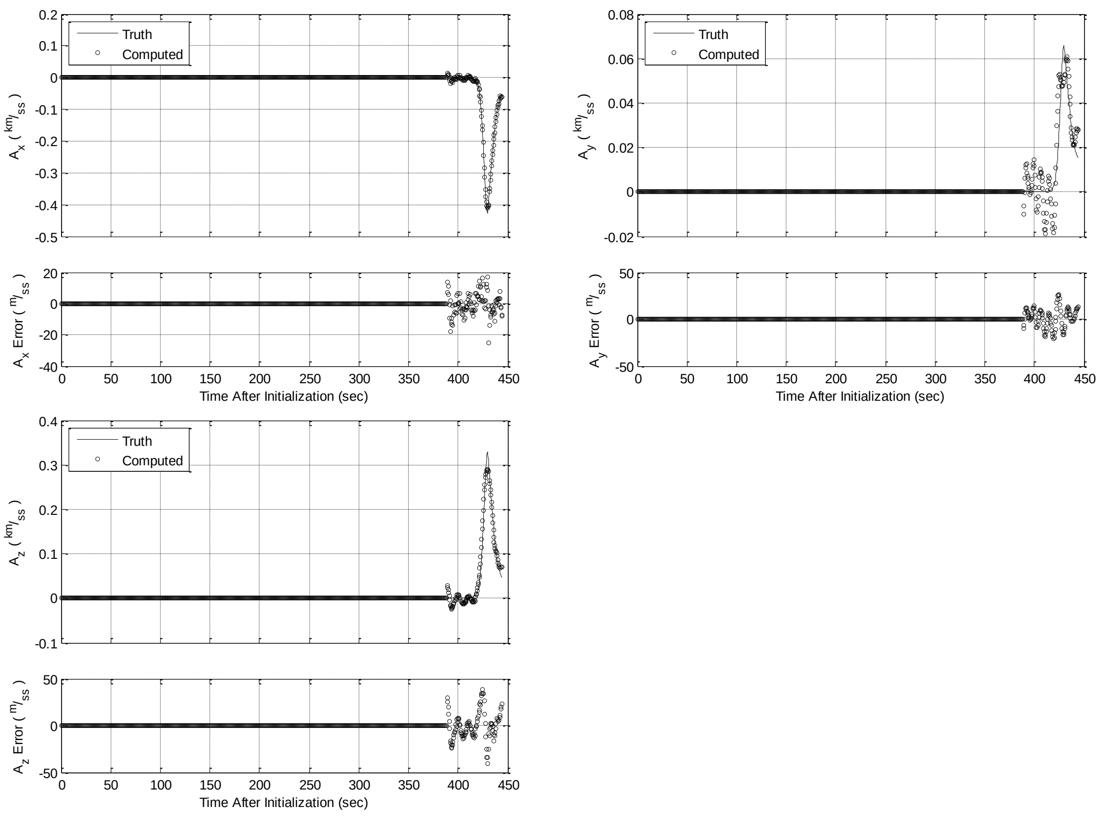
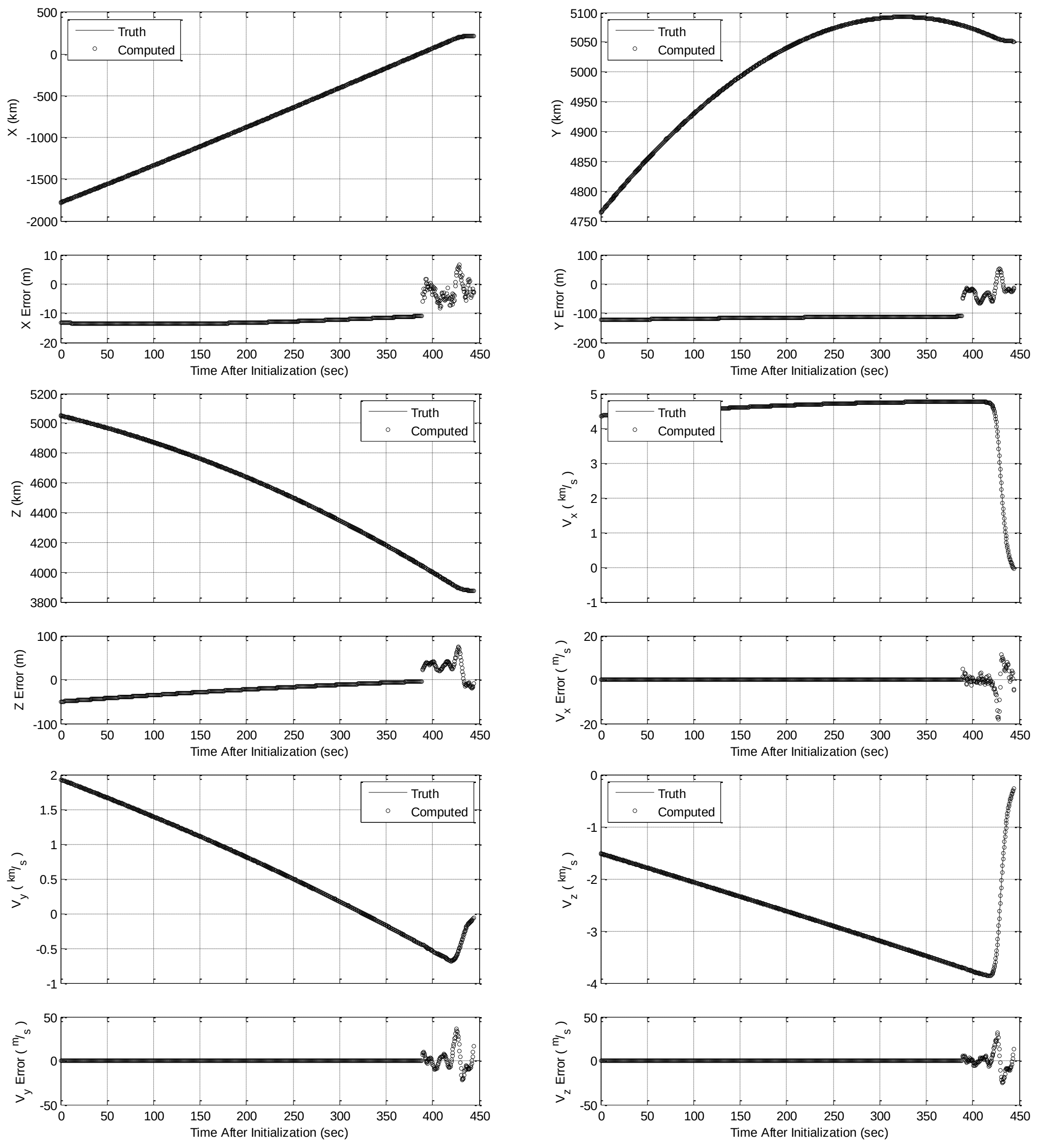


Figure 13 – Trajectory results with addition of sensor 1

Table 8. Combined trajectory errors with addition of sensor 1

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
246.2	443.0	381.2	7.8	9.8	17.3	2.5	3.8	5.5
Error RMS (m):		366.1	Error RMS $\left(\frac{m}{s}\right)$:		12.3	Error RMS $\left(\frac{m}{s^2}\right)$:		4.1
Baseline Error:		79.3	Baseline Error:		6.6	Baseline Error:		2.7
Difference:		$^+$361.7%	Difference:		$^+$86.4%	Difference:		$^+$51.9%

The results of incorporating data from an additional sensor at location 2 are presented in Figure 14 and a summary of the trajectory errors from the truth are presented in Table 9. In general, the addition of a second sensor approximately 45 degrees off of the reentry azimuth has a positive effect on the results of the trajectory fit. The position, velocity, and acceleration errors are all improved over the results achieved using only the primary sensor.



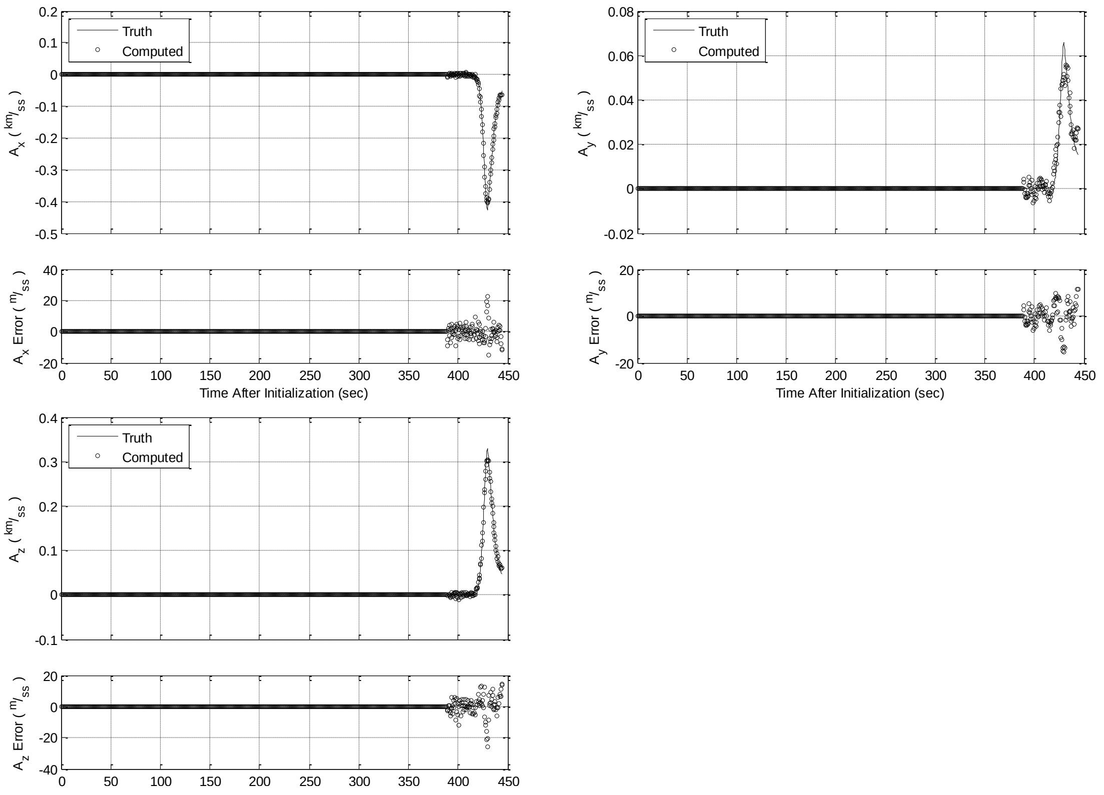
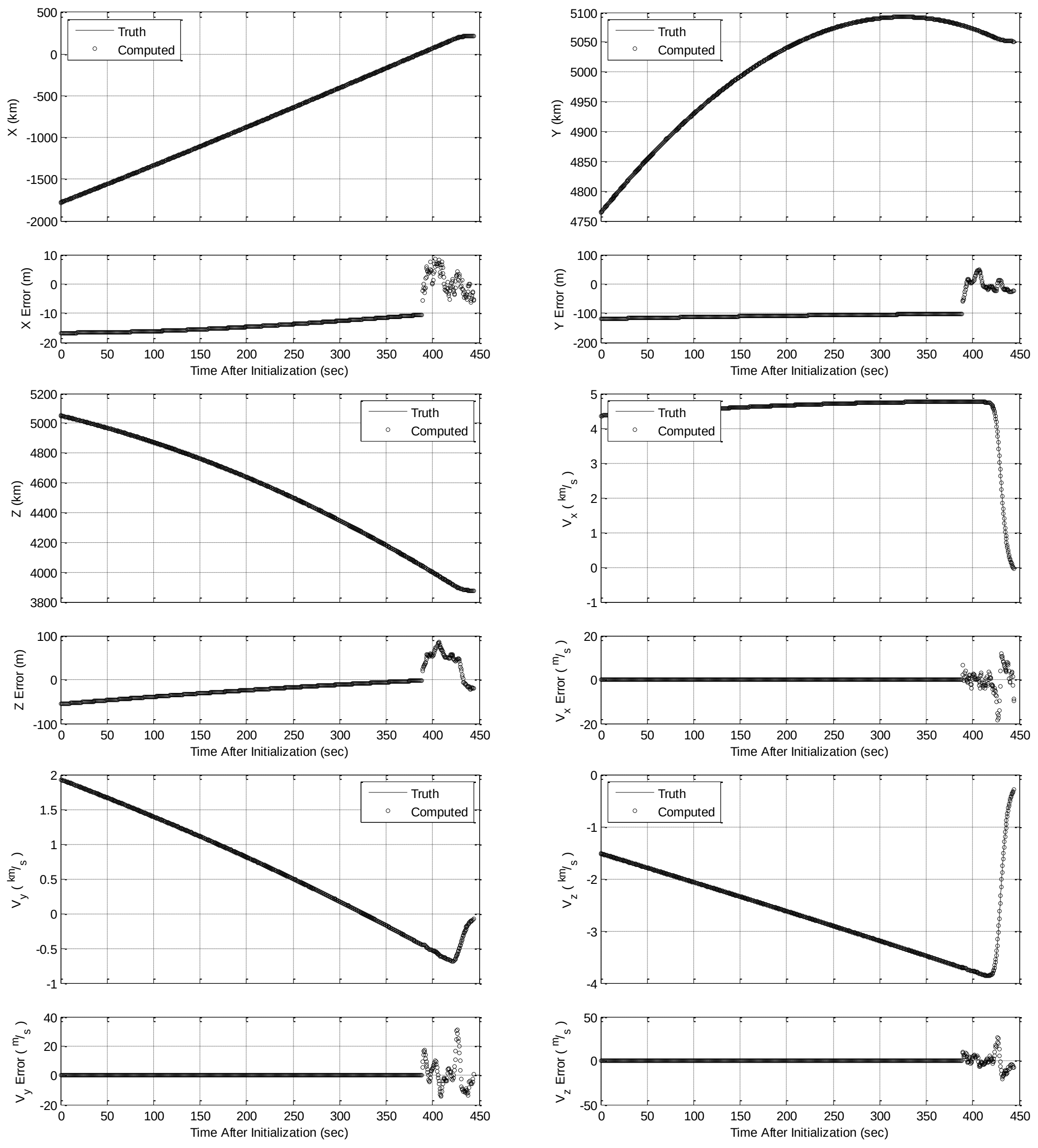


Figure 14 – Trajectory results with addition of sensor 2

Table 9. Combined trajectory errors with addition of sensor 2

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
12.2	109.2	28.7	1.8	4.3	3.8	2.0	2.0	2.3
Error RMS (m):		65.6	Error RMS $\left(\frac{m}{s}\right)$:		3.5	Error RMS $\left(\frac{m}{s^2}\right)$:		2.1
Baseline Error:		79.3	Baseline Error:		6.6	Baseline Error:		2.7
Difference:		-17.3%	Difference:		-47.0%	Difference:		-22.2%

The results of incorporating data from an additional sensor at location 3 are presented in Figure 15 and a summary of the trajectory errors from the truth are presented in Table 10. In general, the addition of a second sensor approximately 90 degrees off of the reentry azimuth has a positive effect on the results of the trajectory fit, similar to the results achieve with additional sensor 2. The position, velocity, and acceleration errors are all improved over the results achieved using only the primary sensor, and when averaged are slightly better than those achieved with additional sensor 2.



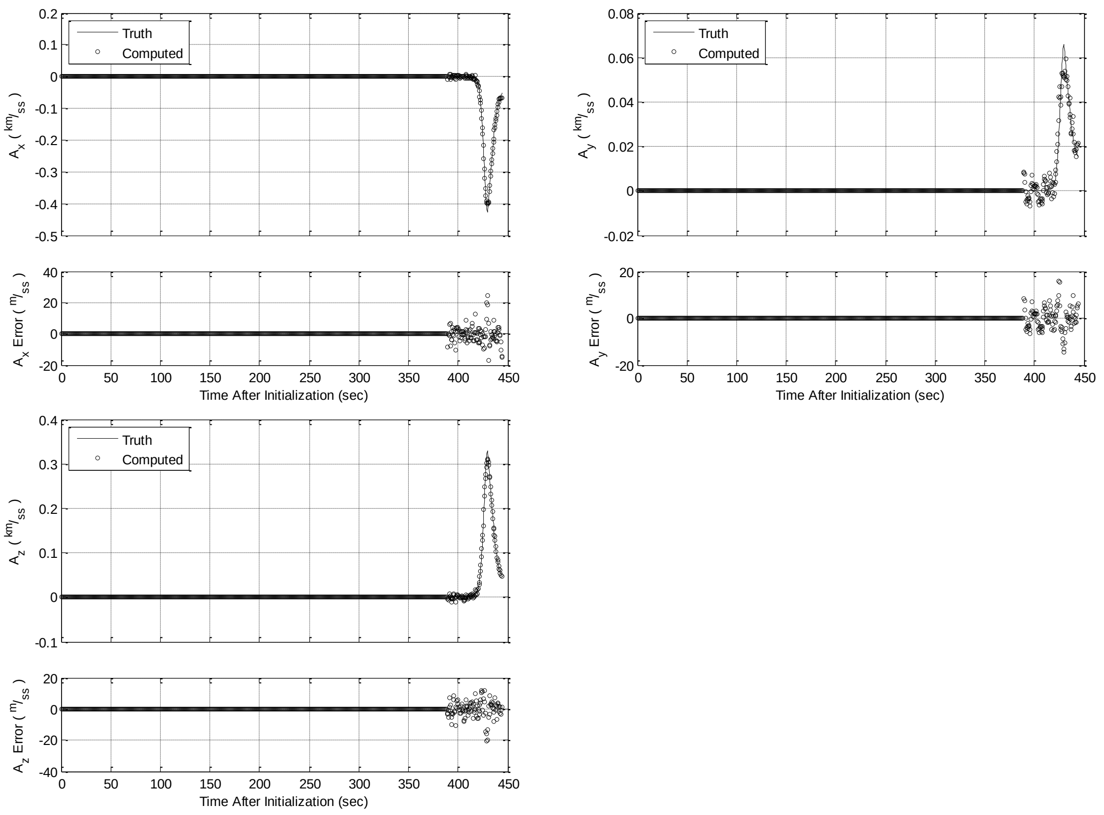


Figure 15 – Trajectory results with addition of sensor 3

Table 10. Combined trajectory errors with addition of sensor 3

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
13.7	102.9	33.4	1.9	3.6	3.2	2.2	1.8	2.0
Error RMS (m):		63.0	Error RMS $\left(\frac{m}{s}\right)$:		3.0	Error RMS $\left(\frac{m}{s^2}\right)$:		2.0
Baseline Error:		79.3	Baseline Error:		6.6	Baseline Error:		2.7
Difference:		-20.6%	Difference:		-54.5%	Difference:		-25.9%

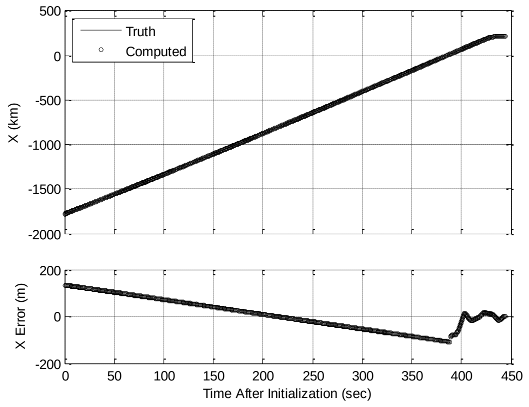
These results show that accuracy can be improved by collecting data from an additional sensor, although there are geometry considerations. A secondary sensor added along the reentry azimuth was detrimental to the accuracy of the combined results, but a secondary sensor located either 45 or 90 degrees to the reentry azimuth improved the accuracy of the results. Alternative scenarios were investigated that did not show the addition of a sensor along the reentry azimuth to be detrimental, but the trend of better results being achieved by shifting a secondary sensor away from the reentry azimuth was consistently found throughout these alternatives.

This was anticipated based on general knowledge of data filtering. By adding a second sensor along the trajectory's azimuth, there is little added information that was not already present from the primary sensor. By adding that sensor orthogonal to the azimuth, the amount of new information added to the filter is maximized.

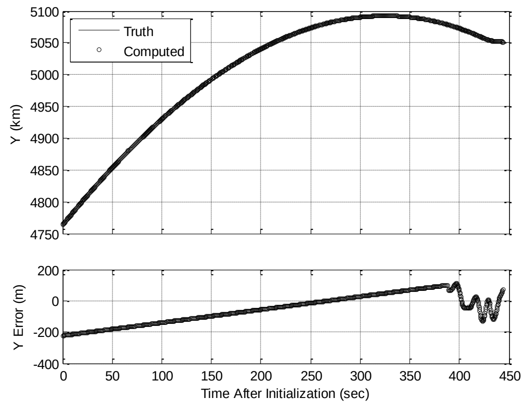
Case 3 – Filter Performance Improvements Through Increased Data Rate.

The next variation to the standard collection scheme is the modification of the data rate at which the sensor or sensors collect observations. For these cases, the rate of collection will be increased from 2 to 3 Hz in order to investigate what effects this may have.

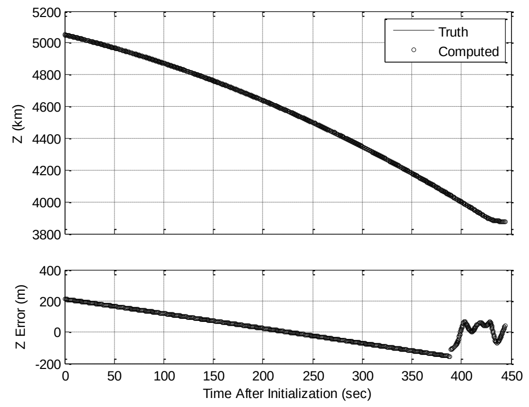
The first scenario that will be modified is the initial scenario whose results are presented in Figure 6 and Table 2. The single, primary sensor collects data at the increased rate; the results of this are presented in Figure 16 and Table 11.



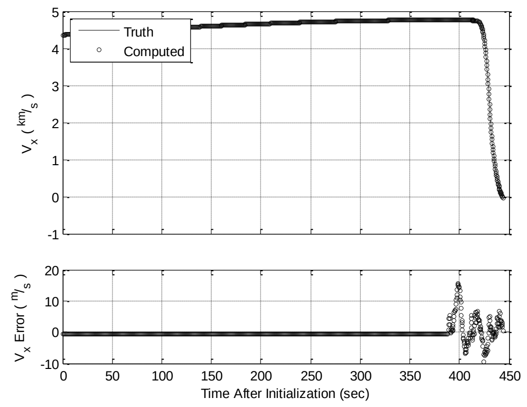
Truth
Computed
X (km)
X Error (m)
Time After Initialization (sec)



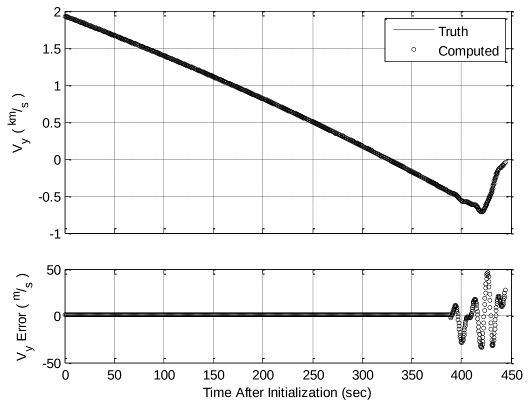
Truth
Computed
Y (km)
Y Error (m)
Time After Initialization (sec)



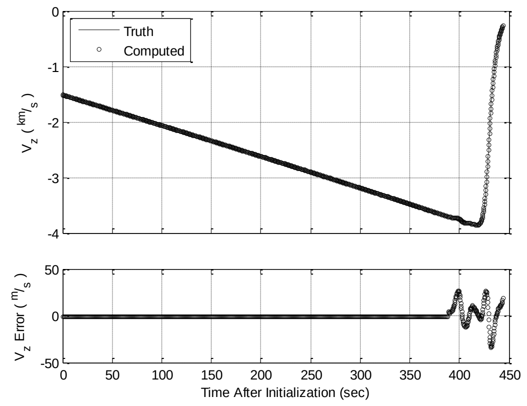
Truth
Computed
Z (km)
Z Error (m)
Time After Initialization (sec)



Truth
Computed
V_x ($^{km}/_{s}$)
V_x Error ($^{m}/_{s}$)
Time After Initialization (sec)



Truth
Computed
V_y ($^{km}/_{s}$)
V_y Error ($^{m}/_{s}$)
Time After Initialization (sec)



Truth
Computed
V_z ($^{km}/_{s}$)
V_z Error ($^{m}/_{s}$)
Time After Initialization (sec)

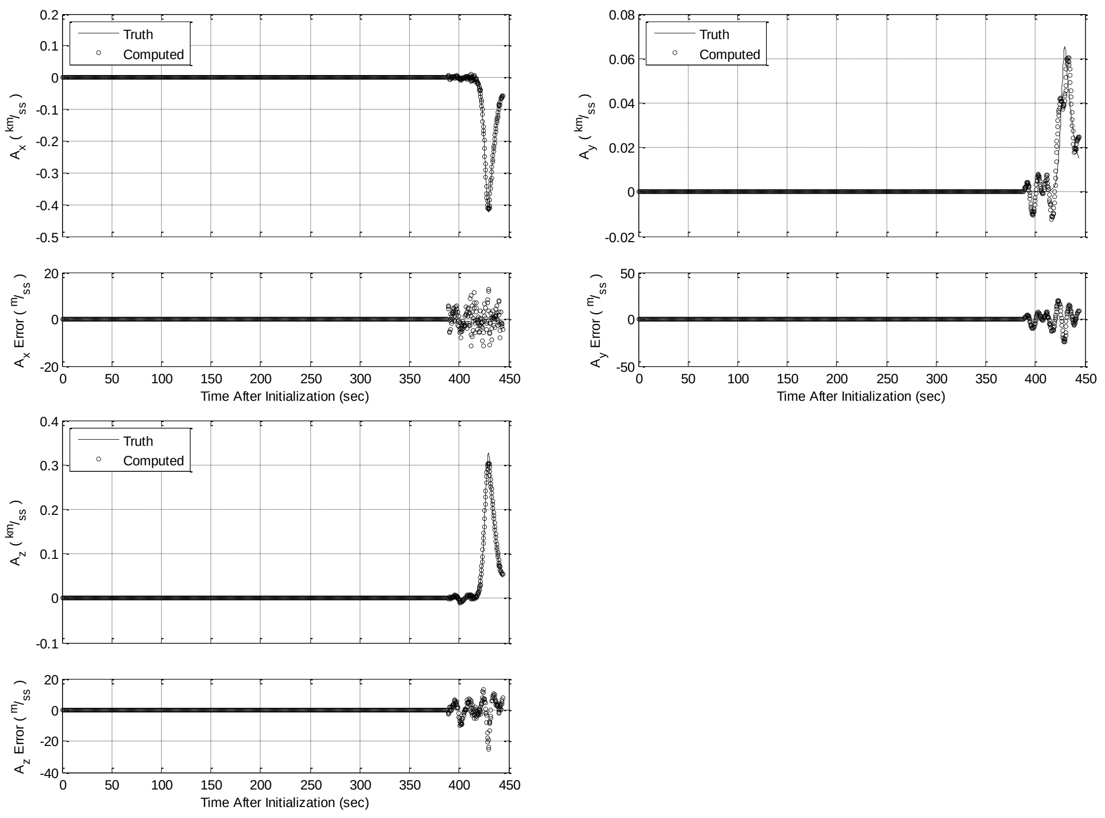


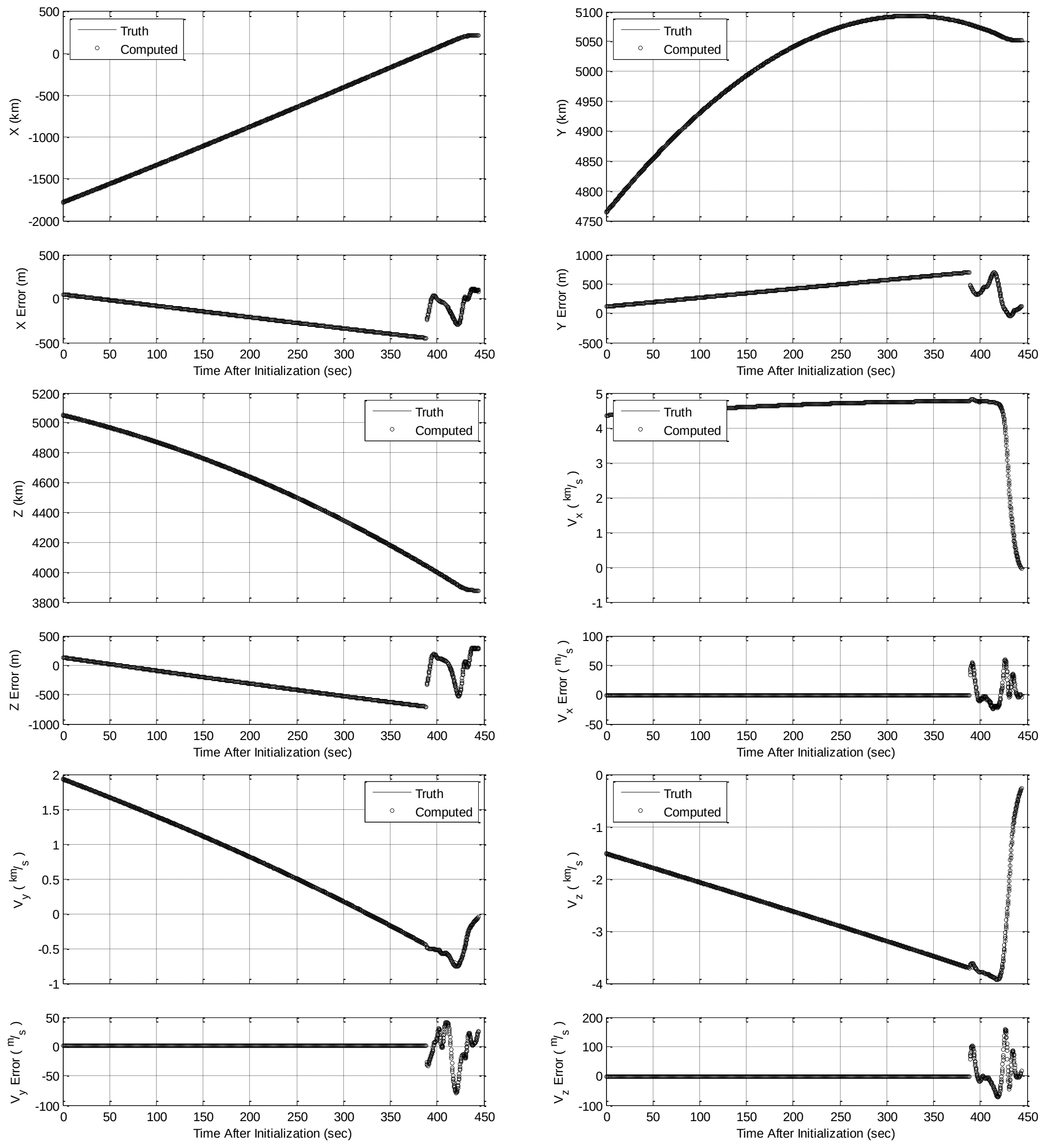
Figure 16 – Trajectory results with primary sensor at increased data rate

Table 11. Trajectory errors with primary sensor at increased data rate

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
67.8	107.5	104.8	2.0	6.8	5.0	1.6	3.3	2.2
Error RMS (m):		95.1	Error RMS $\left(\frac{m}{s}\right)$:		5.0	Error RMS $\left(\frac{m}{s^2}\right)$:		2.5
2Hz Error:		79.3	2Hz Error:		6.6	2Hz Error:		2.7
Difference:		$^{+}$19.9%	Difference:		$^{-}$24.2%	Difference:		$^{-}$7.4%

The single sensor results show improvements in velocity and acceleration at the increased data rate, with reduced accuracy in position. The next modified scenario will increase the data rates of secondary sensors that are collecting data from different locations than the primary sensor. The first of these will recreate the geometry whose results are presented in Figure 13 and Table 8 with the secondary sensor at location 1.

The results of this geometry with the secondary sensor now collecting at 3 Hz are presented in Figure 17 and Table 12.



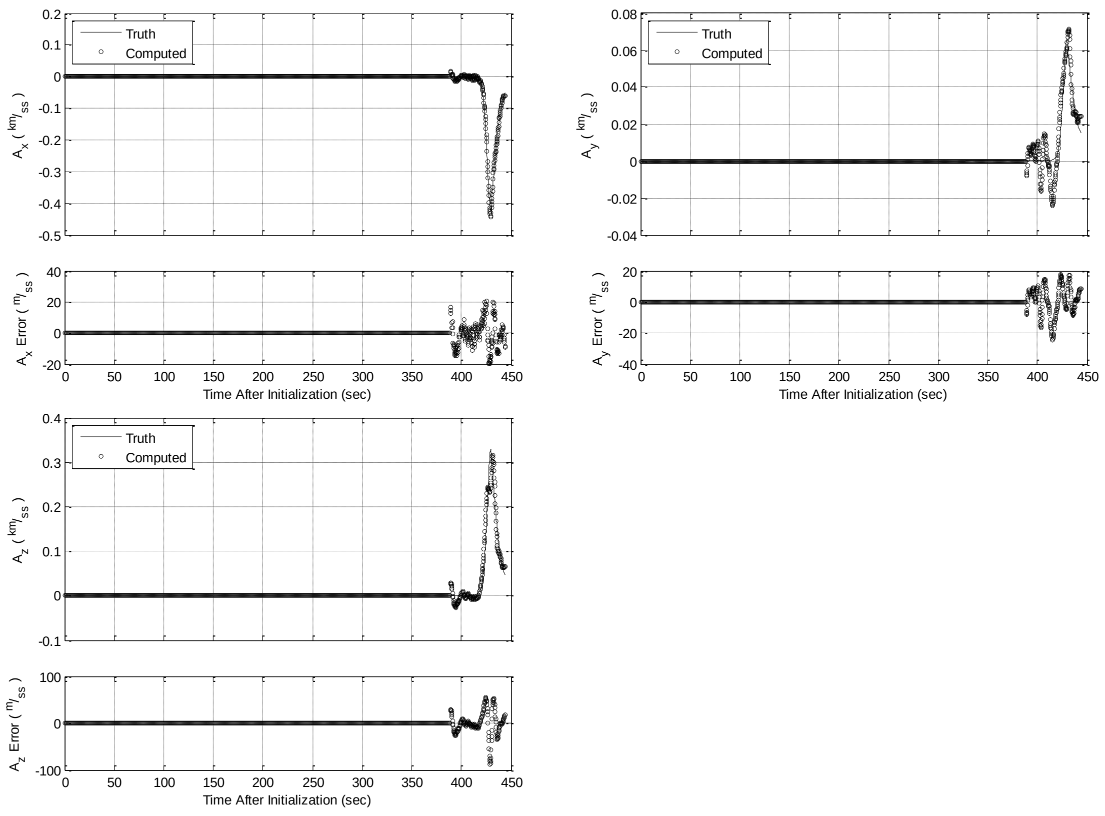


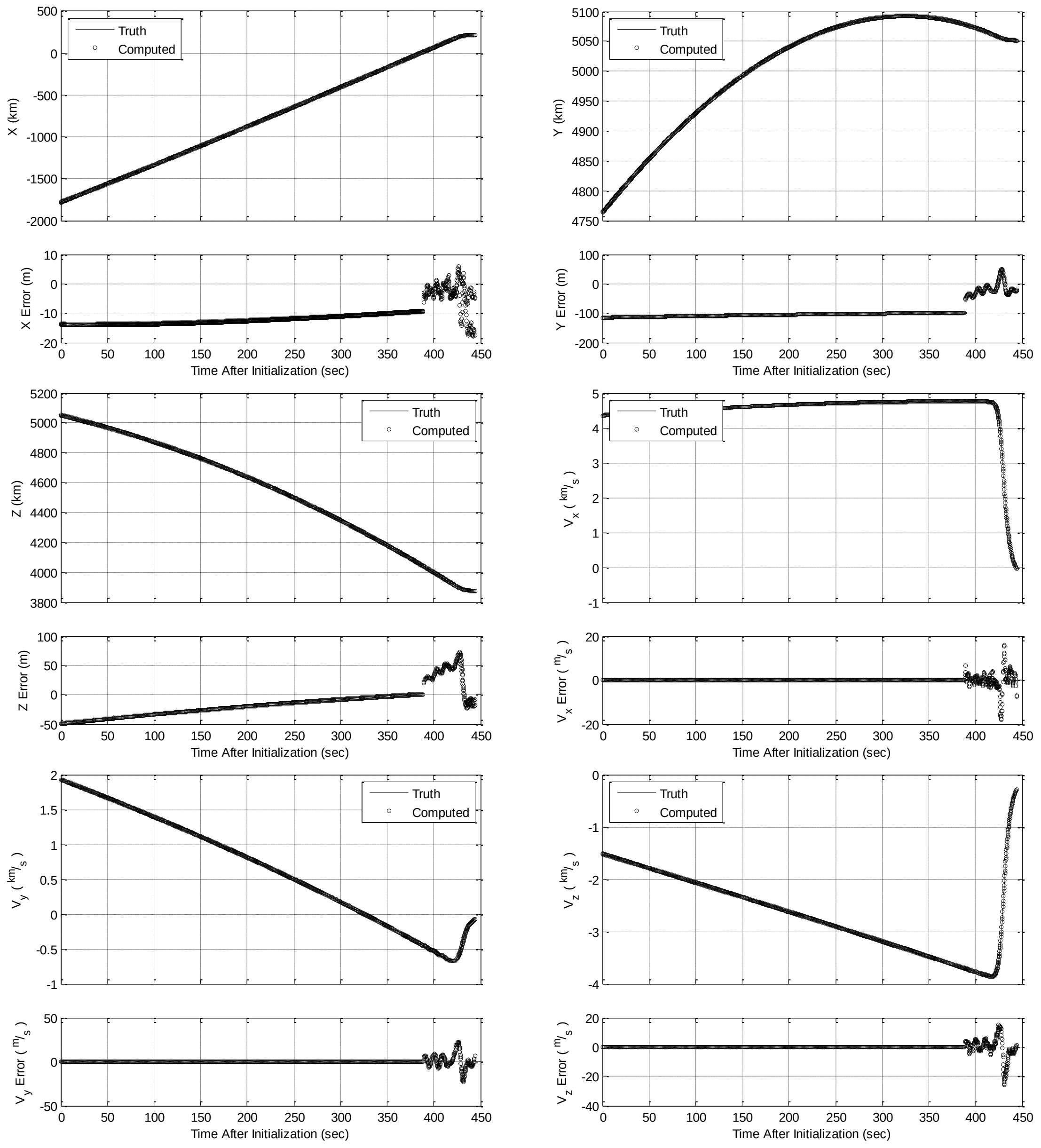
Figure 17 – Trajectory results with addition of sensor 1 at increased rate

Table 12. Trajectory errors with addition of sensor 1 at increased rate

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
237.6	433.2	367.5	8.5	11.3	19.5	3.1	3.4	9.0
Error RMS (m):		355.5	Error RMS $\left(\frac{m}{s}\right)$:		13.9	Error RMS $\left(\frac{m}{s^2}\right)$:		5.8
2Hz Error:		366.1	2Hz Error:		12.3	2Hz Error:		4.1
Difference:		-2.9%	Difference:		+13.0%	Difference:		+41.5%

While the trajectory results still show a significant error when compared to the results from the primary sensor alone, the results show mixed improvement with respect to those obtained when a sensor at location 1 operated at the nominal collection rate. In this case, the position error is improved by 2.9% while the velocity and acceleration errors worsened by 13.0% and 41.5%, respectively. These results do not show promise for an increased collection rate being a means of achieving increased accuracy, but with such poor errors to begin with, it may be an unsuitable case for comparison.

The next modified scenario will operate a sensor at location 2, similar to the results presented in Figure 14 and Table 9, at the increased data rate of 3 Hz. Results from this scenario are presented in Figure 18 and Table 13.



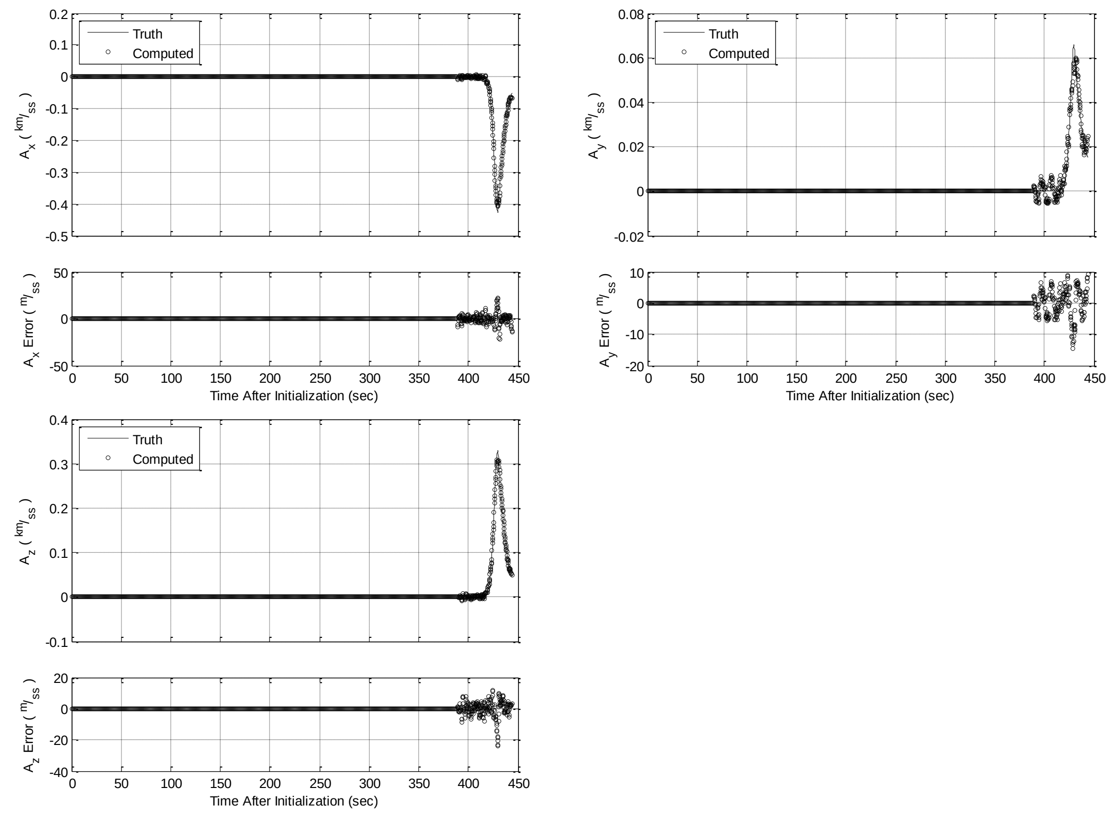


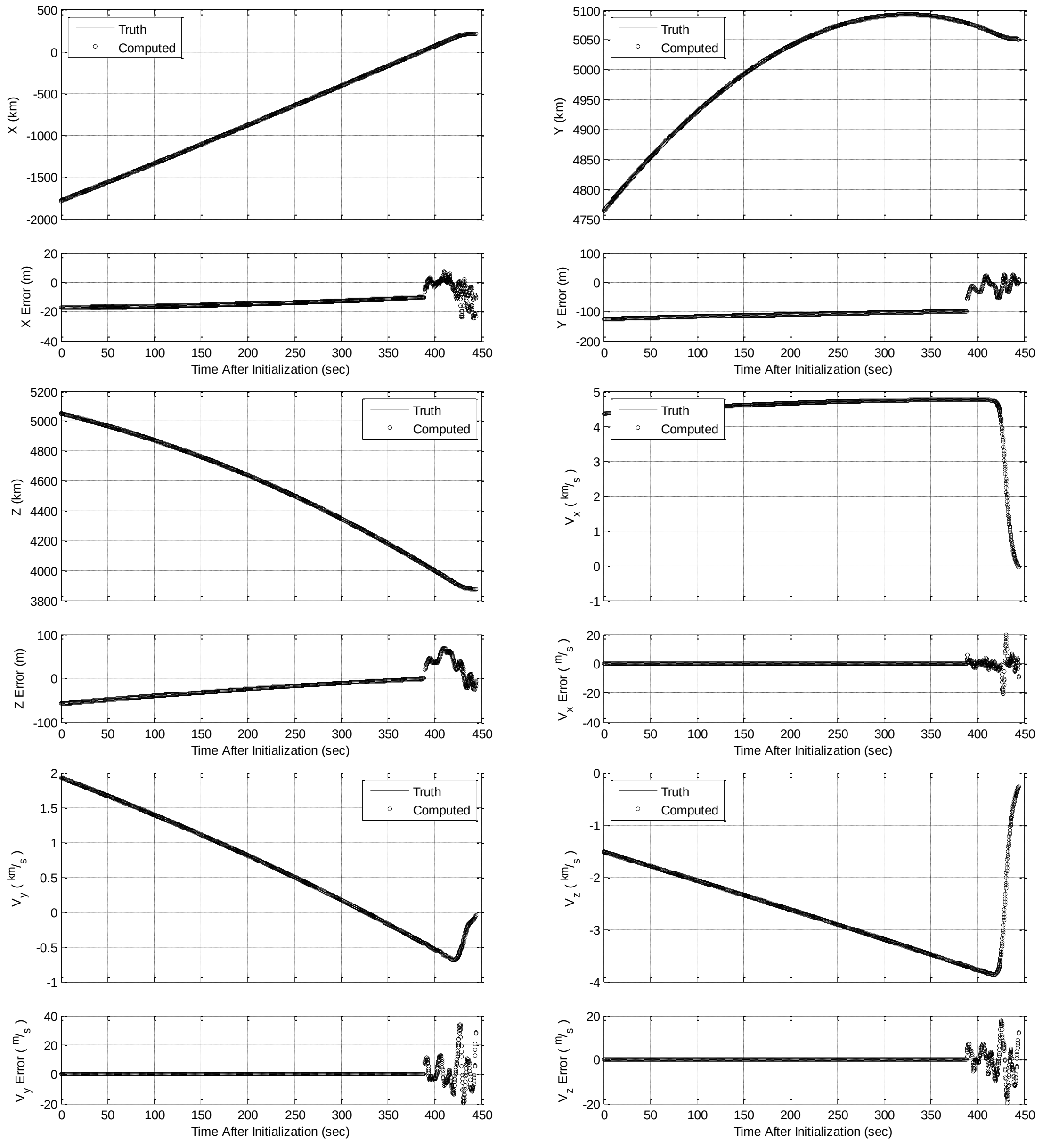
Figure 18 – Trajectory results with addition of sensor 2 at increased rate

Table 13. Trajectory errors with addition of sensor 2 at increased rate

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
14.1	104.4	32.6	1.8	3.8	2.4	2.2	2.6	1.9
Error RMS (m):		63.7	Error RMS $\left(\frac{m}{s}\right)$:		2.8	Error RMS $\left(\frac{m}{s^2}\right)$:		2.3
2Hz Error:		65.6	2Hz Error:		3.5	2Hz Error:		2.1
Difference:		-2.9%	Difference:		-20.0%	Difference:		+9.5%

Similar to the results achieved for increasing the collection rate of a sensor at location 1, the error in position was improved and the error in acceleration was worsened by increasing the collection rate of a sensor at location 2. However, unlike the previous scenario, the error in velocity was improved. Error in position and velocity were improved by 2.9% and 20.0%, respectively, while error in acceleration worsened by 9.5%. This reduction in acceleration accuracy is much less than the 41.5% reduction in acceleration accuracy computed from the previous scenario, but it is unclear why the results are consistently worse in acceleration in these two scenarios.

The last modified scenario will operate a sensor at location 3, similar to the results presented in Figure 15 and Table 10. Results from this scenario are presented in Figure 19 and Table 14.



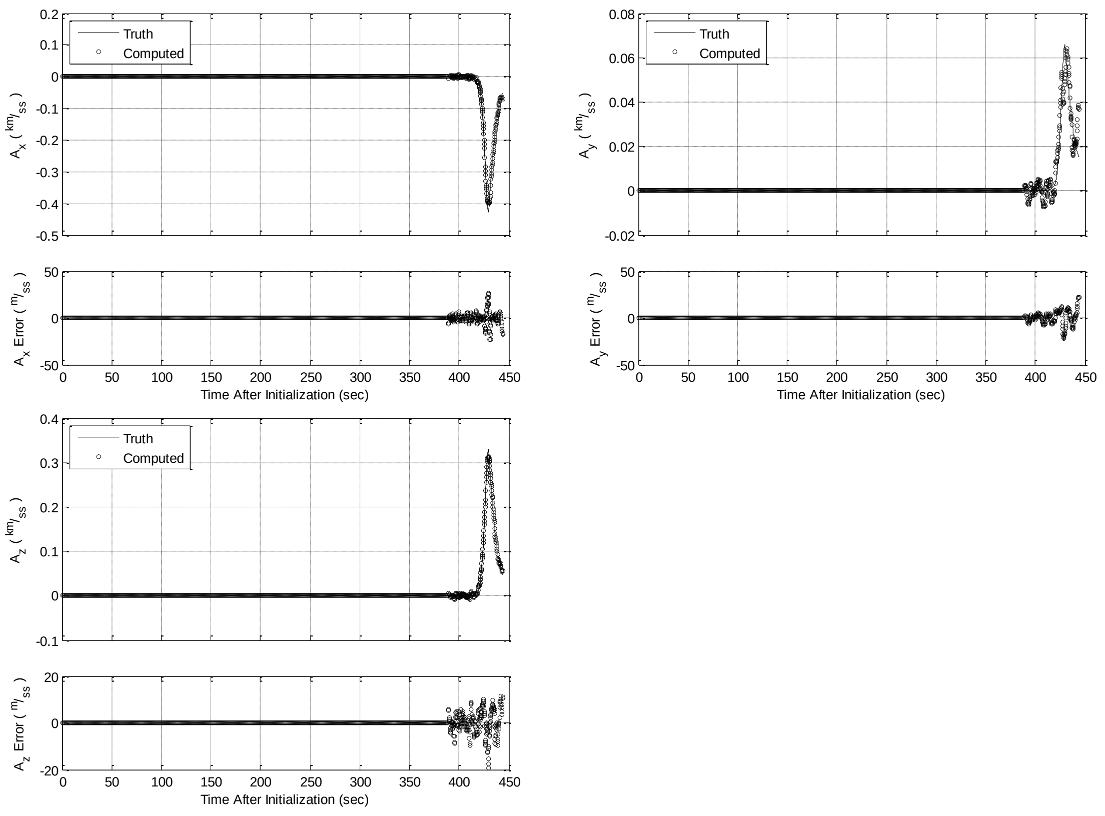


Figure 19 – Trajectory results with addition of sensor 3 at increased rate

Table 14. Trajectory errors with addition of sensor 3 at increased rate

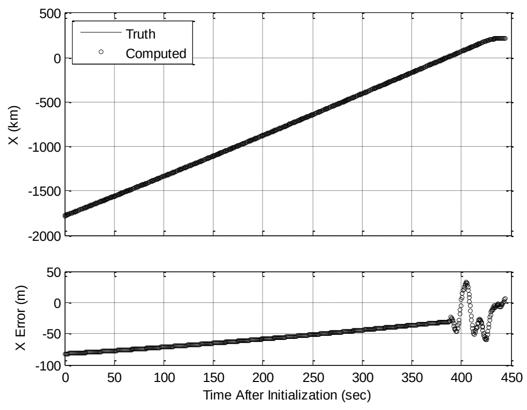
$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
11.8	99.4	28.1	1.5	3.0	2.5	2.0	1.6	1.8
Error RMS (m):		60.0	Error RMS $\left(\frac{m}{s}\right)$:		2.4	Error RMS $\left(\frac{m}{s^2}\right)$:		1.8
2Hz Error:		63.0	2Hz Error:		3.0	2Hz Error:		2.0
Difference:		-4.8%	Difference:		-20.0%	Difference:		-10.0%

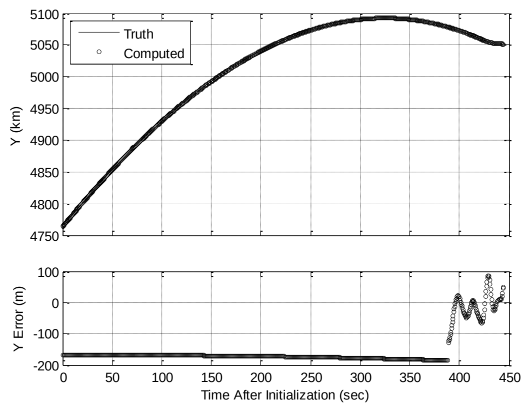
Improving on the results achieved for increasing the collection rate of sensor 2, the results from increasing the collection rate of sensor 3 show improvement in position, velocity, and now acceleration. Errors were improved by 4.8% in position, 20.0% in velocity, and 10.0% in acceleration. The review of these four scenarios suggest that an increase in the collection rate of a sensor can affect the resulting trajectory results either positively or negatively and seem to be highly influenced by the collection geometry. Further analysis into this phenomenon could be carried out focusing on data rates other than those selected here. For many collectors, data rates in excess of 10 or 20 Hz are not unheard of, and could show great improvement over the values presented here.

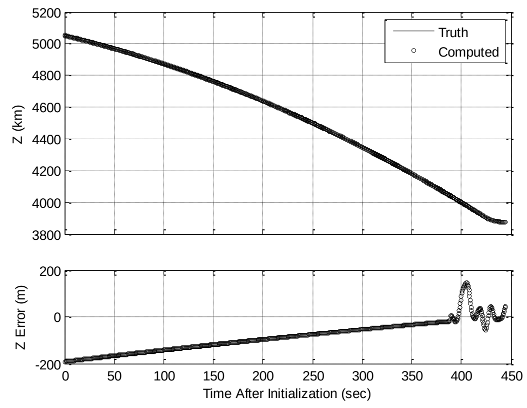
Case 4 – Filter Performance Improvements Through Collection Geometry.

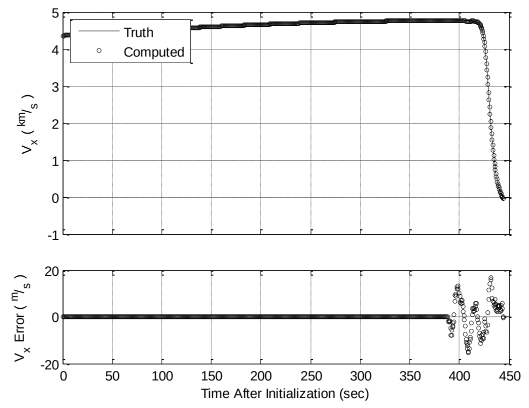
When investigating the effects of increasing the rate of data collection, it was noted that the results showed significant variation depending on the geometry of the collecting sensor. In that case, the addition of a sensor 90 degrees off of the reentry azimuth was of most benefit to the accuracy of the trajectory fit. To further analyze this case, the filter is rerun for the sensors located at locations 1, 2, and 3 without the primary sensor. The results of these three fits can be compared to discover trends in accuracy based solely on a single sensor's collection geometry.

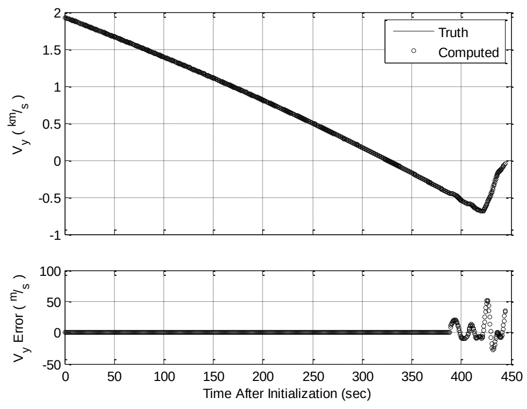
The first scenario reviewed places the sensor at location 1, along the reentry azimuth, operating at the standard data rate of 2 Hz. The results of this scenario are presented in Figure 20 and Table 15.

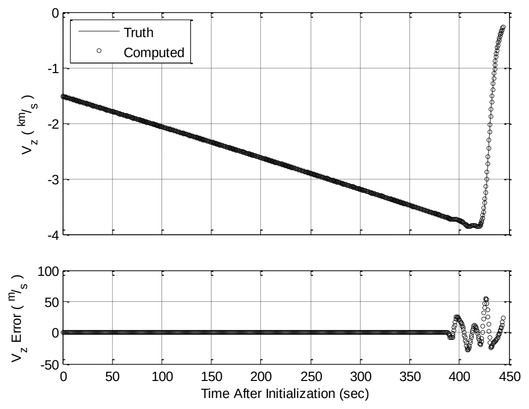

Truth
Computed
X (km)
X Error (m)
Time After Initialization (sec)


Truth
Computed
Y (km)
Y Error (m)
Time After Initialization (sec)


Truth
Computed
Z (km)
Z Error (m)
Time After Initialization (sec)


Truth
Computed
V_x ($^{km}/_{s}$)
V_x Error ($^{m}/_{s}$)
Time After Initialization (sec)


Truth
Computed
V_y ($^{km}/_{s}$)
V_y Error ($^{m}/_{s}$)
Time After Initialization (sec)


Truth
Computed
V_z ($^{km}/_{s}$)
V_z Error ($^{m}/_{s}$)
Time After Initialization (sec)

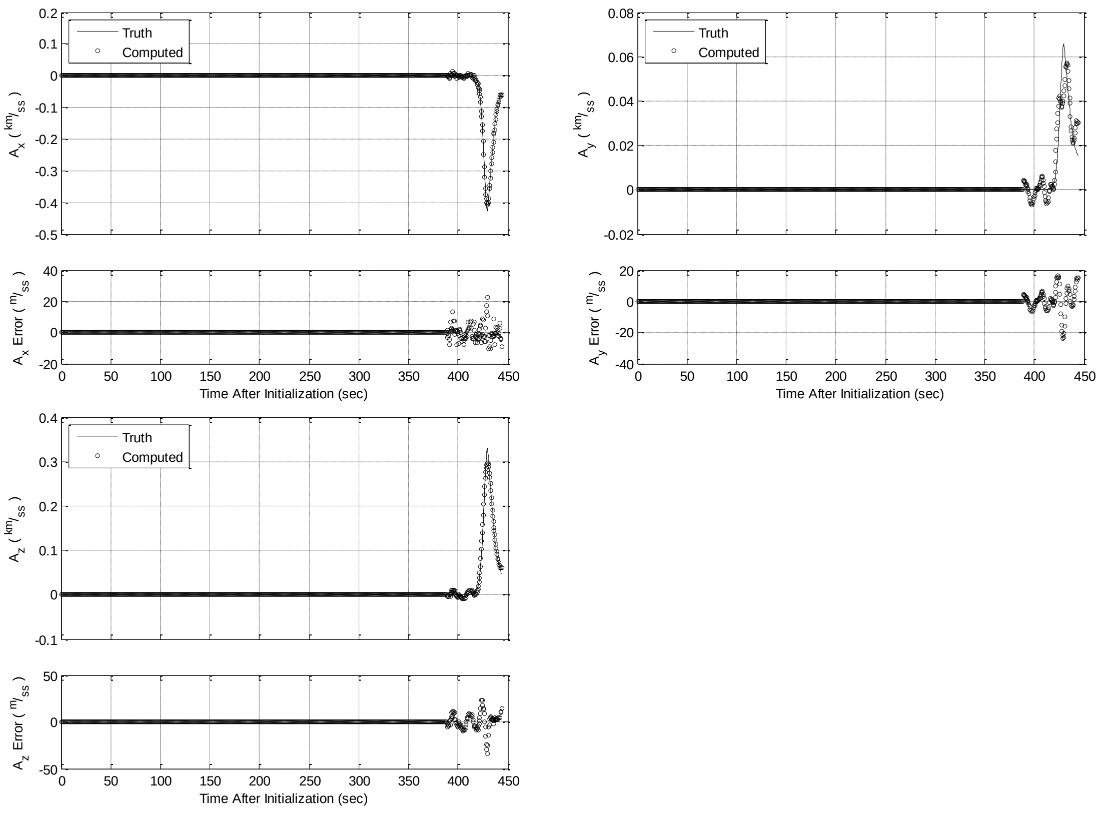


Figure 20 – Trajectory results from sensor 1

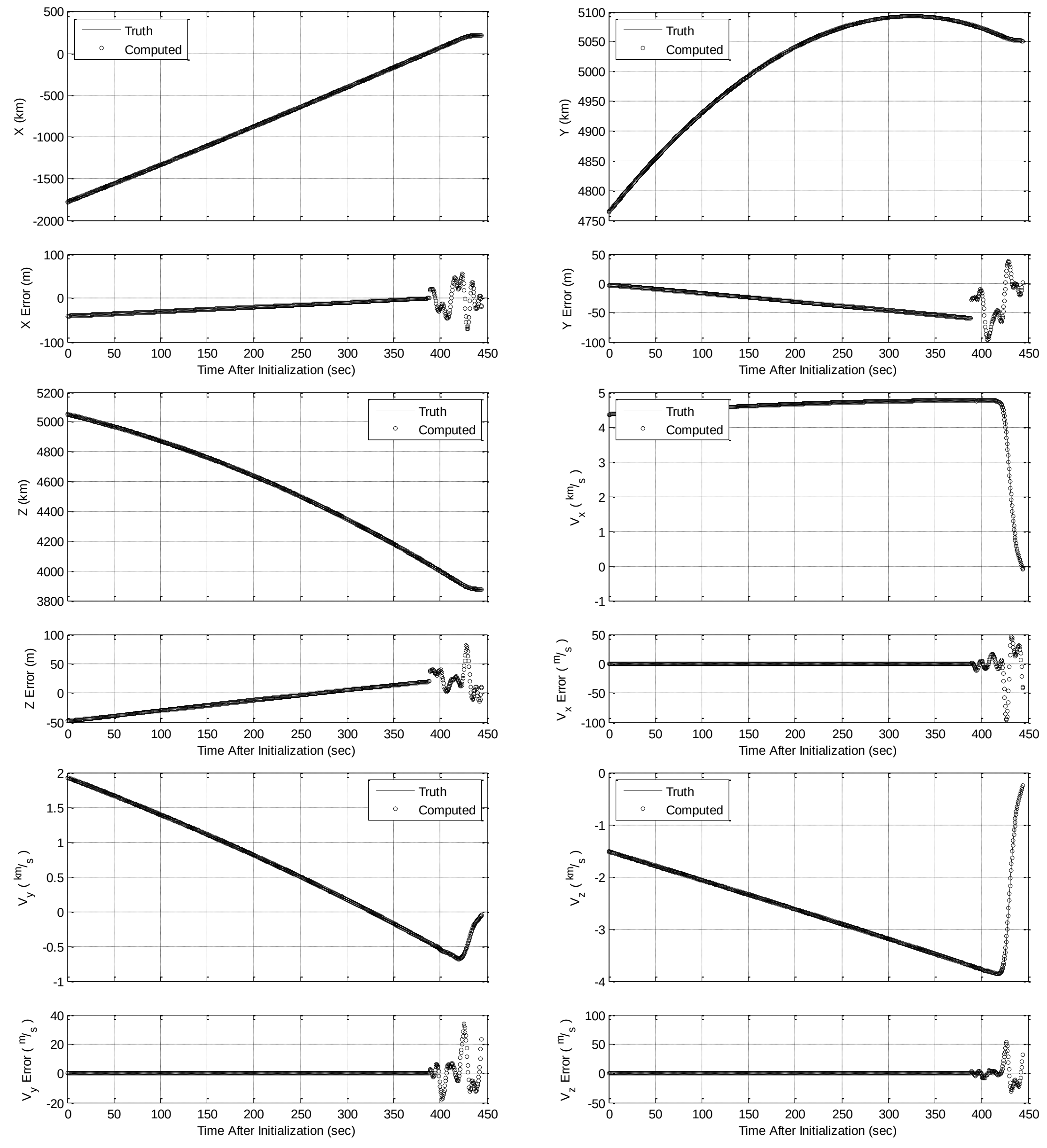
Table 15. Trajectory errors from sensor 1

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
57.5	163.5	107.6	2.7	6.2	6.8	2.0	2.9	3.3
Error RMS (m):		117.8	Error RMS $\left(\frac{m}{s}\right)$:		5.5	Error RMS $\left(\frac{m}{s^2}\right)$:		2.8

The results from this scenario will be used as a baseline for the results achieved when the sensor is placed at locations 2 and 3. Deviations from these results will be used to determine whether collection geometry has a noticeable effect on the accuracy of the reconstructed trajectories.

The next scenario places the sensor at location 2, approximately 45 degrees off of the reentry azimuth, operating at the standard data rate of 2 Hz. The results of this scenario are presented in Figure 21 and

Table 16.



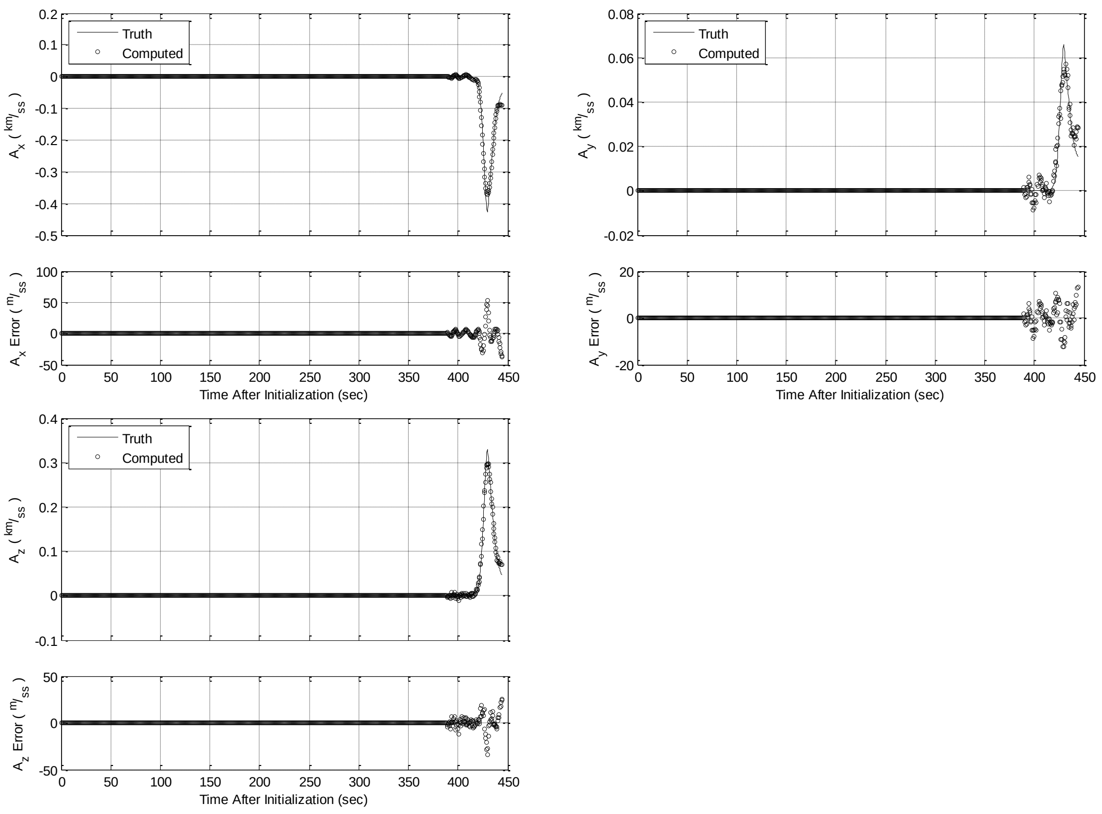


Figure 21 – Trajectory results from sensor 2

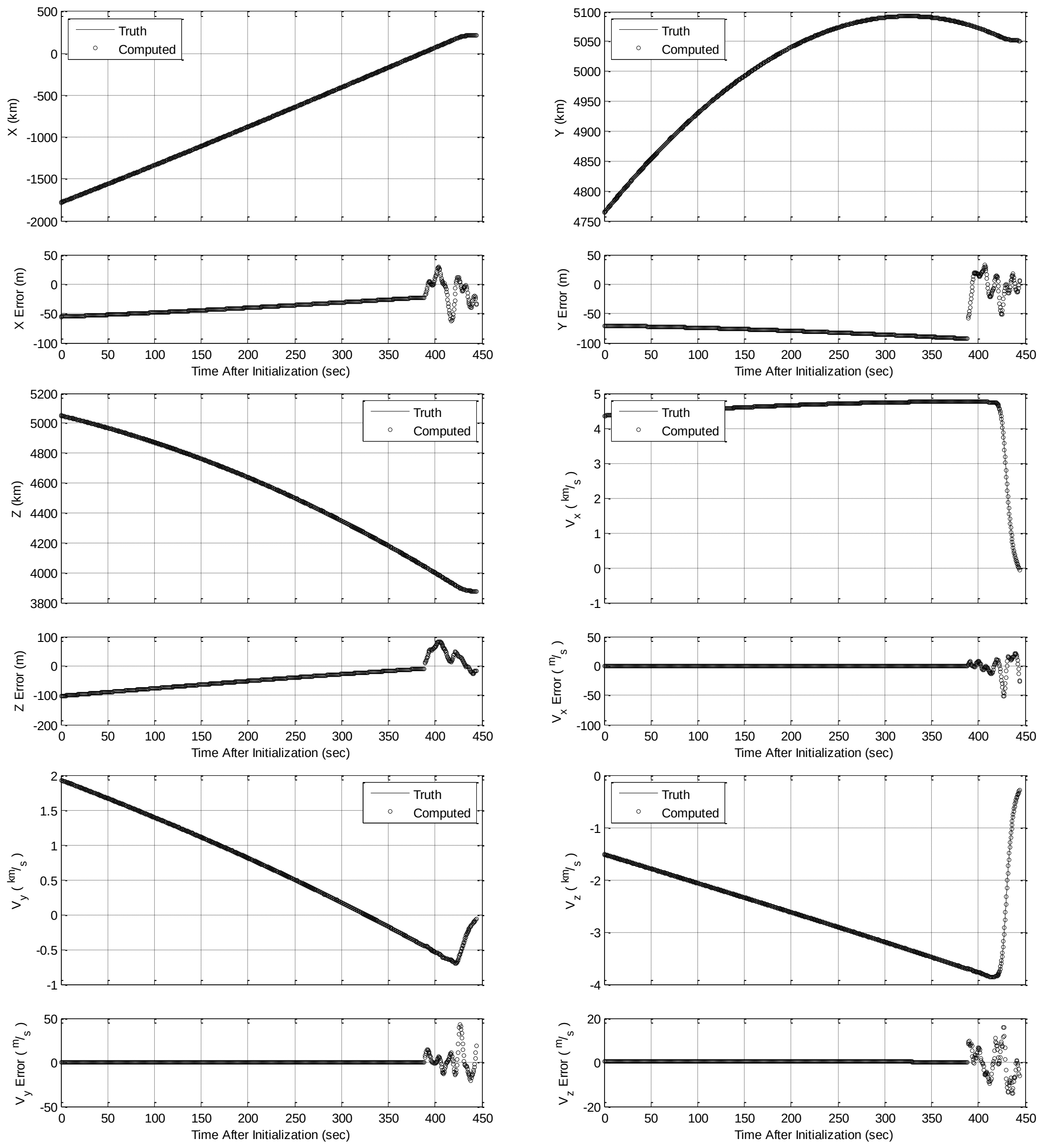
Table 16. Trajectory errors from sensor 2

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
25.3	36.4	24.9	9.8	4.1	5.7	5.1	1.9	3.2
Error RMS (m):		29.4	Error RMS $\left(\frac{m}{s}\right)$:		7.0	Error RMS $\left(\frac{m}{s^2}\right)$:		3.6
Baseline Error:		117.8	Baseline Error:		5.5	Baseline Error:		2.8
Difference:		-75.0%	Difference:		+27.3%	Difference:		+28.6%

With the sensor at location 2, the position accuracy was improved by 75.0% while the velocity and acceleration results both suffered degradations in accuracy, when compared to the truth reference, of 27.3% and 28.6%, respectively.

The other scenario that was tested involved the placement of a sensor at location 3, approximately perpendicular to the reentry azimuth, operating at the standard data rate of 2 Hz. The results from this scenario are presented in Figure 22 and

Table **17**.



Truth
Computed
X (km)
X Error (m)
Y (km)
Y Error (m)
Z (km)
Z Error (m)
V_x ($^{km}/_s$)
V_x Error ($^{m}/_s$)
V_y ($^{km}/_s$)
V_y Error ($^{m}/_s$)
V_z ($^{km}/_s$)
V_z Error ($^{m}/_s$)
Time After Initialization (sec)

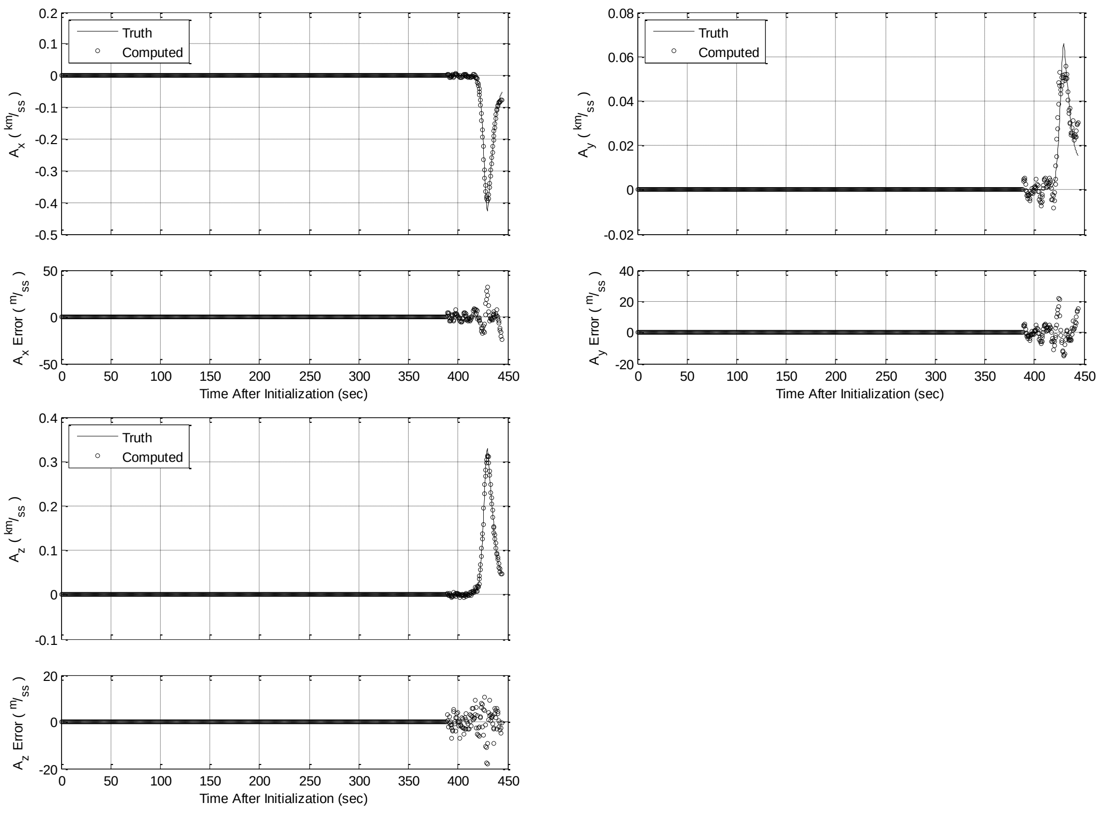


Figure 22 – Trajectory results from sensor 3

Table 17. Trajectory errors from sensor 3

$X\ (m)$	$Y\ (m)$	$Z\ (m)$	$V_X\ \left(\frac{m}{s}\right)$	$V_Y\ \left(\frac{m}{s}\right)$	$V_Z\ \left(\frac{m}{s}\right)$	$A_X\ \left(\frac{m}{s^2}\right)$	$A_Y\ \left(\frac{m}{s^2}\right)$	$A_Z\ \left(\frac{m}{s^2}\right)$
39.6	75.4	58.6	5.3	4.8	2.5	3.0	2.4	1.7
Error RMS (m):		59.7	Error RMS $\left(\frac{m}{s}\right)$:		4.4	Error RMS $\left(\frac{m}{s^2}\right)$:		2.4
Baseline Error:		117.8	Baseline Error:		5.5	Baseline Error:		2.8
Difference:		-49.3%	Difference:		-20.0%	Difference:		-14.3%

As in the previous scenario, the position accuracy was improved over the baseline, this time showing a 49.3% improvement. Furthermore, the velocity and acceleration results for this scenario showed an improvement in accuracy. Velocity and acceleration errors, when compared to the truth reference, were reduced by 20.0% and 14.3%, respectively.

Reviewing these scenarios collectively, it is noted that position errors were reduced in both cases where the sensor was not operating at a location along the reentry azimuth. However, while the position errors showed a trend that encouraged the placement of the sensor farther from the reentry azimuth, the results in velocity and acceleration were mixed and may require further analysis to investigate subtle trends.

Investigative Questions Answered

After validating the filter's performance against different scenarios, several scenario adjustments were investigated to determine any benefits that could be derived. Initially, it was determined that including observations from a second sensor could improve the filter's accuracy, but this varied with the placement of the second sensor. The greatest improvement was derived from placing the sensor perpendicular to the reentry azimuth of the target.

After investigating the effects of a second sensor, the data rate of the collecting sensors were varied. It was determined that increasing the data rate of either the primary or secondary sensors could improve the filter's accuracy. This result was expected as it increases the filter's knowledge of the target during the same time period.

The final scenario modification involved further investigation of sensor geometry. When adding additional sensors to the primary collector, it was noted that the location of the second sensor could vary the resulting filter accuracy. This phenomenon was further investigated by considering data only from the secondary sensor at reduced range. The sensor was operated at the three different collection locations and the resulting filter accuracy was reviewed. In the scenarios investigated, the filter achieved its best accuracy when the single sensor was operated perpendicular to the target's reentry azimuth.

Summary

This chapter investigated the accuracy of the filter by comparing its output to truth data used to generate the observations that were fed to the filter. Several different scenarios were investigated, including adding a second sensor, increasing the data collection rate, and changing the sensor collection geometry. The results of these modifications were reviewed and summarized.

V. Conclusions and Recommendations

Chapter Overview

This chapter will cover the general conclusions that were drawn from the analysis section of the paper. Additionally, recommendations for future action and research are presented.

Conclusions of Research

The filter developed for this thesis combined the strengths of the least squares and Kalman filters. The least squares filter operates rapidly and accurately on the free-flight portions of flight. The Kalman filter provides greater flexibility for the state's acceleration components to vary lower in the atmosphere. The filter successfully transitioned from the least squares to Kalman filter, using the final values of the free flight propagation for the Kalman filter's initial state.

The developed algorithm includes a least squares sliding window filter that estimates an initial guess of position during the maneuvering phase of flight. After investigation of the effects of computing these initial guesses, it was determined that they consistently had a detrimental effect on the filter estimates. After this was concluded, the least squares sliding window was not implemented for the results presented in Chapter IV.

Once the filter was validated against both maneuvering and non-maneuvering targets, the filter was used to investigate other collection scenario modifications and their effects. The filter achieved varying levels of accuracy when the scenario was modified with a different number of sensors, increasing data collection rates, and different collection geometries.

After investigating these scenarios, it was determined that the best results were achieved with additional collectors, by increasing the data collection rate, and by moving the collector position perpendicular to the reentry azimuth.

Significance of Research

This research expands on the work of previous Air Force Institute of Technology (AFIT) graduate students' work in the area of data filtering of collections of reentry vehicles. Whereas previous research only addressed non-maneuvering or simple maneuvering targets with fixed bank angles, this work allows for the study of complex maneuvering targets with varying accelerations.

Of further significance, the filter algorithm allows for the inclusion of multiple sensors. Including all of the available data into the filter estimates ensures that the best results can be achieved.

Recommendations for Action

While several scenarios were investigated to determine the effects of modifications to the number of collectors, their data rates, and their collection geometry, more work could be done to further investigate these areas. The accuracy of the filter results varied significantly in all scenarios, so subtler investigation could identify trends and true optimums. Error contour plots could be generated for different scenarios in order to better illustrate results.

Additionally, further modifications could be investigated beyond those introduced here. The algorithm was written to accommodate multiple targets simultaneously, but is not investigated here. Another major area that could be investigated is the benefit of range, azimuth, and elevation sensors over sensors that only measure azimuth and elevation. The algorithm is written to manage both of these types of sensors and this could be a major area of investigation.

Recommendations for Future Research

As discussed in the filter performance section of the analysis and conclusions, the least squares filter is not performing as well on the non-maneuvering portion of the trajectory as the Kalman filter is performing on the lower regions of flight. There may be ways of improving this performance. Areas to investigate could include modifications to the numerical partial derivatives that are used. These could either be altered with different perturbation sizes or replaced by analytical solutions to the partials. One possibility could be adjusting the perturbation magnitudes as the targets approach convergence. If this portion of the filter performed better, the overall performance would likely be greatly improved.

Other modifications that could be researched include the inclusion of further data types. The algorithm as written addresses two data types: range, azimuth, and elevation sensors and azimuth and elevation-only sensors. There are situations where additional data types could be available and it would be a benefit to incorporate these data types into the analysis. One easy data type that could be included would be GPS or other position information obtained from the operator of the test. Another data type that could be included would be right-ascension and declination angle measurements that are typical of overhead sensors. This data type could easily be incorporated due to its similarity to the azimuth and elevation data type.

Summary

This chapter reviewed the research and offered some general conclusions that were derived and suggested future work that could be performed. In the previous chapters, the filter is derived and validated against various scenarios. Future work could be done in either the areas of collection optimization using the filter as it currently exists, or filter modification to either improve performance or accommodate data types from sensor types that were not considered for this thesis.

Appendix

Coordinate Frames

All computation is done in an Earth-Centered Inertial (ECI) reference frame when possible, most notably in the EOMs which, when expressed in ECI are simplified to those in Equations (6-11). When this is not possible, as in the case of Equation (32), it is important to realize the coordinate frames being referenced. This section does not define the algorithms to transform from one frame to another, but simply defines the reference frames.

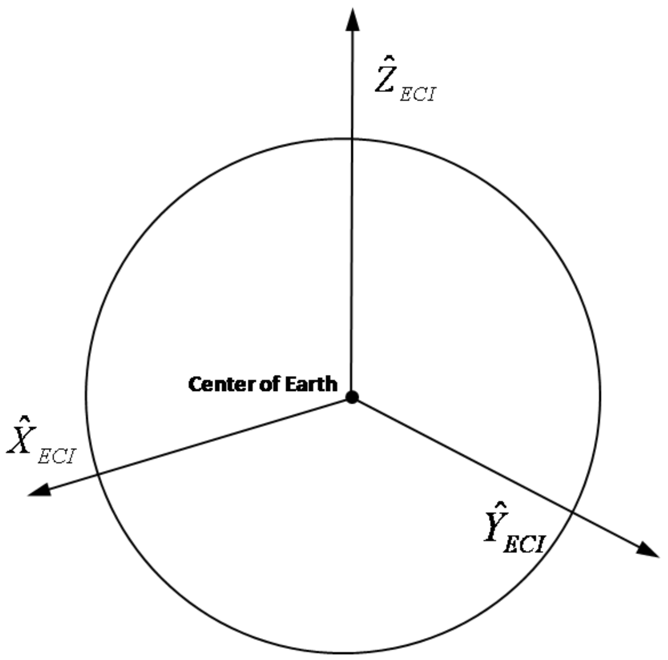


Figure 23 – Earth-Centered Inertial (ECI) coordinate frame

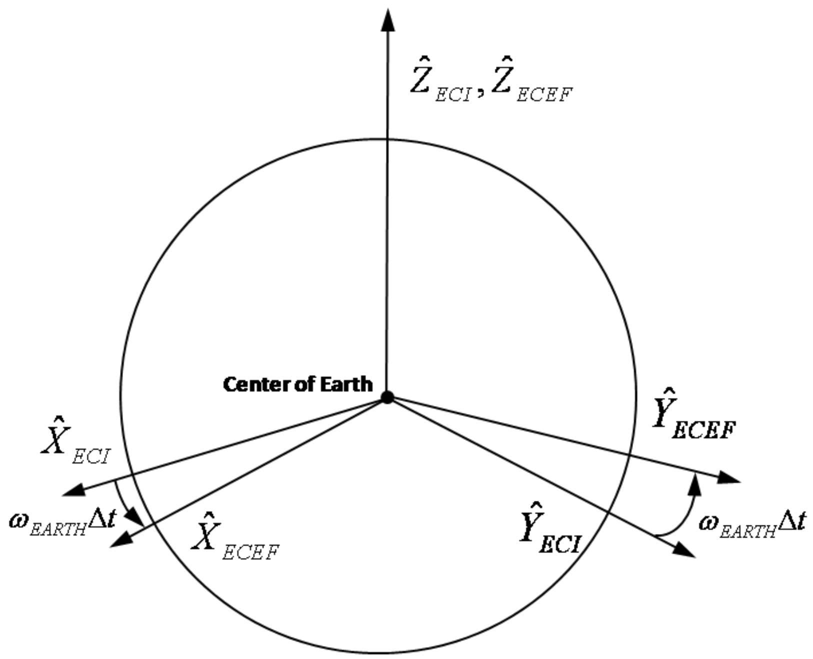


Figure 24 – Earth-Centered Earth-Fixed (ECEF) coordinate frame

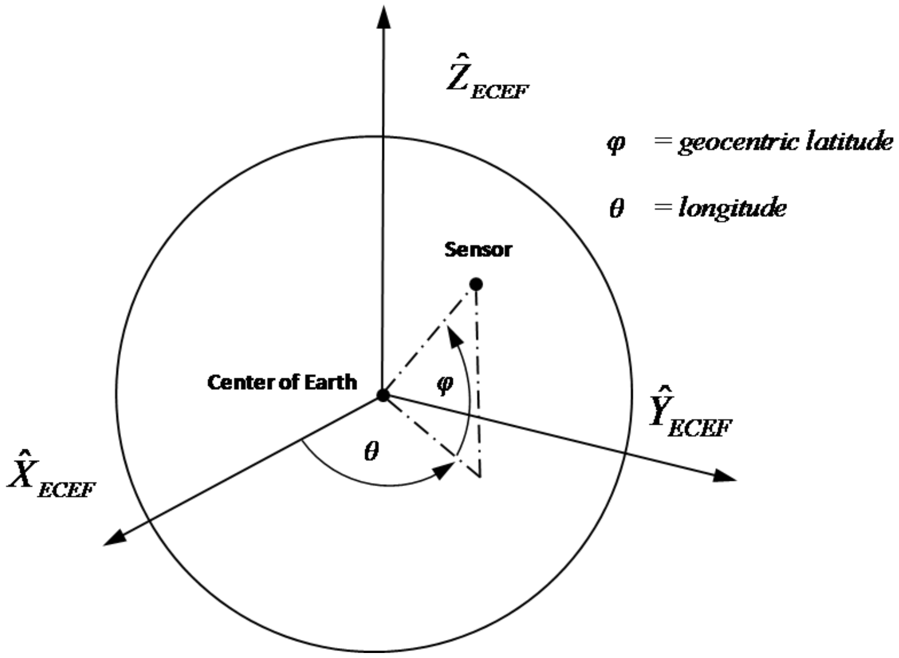


Figure 25 – Latitude/Longitude coordinate frame

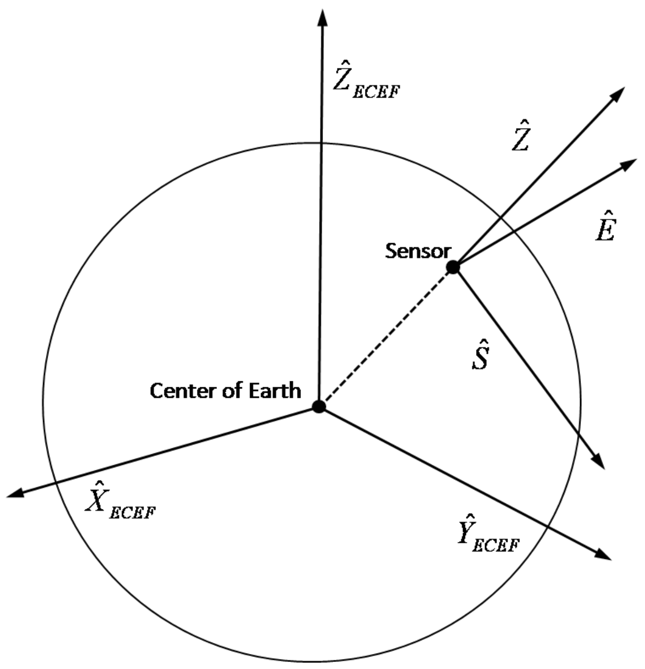


Figure 26 – South, East, Up (SEZ) coordinate frame

kaliper.m

`kaliper.m` is the main filter code written in the MATLAB scripting language.

Original formatting is preserved to maintain functionality when pasted into MATLAB.

```
function varargout = kaliper(varargin)
%KALIPER - Kalman filter And Least squares Integrated Parameter
Estimation Routine
%
% -- Usage --
%  state                 = kaliper(target, sensor, koptions)
% [state, target]        = kaliper(target, sensor, koptions)
% [state, target, stats] = kaliper(target, sensor, koptions)
%
% -- Input definition --
% target - data structure defining observed targets of the form:
%   target{tgt}.obs       - metric observations (numobs x 3)
%   target{tgt}.obs_time  - metric observation times (numobs x 1)
%   target{tgt}.obs_snr   - indices of sensors for all obs (numobs x 1)
%   target{tgt}.obs_ff    - logical array specifying free flight obs
(numobs x 1)
%   target{tgt}.numobs    - number of metric observations (1)
%   target{tgt}.init_time - time of initial guess state vector
%   target{tgt}.init_sv   - initial guess state vector (1x10)
%
% sensor - data structure defining observing sensors of the form:
%   sensor{snr}.snr_type      - sensor type, one of:
%                                1: stationary ranged (range,az,el)
%                                2: stationary two-angle (az,el)
%                                3: moving range (range,az,el)
%                                4: moving two-angle (az,el)
%   sensor{snr}.pos_lla       - LLA sensor position (1x3) OR (numeph x
3)
%                               (geod lat, lon, alt) - (rad, rad, km)
%   sensor{snr}.pos_ecf       - ECEF sensor position (1x3) OR (numeph x
3)
%   sensor{snr}.pos_time      - moving sensor ephemeris times (numeph x
1)
%   sensor{snr}.tm            - SEZ to ECEF rot matrix (3x3) OR (numeph
x 9)
%   sensor{snr}.stddev        - standard deviations of obs (1x3)
%   sensor{snr}.obs_bias      - constant observation biases (1x3)
%
% koptions - data structure defining kaliper run options (Optional)
%   koptions.echo             - true/false flag to echo status to
command (1)
%                               {default value = false(1)}
%   koptions.sig_edit         - number of std deviations for editting
(1)
%                               {default value = 3}
%   koptions.max_iter         - maximum number of least squares
iterations (1)
%                               {default value = 40}
%   koptions.conv_tol         - convergence tolerance for least squares
(1)
%                               {default value = .0001}
%   koptions.lssw             - true/false flag to use LSSW results in
KF (1)
%                               {default value = true(1)}
%   koptions.window_size      - sliding window nominal size for LSSW
pass (1)
%                               {default value = 30}
%   koptions.min_window_size  - sliding window minimum size for LSSW
initialization (1)
%                               {default value = 5}
%   koptions.max_window_size  - sliding window maximum size for LSSW
termination (1)
%                               {default value = 40}
%   koptions.qp               - KF position plant noise scale factor
(1)
%                               {default value = 0.0}
%   koptions.qv               - KF velocity plant noise scale factor
(1)
%                               {default value = 0.0}
%   koptions.qa               - KF acceleration plant noise scale
factor (1)
%                               {default value = 0.001}
%
%
% -- Output definition --
% state - data structure defining state vector components
%   state{tgt}.time           - metric observation times (numobs x 1)
%   state{tgt}.sv             - state vector solution (numobs x 9)
%                {1-3}        - ECI positions (km), ECI epoch @ 0 GMT
day of collect
%                {4-6}        - ECI velocities (km/s)
%                {7-9}        - ECI sensed accelerations (km/s2) (total
accel - gravity)
%   state{tgt}.sv_smooth      - smoothed state vector solution (numobs
x 9)
%                {1-3}        - ECI positions (km), ECI epoch @ 0 GMT
day of collect
%                {4-6}        - ECI velocities (km/s)
%                {7-9}        - ECI sensed accelerations (km/s2) (total
%                accel - gravity)
%
% target - if requested, target structure is returned with updated
parameters
%
% stats - data structure detailing estimation statistics
%   stats{iter,tgt}.cost_func - Weighted cost function of target by
iteration
%   stats{iter,tgt}.cov       - covariance update computed by target
and iteration
```

```
%   stats{iter,tgt}.res{snr}  - residual vectors by iteration, target,
and sensor

% variables persistent to this function for subsequent calls
persistent target sensor koptions earth

% define earth parameters for later use
earth = define_earth;

% define atmospheric parameters
read_stdatmos('stdatmos76.dat')

% accept input
if nargin == 2
    target = varargin{1};
    sensor = varargin{2};
    % options undefined, will be filled with defaults
    koptions = [];
elseif nargin == 3
    target = varargin{1};
    sensor = varargin{2};
    koptions = varargin{3};
else
    error('Input should either be target & sensor or target, sensor,
and koptions')
end

% fill undefined components of the options structure with default
values
if isempty(koptions)
    koptions.echo = false(1);
    koptions.sig_edit = 3;
    koptions.max_iter = 40;
    koptions.conv_tol = .0001;
    koptions.lssw = true(1);
    koptions.window_size = 30;
    koptions.min_window_size = 5;
    koptions.max_window_size = 40;
    koptions.qp = 0;
    koptions.qv = 0;
    koptions.qa = 0.001;
else
    if ~isfield(koptions, 'echo') || isempty(koptions.echo)
        koptions.echo = false(1);
    end
    if ~isfield(koptions, 'sig_edit') || isempty(koptions.sig_edit)
        koptions.sig_edit = 3;
    end
    if ~isfield(koptions, 'max_iter') || isempty(koptions.max_iter)
        koptions.max_iter = 40;
    end
    if ~isfield(koptions, 'conv_tol') || isempty(koptions.conv_tol)
        koptions.conv_tol = .0001;
    end
    if ~isfield(koptions, 'lssw') || isempty(koptions.lssw)
        koptions.lssw = true(1);
    end
    if ~isfield(koptions, 'window_size') ||
isempty(koptions.window_size)
        koptions.window_size = 30;
    end
    if ~isfield(koptions, 'min_window_size') ||
isempty(koptions.min_window_size)
        koptions.min_window_size = 5;
    end
    if ~isfield(koptions, 'max_window_size') ||
isempty(koptions.max_window_size)
        koptions.max_window_size = 40;
    end
    if ~isfield(koptions, 'qp') || isempty(koptions.qp)
        koptions.qp = 0;
    end
    if ~isfield(koptions, 'qv') || isempty(koptions.qv)
        koptions.qv = 0;
    end
    if ~isfield(koptions, 'qa') || isempty(koptions.qa)
        koptions.qa = 0.001;
    end
end

% optional plot of residuals
if koptions.echo
    figure;
    resax1 = subplot(3,1,1);
    resax2 = subplot(3,1,2);
    resax3 = subplot(3,1,3);
end

% number of targets in structure
numtgt = length(target);

% time pad to add to observations at the same time value
time_pad = 1e-10;

% loop through targets
for tgt = 1:numtgt

    % number of observations of this target
    numobs = length(target{tgt}.obs_time);

    % number of sensors for this target
    tgtsnr = unique(target{tgt}.obs_snr);
    numsnr = length(tgtsnr);

    % add small time intervals on to any observations at the same time
    repeat_idx = diff(target{tgt}.obs_time) == 0;
    repeat_idx = [false(1); repeat_idx];
%#ok<AGROW>
```

```
    while any(repeat_idx)
        target{tgt}.obs_time(repeat_idx) =
target{tgt}.obs_time(repeat_idx) + time_pad;
        repeat_idx = diff(target{tgt}.obs_time) == 0;
        repeat_idx = [false(1); repeat_idx];
%#ok<AGROW>
    end

    % convert 3-D observations to ECI frame for SV initial guess
    target{tgt}.obs_eci = repmat(NaN, numobs, 3); % NaN padding

    % locate 3-D observations of stationary and moving sensors
    for snridx = 1:numsnr

        % index of this sensor
        snr = tgtsnr(snridx);

        % observations from this sensor, and those observation times
        obsidx = target{tgt}.obs_snr == snr;
        obs_time = target{tgt}.obs_time(obsidx);

        % stationary range, az, el observations
        if sensor{snr}.snr_type == 1
            obs_sez = tm_rae2sez(target{tgt}.obs(obsidx,:));
            obs_ecf = tm_sez2ecr(obs_sez, sensor{snr}.pos_ecf,
sensor{snr}.tm);
        % moving range, az, el observations
        elseif sensor{snr}.snr_type == 3
            obs_sez = tm_rae2sez(target{tgt}.obs(obsidx,:));

            % interpolate sensor positions at observation times
            interp_lat = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,1), obs_time, 'spline');
            interp_lon = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,2), obs_time, 'spline');
            interp_alt = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,3), obs_time, 'spline');
            [sensor_pos, sensor_tm] = calc_sensor_move(interp_lat,
interp_lon, interp_alt);

            % moving transformation
            obs_ecf = tm_sez2ecr_move(obs_sez, sensor_pos, sensor_tm);
        else
        % sensor without 3-D observation, move to next sensor
            continue;
        end

        % convert ecef to eci
        obs_eci = tm_ecr2eci(obs_ecf, 0, obs_time, 0);

        % insert transformation into structure
        target{tgt}.obs_eci(obsidx,:) = obs_eci;
    end
    % check if initial guess has been specified, otherwise compute it
    if ~isfield(target{tgt}, 'init_sv') ||
~isempty(target{tgt}.init_sv)
        [init_time, init_sv] = kaliper_calcinit(tgt);
        target{tgt}.init_time = init_time;
        target{tgt}.init_sv = init_sv;
    end
end

%%%%
%%%% Phase 1 state vector estimation - Free Flight - Sensed Accels = 0
%%%%

if koptions.echo
    disp('** Entering Phase 1 - Free-Flight Estimation **')
end

% set free flight ode45 options
options = odeset('RelTol', 1e-6, 'Vectorized', 'on');

% initialize least squares run
ls_iter = 1;
converged = false(size(target));

% begin estimation
while any(~converged) && ls_iter <= koptions.max_iter

    if koptions.echo
        disp([' * Iteration ' num2str(ls_iter)])
    end

    for tgt = 1:numtgt

        if koptions.echo
            disp(['  -Target ' num2str(tgt)])
        end

        % initialize running sums for this target
        TtQiT = zeros(6);
        TtQir = zeros(6,1);
        cost_func = 0;

        % time series to propagate to, first value is time of SV
        prop_time = [target{tgt}.init_time;
target{tgt}.obs_time(target{tgt}.obs_ff)];
        if prop_time(1) == prop_time(2), prop_time(1) = []; end

        % propagate SV to all FF observation times, ignoring any sensed
accelerations
        [calc_time,calc_sv] = ode45(@kaliper_eom, prop_time,
target{tgt}.init_sv', options);
```

```
        % propagate perturbed SVs for numerical partial derivates to
form H matrix
        sv_pert = [.001 .001 .001 .0001 .0001 .0001]; % accelerations
not perturbed
        pert_init_sv = repmat(target{tgt}.init_sv, 6, 1) +
[diag(sv_pert) zeros(6,3)];
        [pert_time,pert_sv] = ode45(@kaliper_eom, prop_time,
pert_init_sv', options);

        % transform propagated SVs to sensor data reference, compute
TtQiT and TtQir
        for snridx = 1:numsnr

            % index of this sensor
            snr = tgtsnr(snridx);

            % observations from this sensor, and those observation
times
            obsidx = target{tgt}.obs_snr(target{tgt}.obs_ff) == snr;
            %obs_time =
target{tgt}.obs_time(target{tgt}.obs_ff(obsidx));
            obs_time = target{tgt}.obs_time(target{tgt}.obs_ff);
            obs_time = obs_time(obsidx);
            numobs = length(obs_time);

            if numobs == 0, continue, end

            % stationary range, az, el OR az, el observations
            if sensor{snr}.snr_type == 1 || sensor{snr}.snr_type == 2

                % convert ECI SV positions to stationary RAE
                calc_eci = calc_sv(obsidx,1:3);
                calc_ecf = tm_eci2ecr(calc_eci, 0, obs_time, 0);
                calc_sez = tm_ecr2sez(calc_ecf, sensor{snr}.pos_ecf,
sensor{snr}.tm);
                calc_rae = tm_sez2rae(calc_sez);

                % same for perturbed states
                pert_eci = pert_sv(obsidx,:);
                pert_eci = reshape(pert_eci', 9, [])';  % 1 vec / row
                pert_ecf = tm_eci2ecr(pert_eci(:,1:3), 0, ...
                    reshape(repmat(obs_time, 1, 6)', [], 1), 0);
                pert_sez = tm_ecr2sez(pert_ecf, sensor{snr}.pos_ecf,
sensor{snr}.tm);
                pert_rae = tm_sez2rae(pert_sez);

                % moving range, az, el OR az, el observations
            elseif sensor{snr}.snr_type == 3 || sensor{snr}.snr_type ==
4

                % interpolate sensor positions at observation times
                interp_lat = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,1), obs_time, 'spline');
```

```
                interp_lon = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,2), obs_time, 'spline');
                interp_alt = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,3), obs_time, 'spline');
                [sensor_pos, sensor_tm] = calc_sensor_move(interp_lat,
interp_lon, interp_alt);

                % convert ECI SV positions to moving RAE
                calc_eci = calc_sv(obsidx,1:3);
                calc_ecf = tm_eci2ecr(calc_eci, 0, obs_time, 0);
                calc_sez = tm_ecr2sez_move(calc_ecf, sensor_pos,
sensor_tm);
                calc_rae = tm_sez2rae(calc_sez);

                % same for perturbed states
                pert_eci = pert_sv(obsidx,:);
                pert_eci = reshape(pert_eci', 9, [])';  % 1 vec / row
                pert_ecf = tm_eci2ecr(pert_eci(:,1:3), 0, ...
                    reshape(repmat(obs_time, 1, 6)', [], 1), 0);
                pert_sez = tm_ecr2sez_move(pert_ecf, sensor_pos,
sensor_tm);
                pert_rae = tm_sez2rae(pert_sez);

            else
                error(['Unrecognized sensor type ID: '
num2str(sensor{snr}.snr_type)])
            end

            % use calculated RAE values to compute residuals vectors
            if sensor{snr}.snr_type == 1 || sensor{snr}.snr_type == 3
                % range, azimuth, and elevation observations
                obs_rae = target{tgt}.obs(target{tgt}.obs_ff,:);
                res = obs_rae(obsidx,:) - calc_rae;
                % save residuals
                stats{ls_iter,tgt}.res{snr} = res;
%#ok<AGROW>

                % plot residuals
                if koptions.echo
                    if snr == 1
                        cla(resax1), plot(resax1, obs_time, res(:,1),
'r.'), hold(resax1, 'on')
                        cla(resax2), plot(resax2, obs_time, res(:,2),
'r.'), hold(resax2, 'on')
                        cla(resax3), plot(resax3, obs_time, res(:,3),
'r.'), hold(resax3, 'on')
                        drawnow
                    else
                        cols = 'rbmk'; nc = length(cols);
                        plot(resax1, obs_time, res(:,1), [cols(mod(snr-
1,nc)+1) '.'])
                        plot(resax2, obs_time, res(:,2), [cols(mod(snr-
1,nc)+1) '.'])
```

```
                        plot(resax3, obs_time, res(:,3), [cols(mod(snr-
1,nc)+1) '.'])
                        drawnow
                    end
                end

                % statistical editting of outliers based on sensor std
dev
                if ls_iter > 1
                    res( abs(res(:,1)) >
sensor{snr}.stddev(1)*koptions.sig_edit, 1) = 0;
                    res( abs(res(:,2)) >
sensor{snr}.stddev(2)*koptions.sig_edit, 2) = 0;
                    res( abs(res(:,3)) >
sensor{snr}.stddev(3)*koptions.sig_edit, 3) = 0;
                end

                % mark editted points
                if koptions.echo
                    edit1 = abs(stats{ls_iter,tgt}.res{snr}(:,1)) >
sensor{snr}.stddev(1)*koptions.sig_edit;
                    edit2 = abs(stats{ls_iter,tgt}.res{snr}(:,2)) >
sensor{snr}.stddev(2)*koptions.sig_edit;
                    edit3 = abs(stats{ls_iter,tgt}.res{snr}(:,3)) >
sensor{snr}.stddev(3)*koptions.sig_edit;
                    if any(edit1), plot(resax1, obs_time(edit1),
stats{ls_iter,tgt}.res{snr}(edit1,1), 'kx'), end
                    if any(edit2), plot(resax2, obs_time(edit2),
stats{ls_iter,tgt}.res{snr}(edit2,2), 'kx'), end
                    if any(edit3), plot(resax3, obs_time(edit3),
stats{ls_iter,tgt}.res{snr}(edit3,3), 'kx'), end
                end

                % cost function update
                cost_func = cost_func + ...
                    sqrt( sum( res(:,1).^2 ) ) / sensor{snr}.stddev(1)
+ ...
                    sqrt( sum( res(:,2).^2 ) ) / sensor{snr}.stddev(2)
+ ...
                    sqrt( sum( res(:,3).^2 ) ) / sensor{snr}.stddev(3);

            elseif sensor{snr}.snr_type == 2 || sensor{snr}.snr_type ==
4
                % azimuth and elevation observations
                obs_ae = target{tgt}.obs(target{tgt}.obs_ff,2:3);
                res = obs_ae(obsidx,:) - calc_rae(:,2:3);
                % save residuals
                stats{ls_iter,tgt}.res{snr} = res;
%#ok<AGROW>

                % plot residuals
                if koptions.echo
                    if snr == 1
```

```
                        cla(resax2), plot(resax2, obs_time, res(:,1),
'r.'), hold(resax2, 'on')
                        cla(resax3), plot(resax3, obs_time, res(:,2),
'r.'), hold(resax3, 'on')
                        drawnow
                    else
                        cols = 'rbmk'; nc = length(cols);
                        plot(resax2, obs_time, res(:,1), [cols(mod(snr-
1,nc)+1) '.'])
                        plot(resax3, obs_time, res(:,2), [cols(mod(snr-
1,nc)+1) '.'])
                        drawnow
                    end
                end

                % statistical editting of outliers based on sensor std
dev
                if ls_iter > 1
                    res( abs(res(:,1)) >
sensor{snr}.stddev(1)*koptions.sig_edit, 1) = 0;
                    res( abs(res(:,2)) >
sensor{snr}.stddev(2)*koptions.sig_edit, 2) = 0;
                end

                % mark editted points
                if koptions.echo
                    edit1 = abs(stats{ls_iter,tgt}.res{snr}(:,1)) >
sensor{snr}.stddev(1)*koptions.sig_edit;
                    edit2 = abs(stats{ls_iter,tgt}.res{snr}(:,2)) >
sensor{snr}.stddev(2)*koptions.sig_edit;
                    if any(edit1), plot(resax2, obs_time(edit1),
stats{ls_iter,tgt}.res{snr}(edit1,1), 'kx'), end
                    if any(edit2), plot(resax3, obs_time(edit2),
stats{ls_iter,tgt}.res{snr}(edit2,2), 'kx'), end
                end

                % cost function update
                cost_func = cost_func + ...
                    sqrt( sum( res(:,1).^2 ) ) / sensor{snr}.stddev(1)
+ ...
                    sqrt( sum( res(:,2).^2 ) ) / sensor{snr}.stddev(2);

                % remove computed range values
                calc_rae(:,1) = [];
                pert_rae(:,1) = [];
            else
                error(['Unrecognized sensor type ID: '
num2str(sensor{snr}.snr_type)])
            end

            % resize residuals matrix to a vector
            res = reshape(res',[],1);
            % use perturbed trajectories to compute observation
matrices (Ti)
            %
            % each observation time has as many entries as is has
            % components of range, azimuth, or elevation
            %  (Range, Az, & El - 3 entries per obs time)
            %  (Az & El         - 2 entries per obs time)
            %
            partials_x = calc_rae - pert_rae(1:6:end,:);
            partials_x = reshape(partials_x', [], 1);

            partials_y = calc_rae - pert_rae(2:6:end,:);
            partials_y = reshape(partials_y', [], 1);

            partials_z = calc_rae - pert_rae(3:6:end,:);
            partials_z = reshape(partials_z', [], 1);

            partials_vx = calc_rae - pert_rae(4:6:end,:);
            partials_vx = reshape(partials_vx', [], 1);

            partials_vy = calc_rae - pert_rae(5:6:end,:);
            partials_vy = reshape(partials_vy', [], 1);

            partials_vz = calc_rae - pert_rae(6:6:end,:);
            partials_vz = reshape(partials_vz', [], 1);

            % construct the observation matrix: dX(i)/dt
            T = [partials_x/sv_pert(1)  partials_y/sv_pert(2)
partials_z/sv_pert(3) ...
                partials_vx/sv_pert(4) partials_vy/sv_pert(5)
partials_vz/sv_pert(6)];

            % construct the covariance matrix
            Q = diag( repmat(1./sensor{snr}.stddev.^2, 1, numobs) );
            Qi = inv(Q);

            % add to running sums
            TtQiT = TtQiT + T'*Qi*T;
            TtQir = TtQir + T'*Qi*res;
        end % end of loop through sensors

        % compute state vector update
        cov_update = pinv(TtQiT);
        sv_update = cov_update * TtQir;

%         % check for target convergence - covariance method
%         cov_diag = reshape(cov_update,[],1);
%         cov_diag = cov_diag(1:7:end);
%         converged(tgt) = all(sv_update <= sqrt(cov_diag)*1e-7);

        % check for target convergence - cost function method
        if ls_iter > 1
```

```
            converged(tgt) = abs( (stats{ls_iter-1,tgt}.cost_func -
cost_func) / stats{ls_iter-1,tgt}.cost_func ) < koptions.conv_tol;

            % if cost function improved, apply correction
            if cost_func < stats{ls_iter-1,tgt}.cost_func
                target{tgt}.init_sv = target{tgt}.init_sv - [sv_update'
0 0 0];
            end
        else
            % first run, apply correction
            target{tgt}.init_sv = target{tgt}.init_sv - [sv_update' 0 0
0];
        end

        if koptions.echo && converged(tgt)
            disp('   Convergence Criteria Met.')
        end

        % save statistics
        stats{ls_iter,tgt}.cost_func = cost_func;
%#ok<AGROW>
        stats{ls_iter,tgt}.cov = cov_update;
%#ok<AGROW>

        % increment iteration number
        ls_iter = ls_iter + 1;
    end % end of loop through targets

end % end of iteration while loop

% fill state structure with states during free-flight
for tgt = 1:numtgt

    % time series to propagate to, first value is time of SV
    prop_time = [target{tgt}.init_time-time_pad;
target{tgt}.obs_time(target{tgt}.obs_ff)];

    % propagate converged state vector
    [calc_time,calc_sv] = ode45(@kaliper_eom, prop_time,
target{tgt}.init_sv', options);

    % save values
    state{tgt}.time = target{tgt}.obs_time;
%#ok<AGROW>
    state{tgt}.sv   = zeros(length(target{tgt}.obs_time),9);
%#ok<AGROW>
    state{tgt}.sv(target{tgt}.obs_ff,:) = calc_sv(2:end,:);
%#ok<AGROW>
    state{tgt}.cov  = zeros(length(target{tgt}.obs_time),81);
%#ok<AGROW>
    full_cov = zeros(9,9);
    full_cov(1:6,1:6) = stats{end,tgt}.cov;
    acc_cov = 0.001;
    full_cov(7,7) = acc_cov; full_cov(8,8) = acc_cov; full_cov(9,9) =
acc_cov;
    state{tgt}.cov(target{tgt}.obs_ff,:) = repmat( ...
        reshape(full_cov,1,[]), ...
        length(find(target{tgt}.obs_ff)), 1);
%#ok<AGROW>
end

%%%%
%%%% Phase 2 state vector estimation - Reentry - Sensed Accels ~= 0
%%%%

if koptions.echo
    disp('** Entering Phase 2 - Non-Ballistic Estimation **')
end

%%%%
%%%% Phase 2a - least squares sliding window discontinuous estimation
%%%%

% check whether LSSW should be performed
if koptions.lssw

    if koptions.echo
        disp('   * Phase 2a - LSSW Beginning *')
    end

    % loop through targets
    for tgt = 1:numtgt

        % this will only function on 3D observations, others will have
to be interpolated
        all_time = target{tgt}.obs_time(~target{tgt}.obs_ff);
        all_eci  = target{tgt}.obs_eci(~target{tgt}.obs_ff,:);
        all_sv   = zeros(length(all_time), 9);

        % strip out NaNs associated with non-3D observations
        fit_idx = find(~isnan(all_eci(:,1)));

        % data to fit with sliding window
        fit_time = all_time(fit_idx);
        fit_eci  = all_eci(fit_idx,:);
        numobs = length(fit_time);

        % compute lssw window size for each observation
        win_pt  = [1, koptions.window_size/2, numobs-
koptions.window_size/2, numobs];
        win_win = [koptions.min_window_size, koptions.window_size,
koptions.window_size koptions.max_window_size];
        win_size = ceil(interp1(win_pt, win_win, 1:numobs));

        for obs = 1:numobs
```

```
            win_max = min(numobs, max(1+win_size,
obs+ceil(win_size/2)));
            win_min = win_max - win_size;

            win_time = fit_time(win_min:win_max);
            win_eci  = fit_eci(win_min:win_max,:);
            numfit = length(win_time);

            % precompute times
            dt = win_time - win_time(1);
            dt2 = dt.^2;
            dt3 = dt.^3;
            dt4 = dt.^4;

            % construct A matrix
            A = [];
            A(:,1) = ones(numfit,1);
            A(:,2) = dt;
            A(:,3) = dt2/2;
            A(:,4) = dt3/6;
            A(:,5) = dt4/24;

            % separate b vectors for X,Y,Z of observations
            bx = win_eci(:,1);
            by = win_eci(:,2);
            bz = win_eci(:,3);

            % least squares by Singular Value Decomposition for
stability
            pinv_A = pinv(A); % this is much faster than using svd()
and pinv() seperately
            xh_x = pinv_A * bx;
            xh_y = pinv_A * by;
            xh_z = pinv_A * bz;

            all_sv(fit_idx(obs),:) = [xh_x(1) xh_y(1) xh_z(1) xh_x(2)
xh_y(2) xh_z(2) xh_x(3) xh_y(3) xh_z(3)];

%             % subtract gravity acceleration from computed
accelerations
%             grav_acc = calc_grav([xh_x(1) xh_y(1) xh_z(1)]);
%             all_sv(fit_idx(obs),7:9) = all_sv(fit_idx(obs),7:9) -
grav_acc;

        end

        state{tgt}.lssw_std = std(all_sv(fit_idx,1:3)-fit_eci(:,1:3));

        % save SV solutions to use as initial solutions in Phase 2b
        state{tgt}.sv(~target{tgt}.obs_ff,1:3) = all_sv(:,1:3);
    end

    if koptions.echo
```

```
        disp('   * Phase 2a - LSSW Complete  *')
    end
end

%%%%
%%%% Phase 2b - Kalman Filter pass with weighted LSSW results
%%%%

if koptions.echo
    disp('   * Phase 2b - KF Beginning *')
end

% loop through targets
for tgt = 1:numtgt

    % non-free flight indeces
    kf_idx  = find(~target{tgt}.obs_ff);

    for i = 1:length(kf_idx)

        % index of this observation in the overall array
        obsidx = kf_idx(i);
        obs_time = target{tgt}.obs_time(obsidx);
        snridx = target{tgt}.obs_snr(obsidx);

        % pull out previous state vector for easy access
        prev_sv = state{tgt}.sv(obsidx-1,:);

        % propagate previous state to this time for the prediction
        [prop_time,prop_sv] = ode45(@kaliper_eom,
target{tgt}.obs_time(obsidx-1:obsidx), prev_sv', options);

        X_minus = prop_sv(end,:);

        % compute state transition matrix to propagate covariance from
prev state
        delta_t = diff(target{tgt}.obs_time(obsidx-1:obsidx));
        F = calc_state_dist(state{tgt}.sv(obsidx-1,:));
        Phi = eye(9) + F * delta_t;

        % compute noise on covariance propagation
        % KADRE Q Method
        cov_noise = zeros(9);
        q_scale = diag([koptions.qp koptions.qp koptions.qp ...
                        koptions.qv koptions.qv koptions.qv ...
                        koptions.qa koptions.qa koptions.qa]);
        for j = 0:2
            for k = 0:2
                cov_noise = cov_noise + 1/(factorial(j) * factorial(k))
* ...
                    F^j * q_scale * (F')^k * delta_t^(1+j+k)/(1+j+k);
            end
        end
```

```
%          % MSIC Q Method
%          tot_acc = kaliper_calcaccel(prev_sv);
%          acc_mag = sqrt( tot_acc(1)^2 + tot_acc(2)^2 + tot_acc(3)^2 );
%          cov_pos = koptions.qp * tot_acc * delta_t^4 / acc_mag;
%          cov_vel = koptions.qv * tot_acc * delta_t^2 / acc_mag;
%          cov_acc = koptions.qa * tot_acc / acc_mag;
%          cov_noise = diag([cov_pos cov_vel cov_acc]);

        % propagate previous covariance as the prediction
        if i == 1
            init_cov = diag([0.01 0.01 0.01 0.001 0.001 0.001 0.001
0.001 0.001]);
            P_minus = Phi * init_cov * Phi' + cov_noise;
        else
            P_minus = Phi * reshape(state{tgt}.cov(obsidx-1,:),9,9) *
Phi' + cov_noise;
        end



        % compute observation matrix, H
        %
        %      | (X-X0)/R     (Y-Y0)/R   (Z-Z0)/R    0 0 0    0 0 0|
        %  H = |  dAz/dX       dAz/dY     dAz/dZ     0 0 0    0 0 0|
        %      |  dEl/dX       dEl/dY     dEl/dZ     0 0 0    0 0 0|
        %
        %  R = [ (X-X0)^2 + (Y-Y0)^2 + (Z-Z0)^2 ]^(1/2)
        %  (X0,Y0,Z0) - Inertial location of observer @ observation
time
        %

        % determine position of observing sensor
        if sensor{snridx}.snr_type == 1 || sensor{snridx}.snr_type == 2
            % stationary sensor
            sen_eci = tm_ecr2eci(sensor{snridx}.pos_ecf, 0, obs_time,
0);
            sen_ecf = sensor{snridx}.pos_ecf;
            sen_lla = tm_ecr2lla(sensor{snridx}.pos_ecf);
            sen_lla(1) = geoc2geod(sen_lla(1));
            sen_tm  = sensor{snridx}.tm;
        elseif sensor{snridx}.snr_type == 3 || sensor{snridx}.snr_type
== 4
            % moving sensor, interpolate sensor position at obs_time
            interp_lat = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,1), obs_time, 'spline');
            interp_lon = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,2), obs_time, 'spline');
            interp_alt = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,3), obs_time, 'spline');

            sen_lla = [interp_lat, interp_lon, interp_alt];
            [sen_ecf, sen_tm] = calc_sensor(interp_lat, interp_lon,
interp_alt);
            sen_eci = tm_ecr2eci(sen_ecf, 0, obs_time, 0);
        end

        % compute range
        rng = sqrt( (X_minus(1)-sen_eci(1))^2 + (X_minus(2)-
sen_eci(2))^2 + (X_minus(3)-sen_eci(3))^2 );

        % convert X_minus to the observation variables
        sim_ecf = tm_eci2ecr(X_minus(1:3), 0, obs_time, 0);
        sim_sez = tm_ecr2sez(sim_ecf, sen_ecf, sen_tm);
        sim_rae = tm_sez2rae(sim_sez);

        % determine whether we need the range component
        if target{tgt}.obs(obsidx,1) ~= 0
            H = zeros(3,9);

            % fill in range partials
            H(1,1:3) = (X_minus(1:3)-sen_eci) / rng;

            % row indeces to compute Az and El partials in
            azi = 2;
            eli = 3;

            % define simulated & actual observations - full RAE
            sim_z = sim_rae;
            obs_z = target{tgt}.obs(obsidx,:);
        else
            H = zeros(2,9);

            % row indeces to compute Az and El partials in
            azi = 1;
            eli = 2;

            % define simulated & actual observations - only AE
            sim_z = sim_rae(2:3);
            obs_z = target{tgt}.obs(obsidx,2:3);
        end

        % fill in azimuth partials (see appendix)
        % compute common terms contained in partials:
        % sin & cos of sensor position
        slat = sin(sen_lla(1));
        clat = cos(sen_lla(1));
        slon = sin(sen_lla(2));
        clon = cos(sen_lla(2));
        % sin & cos of ECI->ECEF angle
        st = sin(earth.AngVel * obs_time);
        ct = cos(earth.AngVel * obs_time);
        % predicted SV position components
        x = X_minus(1);
        y = X_minus(2);
        z = X_minus(3);

        % azimuth partials
        numer_Az = -slon*(ct*x+st*y-sen_ecf(1))+clon*(-st*x+ct*y-
sen_ecf(2));
        denom_Az = -slat*clon*(ct*x+st*y-sen_ecf(1))-slat*slon*(-
st*x+ct*y-sen_ecf(2))+clat*(z-sen_ecf(3));
        dAz_base = 1/(1 + (numer_Az/denom_Az)^2); % derivative of inv
tan

        % dAz/dX, dY, dZ
        H(azi,1) = dAz_base * ((-slon*ct-clon*st)*denom_Az - (-
slat*clon*ct+slat*slon*st)*numer_Az) / denom_Az^2;
        H(azi,2) = dAz_base * ((-slon*st+clon*ct)*denom_Az - (-
slat*clon*st-slat*slon*ct)*numer_Az) / denom_Az^2;
        H(azi,3) = dAz_base * (-clat*numer_Az) / denom_Az^2;

        % elevation partials
        numer_El = clat*clon*(ct*x+st*y-sen_ecf(1))+clat*slon*(-
st*x+ct*y-sen_ecf(2))+slat*(z-sen_ecf(3));
        denom_El = rng;
        dEl_base = 1/sqrt(1 - (numer_El/denom_El)^2); % derivative of
inv sin

        % dEl/dX, dY, dZ
        H(eli,1) = dEl_base * ((clat*clon*ct - clat*slon*st)*denom_El -
(x-sen_eci(1))/denom_El*numer_El) / denom_El^2;
        H(eli,2) = dEl_base * ((clat*clon*st + clat*slon*ct)*denom_El -
(y-sen_eci(2))/denom_El*numer_El) / denom_El^2;
        H(eli,3) = dEl_base * (slat*denom_El - (z-
sen_eci(3))/denom_El*numer_El) / denom_El^2;

        % if a solution from the LSSW pass was computed, add it to the
calculation
        if koptions.lssw && ~all(state{tgt}.sv(obsidx,1:3) == 0)
            % add SV components to observation & simulation variables
            obs_z = [obs_z state{tgt}.sv(obsidx,1:3)];
            sim_z = [sim_z X_minus(1:3)];

            % add derivatives to H matrix
            % H = [H; zeros(6,3), eye(6,6); zeros(3,9)]; % (complete
SV)
            H = [H; eye(3,3), zeros(3,6)]; % (position SV)

            % construct measurement noise matrix w/ SV components
            R = diag([sensor{snridx}.stddev.^2,
state{tgt}.lssw_std.^2]);
        else
            % construct measurement noise matrix
            R = diag(sensor{snridx}.stddev.^2);
        end

        % with H&R computed, calculate the Kalman Gain
        K = P_minus*H' * pinv(H*P_minus*H' + R);

        % compute SV update
        X_plus = X_minus(:) + K*(obs_z(:) - sim_z(:));
        % compute covariance update
        P_plus = (eye(9) - K*H)*P_minus;

        % save results from this data point
        state{tgt}.sv_minus(obsidx,:)  = X_minus';
        state{tgt}.cov_minus(obsidx,:) = P_minus(:)';
        state{tgt}.sv(obsidx,:)  = X_plus';
        state{tgt}.cov(obsidx,:) = P_plus(:)';

        if koptions.echo
            disp(['Completed KF Pass - Filter Time = '
num2str(obs_time)])
        end
    end
end

if koptions.echo
    disp('   * Phase 2b - KF Complete  *')
end

if koptions.echo
    disp('   * Phase 2c - Smoother Beginning *')
end

%%%%
%%%% Phase 2c - Smoother
%%%%

% loop through targets
for tgt = 1:numtgt

    % non-free flight indeces
    kf_idx  = find(~target{tgt}.obs_ff);
    state{tgt}.cov_smooth = state{tgt}.cov;
    state{tgt}.sv_smooth = state{tgt}.sv;

    % run smoother backwards
    for i = length(kf_idx)-1:-1:1

        % index of this observation in the overall array
        obsidx = kf_idx(i);
        obs_time = target{tgt}.obs_time(obsidx);

        % compute state transition matrix to propagate covariance from
prev state
        delta_t = diff(target{tgt}.obs_time(obsidx:obsidx+1));
        F = calc_state_dist(state{tgt}.sv(obsidx,:));
        Phi = eye(9) + F * delta_t;

        C = reshape(state{tgt}.cov(obsidx,:),9,9) * Phi' * ...
            pinv(reshape(state{tgt}.cov_minus(obsidx+1,:),9,9));
        P_smooth = reshape(state{tgt}.cov(obsidx,:),9,9) + C * ...
```

```
            (reshape(state{tgt}.cov_smooth(obsidx+1,:),9,9) - ...
             reshape(state{tgt}.cov_minus(obsidx+1,:),9,9)) * C';

        state{tgt}.cov_smooth(obsidx,:) = P_smooth(:)';
        state{tgt}.sv_smooth(obsidx,:) = state{tgt}.sv(obsidx,:) + ...
            (C * (state{tgt}.sv_smooth(obsidx+1,:)' -
state{tgt}.sv_minus(obsidx+1,:)'))';

        if koptions.echo
            disp(['Completed KF Pass - Smoother Time = '
num2str(obs_time)])
        end
    end
end

if koptions.echo
    disp('   * Phase 2c - Smoother Complete  *')
end

%%%%
%%%% Phase 3 - Wrap up, Parameter Computation
%%%%

if koptions.echo
    disp('** Phase 3 - Parameter Computation **')
end

% loop through targets
for tgt = 1:numtgt

    %%%% compute residuals
    sv_ecf = tm_eci2ecr(state{tgt}.sv_smooth(:,1:3), 0,
state{tgt}.time, 0);
    sv_sez = zeros(size(sv_ecf));

    % number of sensors for this target
    tgtsnr = unique(target{tgt}.obs_snr);
    numsnr = length(tgtsnr);

    % loop through sensors, computing SEZ information
    for snridx = 1:numsnr
        snr = tgtsnr(snridx);

        % observations from this sensor, and those observation times
        obsidx = target{tgt}.obs_snr == snr;
        obs_time = target{tgt}.obs_time(obsidx);

        % stationary range, az, el OR az, el observations
        if sensor{snr}.snr_type == 1 || sensor{snr}.snr_type == 2

            % convert ECEF SV positions to stationary SEZ
            sv_sez(obsidx,:) = tm_ecr2sez(sv_ecf(obsidx,:),
sensor{snr}.pos_ecf, sensor{snr}.tm);
```

```
            % moving range, az, el OR az, el observations
        elseif sensor{snr}.snr_type == 3 || sensor{snr}.snr_type == 4

            % interpolate sensor positions at observation times
            interp_lat = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,1), obs_time, 'spline');
            interp_lon = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,2), obs_time, 'spline');
            interp_alt = interp1(sensor{snr}.pos_time,
sensor{snr}.pos_lla(:,3), obs_time, 'spline');
            [sensor_pos, sensor_tm] = calc_sensor_move(interp_lat,
interp_lon, interp_alt);

            % convert ECEF SV positions to moving SEZ
            sv_sez(obsidx,:) = tm_ecr2sez_move(sv_ecf(obsidx,:),
sensor_pos, sensor_tm);
        else
            error(['Unrecognized sensor type ID: '
num2str(sensor{snr}.snr_type)])
        end
    end

    % compute RAE state
    sv_rae = tm_sez2rae(sv_sez);
    target{tgt}.res = repmat(NaN, size(sv_rae));

    % loop through sensors, computing residuals
    for snridx = 1:numsnr
        snr = tgtsnr(snridx);

        % observations from this sensor
        obsidx = target{tgt}.obs_snr == snr;

        if sensor{snr}.snr_type == 1 || sensor{snr}.snr_type == 3
            % range, azimuth, and elevation observations
            target{tgt}.res(obsidx,:) = target{tgt}.obs(obsidx,:) -
sv_rae(obsidx,:);

        elseif sensor{snr}.snr_type == 2 || sensor{snr}.snr_type == 4
            % azimuth and elevation observations
            target{tgt}.res(obsidx,2:3) = target{tgt}.obs(obsidx,2:3) -
sv_rae(obsidx,2:3);
        end
    end

    % plot residuals, if requested
    if koptions.echo
        figure, hold on

        % loop through sensors, computing residuals
        for snridx = 1:numsnr
            snr = tgtsnr(snridx);
            % observations from this sensor
            obsidx = target{tgt}.obs_snr == snr;
            if snr == 1
                subplot(3,1,1), hold on
                plot(target{tgt}.obs_time(obsidx),
target{tgt}.res(obsidx,1), 'r.')
                subplot(3,1,2), hold on
                plot(target{tgt}.obs_time(obsidx),
target{tgt}.res(obsidx,2), 'r.')
                subplot(3,1,3), hold on
                plot(target{tgt}.obs_time(obsidx),
target{tgt}.res(obsidx,3), 'r.')
            else
                cols = 'rbmk'; nc = length(cols);
                subplot(3,1,1)
                plot(target{tgt}.obs_time(obsidx),
target{tgt}.res(obsidx,1), [cols(mod(snr-1,nc)+1) '.'])
                subplot(3,1,2)
                plot(target{tgt}.obs_time(obsidx),
target{tgt}.res(obsidx,2), [cols(mod(snr-1,nc)+1) '.'])
                subplot(3,1,3)
                plot(target{tgt}.obs_time(obsidx),
target{tgt}.res(obsidx,3), [cols(mod(snr-1,nc)+1) '.'])
            end
        end
    end

    %%%% compute lat/lon/alt positions
    state{tgt}.lla = tm_eci2vel(state{tgt}.sv_smooth(:,1:3),
state{tgt}.time);
    state{tgt}.lla(:,3) = state{tgt}.lla(:,3)/1000;

    %%%% compute altitude
    state{tgt}.alt = state{tgt}.lla(:,3);

    %%%% compute ballistic coefficient
    % atmospheric values
    [temp, pres, dens] = calc_atmos(state{tgt}.alt);
%#ok<NASGU>

    % air relative velocity for dynamic pressure
    vxa = state{tgt}.sv_smooth(:,4) +
earth.AngVel.*state{tgt}.sv_smooth(:,2);
    vya = state{tgt}.sv_smooth(:,5) -
earth.AngVel.*state{tgt}.sv_smooth(:,1);
    vz  = state{tgt}.sv_smooth(:,6);
    va  = sqrt( vxa.^2 + vya.^2 + vz.^2 );

    % dynamic pressure (convert dens from kg/m3 to kg/km3)
    dynpres = .5 * 1000^3 * dens .* va.^2;

    % magnitude of drag acceleration
    drag_mag = sqrt( state{tgt}.sv_smooth(:,7).^2 +
state{tgt}.sv_smooth(:,8).^2 + state{tgt}.sv_smooth(:,9).^2 );

    % ballistic coefficient in kg/m2
    state{tgt}.beta = dynpres ./ drag_mag / 1000^2;

    %%%% Mach number (non-dim)
    gamma = 1.4;
    gas_const = 284;
    state{tgt}.mach = va ./ sqrt( gamma * gas_const * temp ) * 1000;
end

if koptions.echo
    disp('** KALIPER Run Complete **')
end

% set output variables
if nargout == 1
    % one output requested
    varargout{1} = state;
elseif nargout == 2
    % two outputs requested
    varargout{1} = state;
    varargout{2} = target;
elseif nargout == 3
    % three outputs requested
    varargout{1} = state;
    varargout{2} = target;
    varargout{3} = stats;
end

    %KALIPER_CALCINIT - compute initial guess for state
    %
    % Initial guess based on least squares solution to equations of
motion
    % assuming constant 4th derivative of position:
    % x_measured =
    %               x_true +
    %               x_true(1) * T +
    %               x_true(2) * T^2 / 2 +
    %               x_true(3) * T^3 / 6 +
    %               x_true(4) * T^4 / 24 + noise
    %
    % Note: number in () is the nth derivative
    %
    function [init_time, init_sv] = kaliper_calcinit(tgt)

        numobs = length(target{tgt}.obs_time);

        % check if free flight observations have been predetermined
        if isfield(target{tgt}, 'obs_ff')
            target{tgt}.obs_ff = false(numobs,1);
        end
        if ~isfield(target{tgt}, 'obs_ff') || ~any(target{tgt}.obs_ff)
            % initial attempt will be to select observations outside of
the
            % Eath's atmosphere
            obs_alt = calc_alt(target{tgt}.obs_eci);
            num_3d = length(find(~isnan(obs_alt)));

            target{tgt}.obs_ff = obs_alt >= earth.AtmAlt;

            % check to see if there is still nothing defined
            if ~any(target{tgt}.obs_ff)
                % select highest altitude points until 10% of available
are
                % selected (arbitrary)
                alt_step = -1;
                cutoff_alt = max(obs_alt) + alt_step;
                while length(find(target{tgt}.obs_ff))/num_3d < .1
                    target{tgt}.obs_ff = obs_alt >= cutoff_alt;
                    cutoff_alt = cutoff_alt + alt_step;
                end
            end
        end

        % compute least squares solution
        fit_time = target{tgt}.obs_time(target{tgt}.obs_ff);
        fit_eci = target{tgt}.obs_eci(target{tgt}.obs_ff,:);
        numfit = length(fit_time);

        % precompute times
        dt = fit_time - fit_time(1);
        dt2 = dt.^2;
        dt3 = dt.^3;
        dt4 = dt.^4;

        % construct A matrix
        A = [];
        A(:,1) = ones(numfit,1);
        A(:,2) = dt;
        A(:,3) = dt2/2;
        A(:,4) = dt3/6;
        A(:,5) = dt4/24;

        % separate b vectors for X,Y,Z of observations
        bx = fit_eci(:,1);
        by = fit_eci(:,2);
        bz = fit_eci(:,3);

        % least squares by Singular Value Decomposition for stability
        pinv_A = pinv(A); % this is much faster than using svd() and
pinv() seperately
        xh_x = pinv_A * bx;
        xh_y = pinv_A * by;
        xh_z = pinv_A * bz;
        % compile initial guess
        init_time = fit_time(1);
        init_sv = [xh_x(1) xh_y(1) xh_z(1) xh_x(2) xh_y(2) xh_z(2)
xh_x(3) xh_y(3) xh_z(3)];

        % for initialization purposes, we're assuming free-flight
        init_sv(7:9) = 0;

        % use the time determined by the 3-D data to include others
        ff_time = target{tgt}.obs_time(target{tgt}.obs_ff);
        target{tgt}.obs_ff = target{tgt}.obs_time >= ff_time(1) & ...
                             target{tgt}.obs_time <= ff_time(end);

    end % end of kaliper_calcinit function

    %KALIPER_EOM - Generatlized equations of motion
    %
    %  These equations of motion can be used with ode45 and the state
to
    %  propagate the state forward in time.
    %
    %  Note: This is a modification to METAL's eom_re and eom_ff.
Since
    %  the sensed acceleration is an element in the state vector, these
    %  EOMs satisfy both free flight and reentry conditions
    %
    %
    function dy = kaliper_eom(t,y)
%#ok<INUSL>

        % change y from single vector to columns of vectors
        nval = size(y,1);
        nvec = nval/9;
        y = reshape(y, 9, nvec);
        dy = zeros(9, nvec);

        % compute gravitational acceleration
        grav_acc = calc_grav(y(1:3,:)')';

        % compute total acceleration
        tot_acc = grav_acc + y(7:9,:);

        % derivatives
        dy(1,:) = y(4,:);       % dx/dt   = vx
        dy(2,:) = y(5,:);       % dy/dt   = vy
        dy(3,:) = y(6,:);       % dz/dt   = vz
        dy(4,:) = tot_acc(1,:); % d2x/dt2 = tax
        dy(5,:) = tot_acc(2,:); % d2y/dt2 = tay
        dy(6,:) = tot_acc(3,:); % d2z/dt2 = taz
        % acceleration derivatives = 0

        % change dy back to single column to satisfy ode45
        dy = reshape(dy, nval, 1);
    end % end of kaliper_eom function

    %CALC_STATE_DIST - calculate the state distribution matrix (F) for
a given SV.
    %                 used to compute state transition matrix by: Phi
= I + F*dt
    function f = calc_state_dist(sv)

        % compute partials of the state, initialize with all zeros
        f = zeros(9,9);

        % position derivates
        f(1,4) = 1; f(2,5) = 1; f(3,6) = 1;

        % terms for velocity partials
        r = sqrt(sv(1)^2 + sv(2)^2 + sv(3)^2);
        mu = earth.GravConst;

        % x velocity gravity partials (simple gravity)
        f(4,1) = -mu/r^3 * (1 - 3*sv(1)^2/r^2);
        f(4,2) = 3*mu*sv(1)*sv(2)/r^5;
        f(4,3) = f(4,2)/sv(2)*sv(3);

        % y velocity gravity partials
        f(5,1) = f(4,2);
        f(5,2) = -mu/r^3 * (1 - 3*sv(2)^2/r^2);
        f(5,3) = f(5,1)/sv(1)*sv(3);

        % z velocity gravity partials
        f(6,1) = f(4,3);
        f(6,2) = f(5,3);
        f(6,3) = -mu/r^3 * (1 - 3*sv(3)^2/r^2);

        % velocity sensed accel partials
        f(4,7) = 1; f(5,8) = 1; f(6,9) = 1;

    end % end of calc_state_dist function

end % end of kaliper main function
```

read_data.m

`read_data.m` is a MATLAB script written to read in data from truth cases. Original formatting is preserved to maintain functionality when pasted into MATLAB.

```
function [target,sensor] = read_data(datafile, varargin)

snr = 1;
tgt = 1;

if nargin == 1
    target{tgt}.obs = [];
    target{tgt}.obs_time = [];
    target{tgt}.obs_snr = [];
elseif nargin == 3
    target = varargin{1};
    sensor = varargin{2};

    tgt = length(target) + 1;
    target{tgt}.obs = [];
    target{tgt}.obs_time = [];
    target{tgt}.obs_snr = [];

    for i = 1:length(target)
        snr = max(max(target{i}.obs_snr), snr);
    end
    snr = snr + 1;
elseif nargin == 4
    target = varargin{1};
    sensor = varargin{2};
    snr = varargin{3};

    tgt = length(target) + 1;
    target{tgt}.obs = [];
    target{tgt}.obs_time = [];
    target{tgt}.obs_snr = [];
elseif nargin == 5
    target = varargin{1};
    sensor = varargin{2};
    snr = varargin{3};
    tgt = varargin{4};

    if ~isfield(target{tgt}, 'obs'), target{tgt}.obs = []; end
    if ~isfield(target{tgt}, 'obs_time'), target{tgt}.obs_time = [];
end
    if ~isfield(target{tgt}, 'obs_snr'), target{tgt}.obs_snr = []; end
end

fid = fopen(datafile);
```

```
sen_pos = fgetl(fid);
dataline = fgetl(fid);
fclose(fid);
[sen_lat, sen_lon, sen_alt] = strread(sen_pos, '%n %n %n');

numdata = length(strread(dataline, '%n')) - 1;

if numdata == 3
    % RAE
    [obs_time, obs_r, obs_a, obs_e] = textread(datafile, '%n %n %n %n',
'headerlines', 1);
    target{tgt}.obs = [target{tgt}.obs; [obs_r(:), obs_a(:)*pi/180,
obs_e(:)*pi/180]];

    sensor{snr}.snr_type = 1;
    sensor{snr}.stddev = [.002 .03 * pi/180, .03 * pi/180];
    sensor{snr}.obs_bias = [0 0 0];
elseif numdata == 2
    % AE
    [obs_time, obs_a, obs_e] = textread(datafile, '%n %n %n',
'headerlines', 1);
    target{tgt}.obs = [target{tgt}.obs; [zeros(size(obs_a(:)))
obs_a(:)*pi/180, obs_e(:)*pi/180]];

    sensor{snr}.snr_type = 2;
    sensor{snr}.stddev = [.03 * pi/180, .03 * pi/180];
    sensor{snr}.obs_bias = [0 0];
else
    error(['Unable to identify data file with ' num2str(numdata) '
entries per line.'])
end

target{tgt}.obs_time = [target{tgt}.obs_time; obs_time];
target{tgt}.obs_snr = [target{tgt}.obs_snr; repmat(snr,
size(obs_time))];

[target{tgt}.obs_time, idx] = sort(target{tgt}.obs_time);
target{tgt}.obs = target{tgt}.obs(idx,:);
target{tgt}.obs_snr = target{tgt}.obs_snr(idx);

target{tgt}.numobs = length(target{tgt}.obs_time);

sensor{snr}.pos_lla = [sen_lat*pi/180, sen_lon*pi/180, sen_alt/1000];
[sensor{snr}.pos_ecf, sensor{snr}.tm] = calc_sensor(sen_lat*pi/180,
sen_lon*pi/180, sen_alt/1000);
```

run_kaliper_cases.m

`run_kaliper_cases.m` is a MATLAB script written to analyze truth cases and demonstrate input syntax. Original formatting is preserved to maintain functionality when pasted into MATLAB.

```
close all
clear

%%% Items for computing truth accelerations
define_air
read_stdatmos('stdatmos76.dat')
% RV definition
rv_beta = 5600;
rv_hca = 9.5;
rv_br = 0.24;
define_berman(rv_hca, rv_br)
%%%

%%%%%% Scenario 1 - Pure Ballistic
%%% Case 1 - Single Sensor - 1x Data Rate, along heading
% files = {'scen1_sen1_pos1_rate1.rae'};
%%% Case 2 - Single Sensor - 1x Data Rate, perp to heading
% files = {'scen1_sen1_pos2_rate1.rae'};
%%% Case 3 - Single Sensor - 1x Data Rate, 45 deg to heading
files = {'scen1_sen2_pos3_rate1.rae'};
%%% Case 4 - Single Sensor - 2x Data Rate, best position
% files = {'scen1_sen1_pos1_rate2.rae'};
%%% Case 5 - 2 Sensors - 1x Data Rate, along & along positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos1_rate1.rae'};
%%% Case 6 - 2 Sensors - 1x Data Rate, along & 45 deg to positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos2_rate1.rae'};
%%% Case 7 - 2 Sensors - 1x Data Rate, along & perp to positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos3_rate1.rae'};
%%% Case 8 - 2 Sensors - 1x&2x Data Rate, along & perp to positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos1_rate2.rae'};
%%% Case 9 - 2 Sensors - 2x&1x Data Rate, along & perp to positions
% files = {'scen1_sen1_pos1_rate2.rae', 'scen1_sen2_pos3_rate1.rae'};
%%% Case 10 - 2 Sensors - 2x Data Rate, along & perp to positions
% files = {'scen1_sen1_pos1_rate2.rae', 'scen1_sen2_pos3_rate2.rae'};
%%% Case 11 - 2 Sensors - 2x Data Rate, perp to & perp to positions
% files = {'scen1_sen1_pos3_rate2.rae', 'scen1_sen2_pos3_rate2.rae'};
%%% Case 12 - 3 Sensors - 1x Data Rate, along, along, & along positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos1_rate1.rae',
'scen1_sen3_pos1_rate1.rae'};
%%% Case 13 - 3 Sensors - 1x Data Rate, along, along, & 45 deg to
positions
```

```
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos1_rate1.rae',
'scen1_sen3_pos2_rate1.rae'};
%%% Case 14 - 3 Sensors - 1x Data Rate, along, along, & perp to
positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos1_rate1.rae',
'scen1_sen3_pos3_rate1.rae'};
%%% Case 15 - 3 Sensors - 1x Data Rate, along, along, & perp to
positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen2_pos1_rate1.rae',
'scen1_sen3_pos2_rate2.rae'};
%%% Case 16 - 2 Sensors (RAE & AE) - 1x Data Rate, along & along
positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen3_pos1_rate1.rae'};
%%% Case 17 - 2 Sensors (RAE & AE) - 1x Data Rate, along & 45 deg to
positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen3_pos2_rate1.rae'};
%%% Case 18 - 2 Sensors (RAE & AE) - 1x Data Rate, along & perp to
positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen3_pos3_rate1.rae'};
%%% Case 19 - 2 Sensors (RAE & AE) - 1x&2x Data Rate, along & 45 deg to
positions
% files = {'scen1_sen1_pos1_rate1.rae', 'scen1_sen3_pos2_rate2.rae'};

%%%%%% Scenario 2 - Single Maneuver
%%% Case 20 - ` Sensors (RAE) - 1x Data Rate, along position
% files = {'scen2_sen1_pos1_rate1.rae'};
%%% Case 21 - ` Sensors (RAE) - 1x Data Rate, 45 deg to position
% files = {'scen2_sen1_pos2_rate1.rae'};
%%% Case 22 - ` Sensors (RAE) - 1x Data Rate, perp to position
% files = {'scen2_sen1_pos3_rate1.rae'};
%%% Case 23 - Single Sensor - 2x Data Rate, best position
% files = {'scen2_sen1_pos3_rate2.rae'};
%%% Case 24 - 2 Sensors - 1x Data Rate, along & along positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen2_pos1_rate1.rae'};
%%% Case 25 - 2 Sensors - 1x Data Rate, along & 45 deg to positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen2_pos2_rate1.rae'};
%%% Case 26 - 2 Sensors - 1x Data Rate, along & perp to positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen2_pos3_rate1.rae'};
%%% Case 27 - 2 Sensors - 1x Data Rate, 45 deg to & along positions
% files = {'scen2_sen1_pos2_rate1.rae', 'scen2_sen2_pos1_rate1.rae'};
%%% Case 28 - 2 Sensors - 1x Data Rate, 45 deg to & 45 deg to positions
% files = {'scen2_sen1_pos2_rate1.rae', 'scen2_sen2_pos2_rate1.rae'};
%%% Case 29 - 2 Sensors - 1x Data Rate, 45 deg to & perp to positions
% files = {'scen2_sen1_pos2_rate1.rae', 'scen2_sen2_pos3_rate1.rae'};
%%% Case 30 - 2 Sensors - 1x&2x Data Rate, 45 deg to & along positions
% files = {'scen2_sen1_pos2_rate1.rae', 'scen2_sen2_pos1_rate2.rae'};
%%% Case 31 - 3 Sensors - 1x Data Rate, along, 45 deg to, & along
positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen2_pos2_rate1.rae',
'scen2_sen3_pos1_rate1.ae'};
%%% Case 32 - 3 Sensors - 1x Data Rate, along, 45 deg to, & 45 deg to
positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen2_pos2_rate1.rae',
'scen2_sen3_pos2_rate1.ae'};
```

```
%%% Case 33 - 3 Sensors - 1x Data Rate, along, 45 deg to, & perp to
positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen2_pos2_rate1.rae',
'scen2_sen3_pos3_rate1.ae'};
%%% Case 34 - 3 Sensors - 1x,2x,2x Data Rate, along, 45 deg to, & perp
to positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen2_pos2_rate2.rae',
'scen2_sen3_pos3_rate2.ae'};
%%% Case 35 - 2 Sensors (RAE & AE) - 1x Data Rate, along & along
positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen3_pos1_rate1.ae'};
%%% Case 36 - 2 Sensors (RAE & AE) - 1x Data Rate, along & 45 deg to
positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen3_pos2_rate1.ae'};
%%% Case 37 - 2 Sensors (RAE & AE) - 1x Data Rate, along & perp to
positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen3_pos3_rate1.ae'};
%%% Case 38 - 2 Sensors (RAE & AE) - 1x&2x Data Rate, along & 45 deg to
positions
% files = {'scen2_sen1_pos1_rate1.rae', 'scen2_sen3_pos2_rate2.ae'};

%%%%%% Scenario 3 - Double Maneuver
%%% Case 39 - ` Sensors (RAE) - 1x Data Rate, along position
% files = {'scen3_sen1_pos1_rate1.rae'};
%%% Case 40 - ` Sensors (RAE) - 1x Data Rate, 45 deg to position
% files = {'scen3_sen1_pos2_rate1.rae'};
%%% Case 41 - ` Sensors (RAE) - 1x Data Rate, perp to position
% files = {'scen3_sen1_pos3_rate1.rae'};
%%% Case 42 - Single Sensor - 2x Data Rate, best position
% files = {'scen3_sen1_pos2_rate2.rae'};
%%% Case 43 - 2 Sensors - 1x Data Rate, along & along positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen2_pos1_rate1.rae'};
%%% Case 44 - 2 Sensors - 1x Data Rate, along & 45 deg to positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen2_pos2_rate1.rae'};
%%% Case 45 - 2 Sensors - 1x Data Rate, along & perp to positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen2_pos3_rate1.rae'};
%%% Case 46 - 2 Sensors - 1x Data Rate, 45 deg to & along positions
% files = {'scen3_sen1_pos2_rate1.rae', 'scen3_sen2_pos1_rate1.rae'};
%%% Case 47 - 2 Sensors - 1x Data Rate, 45 deg to & 45 deg to positions
% files = {'scen3_sen1_pos2_rate1.rae', 'scen3_sen2_pos2_rate1.rae'};
%%% Case 48 - 2 Sensors - 1x Data Rate, 45 deg to & perp to positions
% files = {'scen3_sen1_pos2_rate1.rae', 'scen3_sen2_pos3_rate1.rae'};
%%% Case 49 - 2 Sensors - 1x&2x Data Rate, 45 deg to & along positions
% files = {'scen3_sen1_pos2_rate1.rae', 'scen3_sen2_pos1_rate2.rae'};
%%% Case 50 - 3 Sensors - 1x Data Rate, along, 45 deg to, & along
positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen2_pos2_rate1.rae',
'scen3_sen3_pos1_rate1.ae'};
%%% Case 51 - 3 Sensors - 1x Data Rate, along, 45 deg to, & 45 deg to
positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen2_pos2_rate1.rae',
'scen3_sen3_pos2_rate1.ae'};
%%% Case 52 - 3 Sensors - 1x Data Rate, along, 45 deg to, & perp to
positions
```

```
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen2_pos2_rate1.rae',
'scen3_sen3_pos3_rate1.ae'};
%%% Case 53 - 3 Sensors - 1x,2x,2x Data Rate, along, 45 deg to, & 45
deg to positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen2_pos2_rate2.rae',
'scen3_sen3_pos2_rate2.ae'};
%%% Case 54 - 2 Sensors (RAE & AE) - 1x Data Rate, along & along
positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen3_pos1_rate1.ae'};
%%% Case 55 - 2 Sensors (RAE & AE) - 1x Data Rate, along & 45 deg to
positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen3_pos2_rate1.ae'};
%%% Case 56 - 2 Sensors (RAE & AE) - 1x Data Rate, along & perp to
positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen3_pos3_rate1.ae'};
%%% Case 57 - 2 Sensors (RAE & AE) - 1x&2x Data Rate, along & 45 deg to
positions
% files = {'scen3_sen1_pos1_rate1.rae', 'scen3_sen3_pos2_rate2.ae'};

% indeces for truth comparisons
scen = 1;
rate = [1,1,1];

% read in data files
[target,sensor] = read_data(files{1});
for i = 2:length(files)
    [target,sensor] = read_data(files{i},target,sensor,i,1);
end

% run kaliper
% koptions.echo = true(1);
koptions.max_iter = 15;
% koptions.conv_tol = .00001;
koptions.lssw = false(1);
koptions.sig_edit = 20;
koptions.qp = 0.0;
koptions.qv = 0.0;
koptions.qa = 0.001;

tic, [state, target] = kaliper(target, sensor, koptions); toc

load all_case_truth

eci = [];
acc = [];
comp = [];
ts = [];

for i = 1:length(files)

    traj_acc = calc_accel(traj_eci{scen,rate(i)}, rv_beta*10^6, 0, 0);
    traj_acc = traj_acc - calc_grav(traj_eci{scen,rate(i)});
    if scen == 2
        alt = calc_alt(traj_eci{scen,rate(i)});
        traj_acc(alt <= 40,1) = traj_acc(alt <= 40,1) - (40 - alt(alt
<= 40))/20*0.0098;

    elseif scen == 3
        alt = calc_alt(traj_eci{scen,rate(i)});
        traj_acc(alt <= 40,1) = traj_acc(alt <= 40,1) - (40 - alt(alt
<= 40))/20*0.0098;
        traj_acc(alt <= 20,2) = traj_acc(alt <= 20,2) - (20 - alt(alt
<= 20))/5*0.0098;
    end

    eci = [eci; traj_eci{scen,rate(i)}];
    acc = [acc; traj_acc];
    ts = [ts; traj_time{scen,rate(i)}];

    snridx = target{1}.obs_snr == i;
    comp = [comp; state{1}.sv_smooth(snridx,:)];
end

[ts,id] = sort(ts);
eci = eci(id,:);
acc = acc(id,:);
comp = comp(id,:);

stateerr = comp - [eci, acc];
staterms = sqrt( mean( (stateerr).^2 ) ) * 1000;

% easy to copy from:
fprintf('%.1f\n', staterms)
```

Bibliography

1. Bittle, N. M. (2009, March). Estimating the Aerodynamic and Heating Properties of an Unknown Reentry Vehicle Using Least Squares Filtering. *AFIT/GSS/ENY/07-M01* . Wright-Patterson AFB, OH: School of Engineering, Air Force Institute of Technology.

2. Brooks, E., Hill, J., Meiss, A., Merchant, J., Precoda, P., & Valentine, C. (n.d.). Mathematical and Engineering Trajectory Analysis Library. Wright-Patterson AFB, OH: NASIC.

3. Daum, F. (2005, August). Nonlinear Filters: Beyond the Kalman Filter. *20 (8)* , 57-69. Aerospace and Electronic Systems Magazine, IEEE.

4. Donald, R. G., Stirling, W. C., & & Westmiller, J. C. (1983, Dec 23). KADRE: Mathematical Theory and Engineering Description, Ver B.1. Sunnyvale, California: ESL Inc, Subsidary of TRW.

5. Hicks, K. D. (2008, Summer Quarter). Astrodynamic Reentry. Wright-Patterson AFB, OH: School of Engineering and Management, Air Force Institute of Technology.

6. Holmes, L. M. (2006, March). Estimating the Aerodynamic Properties of an Unknown Reentry Vehicle Using Least Squares Filtering. *AFIT/GA/ENY/06-M04* . Wright-Patterson AFB, OH: Air Force Institute of Technology.

7. Jackson, K., & Farbman, M. (2007, August 20-23). Trajectory Reconstruction with a Least Squares Sliding Window (LSSW) Filter. *AIAA Guidance, Navigation and Control Conference and Exhibit* . Hilton Head, South Carolina.

8. Kawase, T. T. (2001). A Kalman Tracker with a Turning Acceleration Estimator. *84 (1)* , 1-11.

9. Lee, S., & Liu, C. (1999, November-December). Trajectory Estimation of Reentry Vehicles by Use of On-Line Input Estimator. *Journal of Guidance, Control, and Dynamics , 22 (6)* , 808-815.

10. Minvielle, P. (2005). Decades of Improvement in Re-entry Ballistic Vehicle Tracking. *20 (8)* , 1-14.

11. Strang, G. (2006). *Linear Algebra and Its Applications.* Brooks/Cole.

12. U.S. Standard Atmosphere. (1976). 53-59. National Aeronautics and Space Administration.

13. Wiesel, W. E. (2003). Modern Orbit Determination. Beavercreek, OH: Aphelion Press.

14. Wynne, M. &. (2008). *Fiscal Year 2009 Air Force Posture Statement*. Retrieved December 20, 2009, from USAF Website: http://www.posturestatement.af.mil/shared/media/document/AFD-080310-037.pdf

15. Yu, Y. a. (2006). Particle Filters for Maneuvering Target Tracking Problem. *Signal Processing , 86* , 195-203.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 25-03-2010	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* June 2008 – March 2010

TITLE AND SUBTITLE Estimating Characteristics of a Maneuvering Reentry Vehicle Observed by Multiple Sensors	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Evan M. Brooks	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/ENY), 2950 Hobson Way, Building 640, WPAFB OH 45433-8865	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GA/ENY/10-M02
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) This field intentionally left blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Post flight analysis of ballistic missile reentry vehicles is an area of focus for the U.S. Government, especially for those involved in ballistic missile defense. Typically, this analysis incorporates either a model-driven least squares filter or a data-following Kalman filter. The research performed here developed a filter that attempts to integrate the strengths of both filters. A least squares filter operates on observation data collected during exoatmospheric free flight and a Kalman filter is used to analyze data collected lower in the atmosphere, where potential maneuvers could be performed. Additionally, the filter was written to incorporate data from multiple sensors.

Using this hybrid filter, different scenarios are investigated to determine the potential benefits of adding additional collectors, increasing the data rate of collecting sensors, and investigating the effects of different collector geometry on the accuracy of results.

Results show that the filter successfully transitions from the least squares to Kalman filter, using the final values of the free flight propagation for the Kalman filter's initial state. Using this filter to investigate different collection scenarios, it was determined that the best results are achieved when multiple collectors are used, the data collection rate of the collectors is increased, and collectors are positioned perpendicular to the reentry vehicle heading.

15. SUBJECT TERMS
maneuvering, reentry, Kalman, least squares, hybrid filter, collection geometry, radar

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Richard Cobb
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	123	(937) 255-6565, ext 4559 (Richard.Cobb@afit.edu)