Tracking System for Photon-Counting Laser Radar

by

Joshua TsuKang Chang

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May, 2007

[June 2007]

Author [illegible]
Department of Electrical Engineering and Computer Science
May 11, 2007

Certified by [illegible]
Dr. Leaf Jiang
VI-A Company Thesis Supervisor

Certified by [illegible]
Professor James K. Roberge
M.I.T. Thesis Supervisor

Accepted by [illegible]
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Tracking System for Photon-Counting Laser Radar

by

Joshua TsuKang Chang

Submitted to the Department of Electrical Engineering and Computer Science

May 11, 2007

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

The purpose of this thesis is to build the tracking system for a photon-counting laser radar – specifically a laser radar that has the ability to perform direct and coherent detection measurement at low signal levels with common laser, optics and detector hardware. The heart of the tracking algorithm is a Kalman filter, and optimal Kalman filter parameters are determined using software simulations. The tracking algorithm was tested against various simulated (software only) and emulated (with actual hardware) trajectories. We also built and tested the real-time tracking system hardware. The algorithms and methods proposed in this thesis achieve the objective of tracking a target at 1,500 km range to within 1-cm accuracy.

VI-A Company Thesis Supervisor: Leaf A. Jiang, Ph.D.

Title: LL-Technical Staff

M.I.T. Thesis Supervisor: James K. Roberge, Sc.D.

Title: Professor of Electrical Engineering

This work is sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Acknowledgements

First, I would like to thank my mentor at MIT Lincoln Laboratory, Dr. Leaf Jiang, for teaching me what research really means, answering all of my questions, nurturing me to understand each and every facet of my thesis , encouraging me when portions of the software or hardware did not work, and overall, making this thesis move from concept to reality. I would also like to thank Dr. Jane Luu for her questions, which helped me realize what I did and did not really understand. Ted Square is yet another member of the staff at MIT Lincoln Laboratory that helped make this thesis a reality. His willingness to stay extra hours to help debug portions of my code, as well as his willingness to develop other programs for my benefit has definitely helped me finish my thesis on time. I would also like to thank Dr. Adam Milstein for his immeasurable help in understanding Kalman filters and their implementations.

I would also like to thank family and friends who have been encouraging me through this process, whether it is by verbal or written means of communication, or by providing food, or by helping me get to and from work at odd hours.

Above all, I would like to thank God for the strength and energy He has given to me throughout this entire experience, for providing me with the best mentors and the best staff to assist me, and for providing friends and family who have been a constant encouragement. In the words of Johann Sebastian Bach – *Soli Deo Gloria – To God Alone be the Glory.*

Table of Contents

Table of Figures

Table of Tables

1. Introduction

From World War II to the present, radar has been the principal tool for tracking objects in space, and with the advent of the laser, the same tracking techniques have been extended to laser radar (ladar) systems. The shorter wavelengths offered by lasers allow for better angular resolution, albeit at the expense being more susceptible to adverse weather conditions (e.g. rain, fog). MIT Lincoln Laboratory has been building a ladar system capable of both direct and coherent detection, using common transmit and receive hardware. Two innovations were required for achieving this dual-mode ladar: (1) a photon-counting receiver [1] and (2) a 100-W flexible waveform generator laser that could generate both direct waveforms (e.g., short pulses) and coherent waveforms (e.g., long pulses). The result is a versatile ladar that is capable of handling the large assortment of measurements.

The receiver for this dual-mode ladar is an InGaAs Geiger-mode avalanche photodiode (APD) array and the transmitter is a 1064-nm flexible waveform generator. The transmitter actually consists of a Lincoln Laboratory-developed microchip laser and a narrow-linewidth fiber laser; both operating at 1 kHz. By making use of both these laser sources, the transmitter can switch back and forth between various waveforms, depending on the desired measurement. Similarly, the receiver also switches between direct and coherent detection simply by adding or blocking out the local oscillator. Thanks to the use of a photon-counting detector as a receiver, the system can operate at very low flux – even sub-photon – levels, and thus push the limits of ladar application to large distances (e.g., 1000 km).

A brassboard system for this dual-mode ladar has been built and tested in MIT Lincoln Laboratory's Optical Systems Test Facility (OSTF), a facility that is capable of emulating signal returns from meter-class targets at large ranges (tens of kilometers to several thousands of kilometers). Several laser modalities, such as range profile and angle-angle-Doppler, have been successfully demonstrated on meter-scale sized targets at hundreds to a thousand kilometers [2].

This thesis focuses on the acquisition and tracking of the velocity and range of a target and is not concerned with angular tracking. The angular tracking is typically controlled by feedback from a

CCD camera to the telescope mount and its hardware and software are typically independent and separate from the ladar.

1.1 Organization of Thesis

This thesis discusses the design and implementation of a functional real-time tracking system, which makes use of Kalman filters, to enable the dual-mode ladar system to track targets in space. Chapter 2 introduces the extended Kalman filter, explores the rationale behind why it was chosen for the tracking system, and describes the characteristics of the filter. Before implementing the tracking algorithm in the actual hardware, the algorithm was simulated (in MATLAB) to test its robustness and accuracy (Chapter 3). Once the design of the tracking algorithm was complete, it was implemented into hardware (Chapter 4). Chapter 5 describes the performance of the tracking algorithm in experiments carried out with the hardware. Finally, future research and work is suggested and discussed.

2. The Kalman Filter and its Variations

One of the most well-known and often-used tracking algorithms is the Kalman filter, named after Rudolph E. Kalman, who published his seminal paper in 1960 [3], describing a recursive algorithm with estimates of the hidden states of a dynamic system using noisy measurements. The Kalman filter has since then been one of the main algorithms used for tracking applications. This chapter presents the underlying system and measurement model of the Kalman filter, the Kalman filter algorithm, the extended Kalman filter algorithm, and finally a comparison of the Kalman filter to other tracking algorithms.

2.1 The Underlying System and Measurement Model

The Kalman filter is based upon two underlying linear dynamic models: a system model defining the changes in the hidden state of the system, and an observation model defining the relationship between the measurements and the actual state. Both models have been discretized in the time domain. For the purposes of tracking targets in space, the hidden state of the system is the position and velocity of the target along three axes. Thus, in vector form the state variable x_k is *{x position, y position, z position, x velocity, y velocity, z velocity}*.

The current state variable x_k is dependent on three other variables: the previous state, any other external force variables, such as gravity or drag, and some noise. These three components define the system model. The system model is as follows:

$$x_k = F_k x_{k-1} + B_k u_k + w_k, \tag{2-1}$$

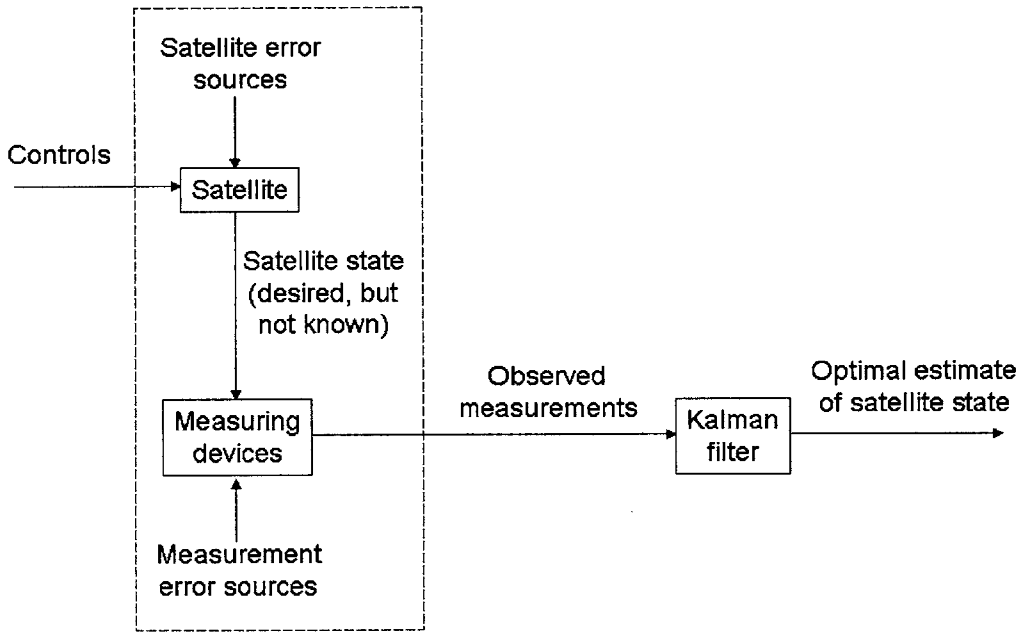
where k is the current time, x_k is the current state, $F_k\, x_{k-1}$ corresponds to the impact of the previous state, $B_k u_k$ corresponds to the impact of the external variables such as gravity and drag, and w_k corresponds to the target-control error sources such as turbulence or other random interactions in the air that would affect the motion of the target. These error sources are assumed to have a zero mean normal distribution and covariance of Q_k. Figure 2-1 shows a graphical perspective of all of the components of the two models.

The actual ladar measurements of the target are projections of the state vector x_k. For instance, when tracking an object in space, the desired quantities may be position vectors along the x, y, and z axis, but the ladar may receive measurements in azimuth, elevation and range. These measurements would have to then be converted to the position vector that the system requires. Furthermore, in any measurement device, there is always some margin of error due to the nature of the device, or the resolution of the measurement.

These characteristics of the measurement have been modeled so that an observation of the true state x_k, is in the form:

$$z_k = H_k x_k + v_k, \tag{2-2}$$

where again, k is the current time, z_k is the observation state, H_k is the projection of the state x_k at time k into the observational basis., and v_k represents the noise that is inherent in the measurement device with covariance R_k.



Satellite error sources
Controls
Satellite
Satellite state (desired, but not known)
Observed measurements
Optimal estimate of satellite state
Measuring devices
Kalman filter
Measurement error sources

Figure 2-1: General Concept of Tracking Systems

2.2 The Kalman Filter Algorithm

By using these models of the system, the Kalman filter can then predict where the target will be, given the current measurements. This prediction is done through two phases, using a form of feedback control. The filter estimates the hidden state at some time. Using measurements z_k, the Kalman filter corrects the estimate after each time step. Because of this, the equations used in the Kalman filter algorithm can be broken into two categories: time update equations and measurement update equations. The time update equations estimate the current state based upon the previous state, and the measurement update equations correct the estimate. This cycle continues as the results from the measurement update equations are then fed back into the time update equations, etc. It is important to note that in both of these phases, not only is the state being predicted and corrected, but also the error covariance of the state.

This tracking is done through two phases: (1) prediction, and (2) update. In the prediction phase, the state estimate from the previous time step is used to produce an estimate of the current state. The update phase then uses the measurement information from the current time step to refine the estimate of the state. In both of these phases, the error covariance is also propagated along with the state, thus both mean and covariance of each variable in the state vector is kept as the filter processes more data. Only the state and the error covariance are essential because the Kalman filter assumes that the system and measurement model noises are random variables that are Gaussian and zero-mean. Thus, the only pieces of information necessary to recreate the model would be the state and the error covariance of each of the variables.

The equations for the time update ("prediction" phase) are:

$$x_{k|k-1} = F_k x_{k-1|k-1} + B_k u_k, \tag{2-3}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k. \tag{2-4}$$

In these equations,$x_{k-1|k-1}$ describes the results from the previous measurement update, $x_{k|k-1}$ describes the results of the time update, $P_{k-1|k-1}$ is the estimated covariance from the previous measurement update, and $P_{k|k-1}$ is the estimated covariance after the time update. The use of F_k^T indicates that the transpose of F_k is being used. Again, these two equations represent the time update of both the state and the estimated covariance from time step $k-1$ to k. All variables in this section represent the same values as that of the previous section.

The equations for the measurement update ("update" phase) are:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R)^{-1}, \tag{2-5}$$

$$x_{k|k} = x_{k|k-1} + K_k (z_k - H_k x_{k|k-1}), \tag{2-6}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}. \tag{2-7}$$

In these equations, the difference $z_k - H_k x_{k|k-1}$ is called the measurement innovation, or the residual. The residual represents the difference between the predicted measurement $(H_k x_{k|k-1})$ and the actual measurement (z_k). The variable K_k is the Kalman gain chosen to minimize the error covariance equation after the measurement update.

The results from the measurement update are then fed back into the time update equations, and the process continues. The results from the time update can be used to predict future states, and then once measurements are made, the measurements can be used to update the states again.

2.3 The Extended Kalman Filter

One of the biggest limitations of the original Kalman filter is the assumption that the underlying system model is governed by a linear stochastic difference equation. In fact, the governing equations for free-flight targets in space are nonlinear. To address nonlinear equations of motion, an extended Kalman filter was developed – a filter that linearizes the equations of motion for each time step. Thus, the system model and measurement model for a nonlinear system can be defined respectively as:

$$x_k = f(x_{k-1}, u_k, w_k), \tag{2-8}$$

$$z_k = h(x_k, v_k), \tag{2-9}$$

where f and h are the functions that truly determine what the next state should be and what the measurements should be. In the extended Kalman filter, the Jacobian of these functions are used to linearize the nonlinearities.

The equations for the time update become:

$$x_{k|k-1} = f(x_{k-1}, u_k, w_k), \tag{2-10}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k, \tag{2-11}$$

where F_k is defined to be the Jacobian of f:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{x_{k-1|k-1}, u_k}. \tag{2-12}$$

The equations for the measurement update become:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R)^{-1}, \qquad (2\text{-}13)$$

$$x_{k|k} = x_{k|k-1} + K_k (z_k - h(x_{k|k-1}, 0)), \qquad (2\text{-}14)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}, \qquad (2\text{-}15)$$

where H_k is defined to be the Jacobian of h:

$$H_k = \left. \frac{\partial h}{\partial x} \right|_{x_{k|k-1}} . \qquad (2\text{-}16)$$

The use of the Jacobian linearizes the nonlinearity at that time step.

2.4 Comparison of the Extended Kalman Filters to Other Nonlinear Tracking Algorithms

The extended Kalman filter is one of three popular nonlinear tracking algorithms, the other two being particle filters and the unscented Kalman filter. Before comparing the three tracking algorithms, it is important to note the time constraints imposed by the ladar: the ladar allows the tracking filter 100 ms to compute the next state. Data is grabbed from the photon-counting detectors in 100 ms intervals. The processor used for the real-time tracking filter was an Intel Pentium M. The short time of 100 ms only allowed for simple processing and hence we decided to use the least computationally intensive filter – the extended Kalman filter.

Particle filters aim to estimate the hidden state of an object, based upon all previous data. While Kalman filters create first-order approximations given data from just the previous time step, the particle filters use all of the old data in order to develop unique probability density functions for the estimates. With more data points, or particles, the probability density function becomes much more accurate. Using particle filters, one can then deal with nonlinearities and non-Gaussian noise because the probability density function is fluid and can adapt to any scenario [4]. However, the tradeoff for this accuracy is that a great deal of computing power is required for the particle filters, and many samples are required for the particle filter to be more effective than just a normal Kalman filter. Both of these limitations in the particle filter make it difficult to use this particular method for photon-counting ladar systems. In order to achieve real-time, the particle filter would be harder to implement and maintain real-time computational speed. Furthermore, because the photon-counting ladar system aims to work at sub-photon levels, the amount of data given to make future predictions may not be enough for particle filters to function effectively.

The unscented Kalman filter uses a deterministic sampling technique to pick a set of data points around the hidden state. The filter propagates only these sample points through nonlinear functions in order to maintain the mean and covariance as the filter is tracking an object. The concept is similar to particle filters, except that the unscented Kalman filter does not retain all of the data points, but instead chooses a sample of points that can accurately depict the mean and covariance of the data. The unscented Kalman filter uses the assumption that the noise of the data is Gaussian, and thus only a few points are necessary [5].

The extended Kalman filter assumes much more about the data. The extended Kalman filter assumes a Gaussian noise model, and thus only the mean and covariance of the estimates are necessary from time step to time step. The extended Kalman filter linearizes the nonlinearities by using Jacobian matrices. The filter then uses a regular Kalman filter with the linearization. The filter is only a series of four equations that does not require old information about the target to be stored. Each step makes a prediction of the next step, which is then used to fine tune future steps. The extended Kalman filter has been used in previous studies of laser radar tracking as seen in Enders and Shapiro's theory of laser radar tracking [6]. In the end, the extended Kalman filter was chosen because (1) it is able to track free-flight objects in noise, (2) it is simple to implement, and (3) it is fast enough to perform all of these calculations in real-time.

3. Defining Kalman Filter Parameters

Before the Kalman filter can be implemented with the actual photon-counting ladar hardware, there are two major design considerations required to fit the Kalman filter to the particular application of tracking using both range and Doppler. The first of these design considerations is the effect Doppler information has on tracking, to determine whether it will be used in the measurement model. The second of these design considerations is the maximum tolerbale level of system noise if one wishes to accurately track all types of targets. This chapter explains the design of the Kalman filter and the definition of the parameters in the system and measurement model, as well as how Doppler information and system noise affect the accuracy of both range and Doppler tracking.

3.1 Defining System and Measurement Model Parameters

3.1.1 System Model Parameters

Given the parameters of the Kalman filter, each of the variables needed to be defined to fit the photon-counting ladar. As stated in the previous chapter, the state of the target is

$$x_k = \begin{bmatrix} x_{pos} \\ y_{pos} \\ z_{pos} \\ x_{vel} \\ y_{vel} \\ z_{vel} \end{bmatrix}. \qquad (3\text{-}1)$$

The three components that define the current state, are the previous state ($F_k\, x_{k-1}$), the control vector ($B_k u_k$) and some system noise (w_k).

The state transition model is defined to be

$$F_k = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3\text{-}2)$$

This model can be understood when multiplied with the previous state vector to get

$$F_k x_{k-1} = \begin{bmatrix} x_{pos} + dt * x_{vel} \\ y_{pos} + dt * y_{vel} \\ z_{pos} + dt * z_{vel} \\ x_{vel} \\ y_{vel} \\ z_{vel} \end{bmatrix} \tag{3-3}$$

where *dt* is the amount of time between time step *k-1* and *k*. One can see that the current state is the previous position affected by the velocity of the previous state.

The control vector, u_k, that is being modeled in the measurement vector is the acceleration of the object due to gravity and drag. Both of these are functions of the state vector.

$$u_k = \begin{bmatrix} x_{acc} \\ y_{acc} \\ z_{acc} \end{bmatrix}. \tag{3-4}$$

The control-input model, B_k, is then defined to be:

$$B_k = \begin{bmatrix} \frac{1}{2}dt^2 & 0 & 0 \\ 0 & \frac{1}{2}dt^2 & 0 \\ 0 & 0 & \frac{1}{2}dt^2 \\ dt & 0 & 0 \\ 0 & dt & 0 \\ 0 & 0 & dt \end{bmatrix}. \tag{3-5}$$

which, when multiplied to the control vector applies the acceleration information to the state information:

$$B_k u_k = \begin{bmatrix} \frac{1}{2}x_{acc}dt^2 \\ \frac{1}{2}y_{acc}dt^2 \\ \frac{1}{2}z_{acc}dt^2 \\ x_{acc}dt \\ y_{acc}dt \\ z_{acc}dt \end{bmatrix}. \tag{3-6}$$

The system model noise is modeled by its covariance, Q_k. Because of the assumption that noise is Gaussian, only the covariance needs to be propagated. Zarchan [7] shows that Q_k can be determined by

$$Q_k = \int_0^{dt} F(t) Q_C F(t)^T dt. \qquad (3\text{-}7)$$

where $F(t)$ is defined to be the state transition matrix as shown before with the dt's replaced by the time variable t. Q_c refers to the variances of the largest sources of noise. Because the state values of velocity are a much higher source of error, Q_c is

$$Q_c = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{noise} & 0 & 0 \\ 0 & 0 & 0 & 0 & S_{noise} & 0 \\ 0 & 0 & 0 & 0 & 0 & S_{noise} \end{bmatrix} \qquad (3\text{-}8)$$

where S_{noise} represents the variance of the possible noise in the system. When integrating the entire system, the covariance of the system model noise, Q_k, becomes

$$Q_k = \begin{bmatrix} \frac{1}{3}dt^3 S_{noise}{}^2 & 0 & 0 & \frac{1}{2}dt^2 S_{noise}{}^2 & 0 & 0 \\ 0 & \frac{1}{3}dt^3 S_{noise}{}^2 & 0 & 0 & \frac{1}{2}dt^2 S_{noise}{}^2 & 0 \\ 0 & 0 & \frac{1}{3}dt^3 S_{noise}{}^2 & 0 & 0 & \frac{1}{2}dt^2 S_{noise}{}^2 \\ \frac{1}{2}dt^2 S_{noise}{}^2 & 0 & 0 & dtS_{noise}{}^2 & 0 & 0 \\ 0 & \frac{1}{2}dt^2 S_{noise}{}^2 & 0 & 0 & dtS_{noise}{}^2 & 0 \\ 0 & 0 & \frac{1}{2}dt^2 S_{noise}{}^2 & 0 & 0 & dtS_{noise}{}^2 \end{bmatrix} \qquad (3\text{-}9)$$

This variable, S_{noise}, can be tweaked to account for small nonlinearities that occur during tracking. However, while tweaking S_{noise} can cause the tracking algorithm to recover from nonlinearities more easily, the accuracy of the tracking estimates decreases because a greater covariance error has been provided.

3.1.2 Measurement Model Parameters

The measurement model represents how the actual state is viewed by the measurement device. The available measurements from photon-counting ladar are azimuth, elevation, range and body Doppler of the target. All four of these measurements are used for the tracking algorithm. A discussion of this decision is detailed in Section 3.2. The measurement state is:

$$z_k = \begin{bmatrix} azimuth \\ elevation \\ range \\ doppler \end{bmatrix}. \tag{3-10}$$

To make all of the equations easier to develop and derive, a transformation is used to convert the measurement state from an Earth-Centered Coordinate (ECC) system to a Body-Centered Coordinate (BCC) system. In an ECC system, the x-, y-, and z-axis are fixed with respect to the earth. In a BCC system, the z-axis lies along the line of sight from the observer to the target, and the x and y-axis lie perpendicular to the z-axis. Thus, the location of the target determines where the coordinate system is. After all of the calculations, the BCC coordinates are transformed back into ECC coordinates. All of the following equations are written in context of the BCC system.

The observation model that maps the true target state to the measurement is a Jacobian matrix, used to linearize the nonlinearities of this transformation. The matrix is thus:

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{x_{k|k-1}}. \tag{3-11}$$

where x is the state vector and h is defined to be the:

$$h = \begin{bmatrix} \arctan(\frac{x_{pos}}{z_{pos}}) \\ \arctan(\frac{y_{pos}}{z_{pos}}) \\ \sqrt{x_{pos}^2 + y_{pos}^2 + z_{pos}^2} \\ z_{vel} \end{bmatrix}. \tag{3-12}$$

This definition causes H_k to be:

$$H_k = \begin{bmatrix} \dfrac{1}{1+\dfrac{x_{pos}^2}{z_{pos}^2}} * \dfrac{1}{z_{pos}} & 0 & -\dfrac{x_{pos}}{z_{pos}^2} * \dfrac{1}{1+\dfrac{x_{pos}^2}{z_{pos}^2}} & 0 & 0 & 0 \\ 0 & \dfrac{1}{1+\dfrac{y_{pos}^2}{z_{pos}^2}} * \dfrac{1}{z_{pos}} & -\dfrac{y_{pos}}{z_{pos}^2} * \dfrac{1}{1+\dfrac{y_{pos}^2}{z_{pos}^2}} & 0 & 0 & 0 \\ \dfrac{x_{pos}}{\sqrt{x_{pos}^2+y_{pos}^2+z_{pos}^2}} & \dfrac{y_{pos}}{\sqrt{x_{pos}^2+y_{pos}^2+z_{pos}^2}} & \dfrac{z_{pos}}{\sqrt{x_{pos}^2+y_{pos}^2+z_{pos}^2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} . \quad (3\text{-}13)$$

Because the measurement state is of the form *{azimuth, elevation, range, Doppler}*, the covariance of the measurement model noise, R_k is:

$$R_k = \begin{bmatrix} error_{azimuth} & 0 & 0 & 0 \\ 0 & error_{elevation} & 0 & 0 \\ 0 & 0 & error_{range} & 0 \\ 0 & 0 & 0 & error_{doppler} \end{bmatrix} . \qquad (3\text{-}14)$$

The derivation of the errors $\text{error}_{\text{azimuth}}$, $\text{error}_{\text{elevation}}$, $\text{error}_{\text{range}}$, and $\text{error}_{\text{doppler}}$, are explained in the next section.

3.1.3 Measurement Errors

The values $\{error_{azimuth}, error_{elevation}, error_{range}, error_{doppler},\}$ in equation (3-14) are defined as $\{\Delta\varphi^2, \Delta\theta^2, \Delta R^2, \Delta v^2\}$, the variance of the azimuth angle, elevation angle, range, and velocity measurements for a given time step.

The pointing angular accuracy of modern astronomical telescopes is about 10 μrad. In our simulations, we use $\Delta\varphi = \Delta\theta = 10\ \mu rad$. This is an approximation since the actual angular measurement resolution improves as the time step, *dt*, for the Kalman filter increases. As *dt* increases, the object imaged onto an auxiliary tracking CCD sensor contains a stronger signal and hence can be centroided with better accuracy. Since the angular accuracy is highly dependent on the particular telescope that the ladar is attached to and since we are mainly interested in the ladar -- not the telescope -- tracking performance, a rough estimate of the angular accuracy is sufficient for the analysis presented in this thesis.

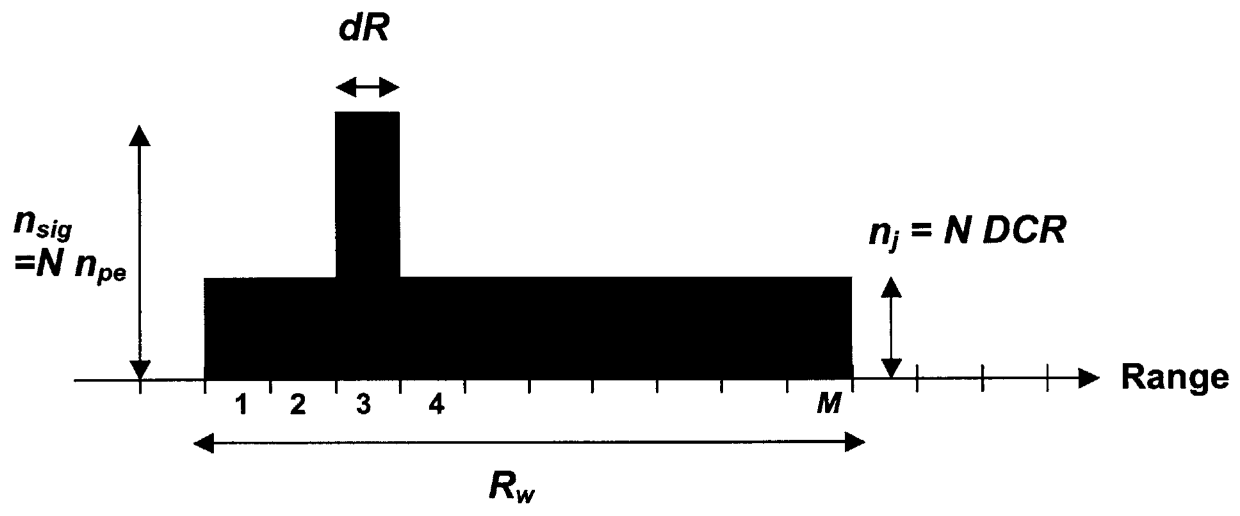


Figure 3-1: Histogram of corrected range returns averaged over N pulses

The range measurement error is determined by the range extent of the target, the return signal strength, the dark count rate of the detectors, the range window, and the number of return pulses used for range estimation. Figure 3-1 shows the histogram of range returns averaged over N pulses, where each frame is correctly shifted to correct for movement of the target. The mean values are displayed for each bin. We estimate the location of the target by finding the peak value of the histogram. Typical values for the photon-counting ladar are $N = 100$ pulses, dark count rate (DCR) = 500 Hz per pixel, target range extent of dR = 10 m, number of received photoelectrons per pulse of $n_{pe} = 1$, and a range window of R_w = 1 km. For these typical values, $N\, n_{pe} = 100$, and $N\, DCR\, (2dR/c) = 0.003$ – which shows that for this signal level the peak is 100/0.003 = 30,000 times greater than the background noise level. It should be noted that if the unresolved target is imaged onto multiple pixels, the dark count rate increases proportionally to the number of illuminated pixels – the best receiver design indicates that there should only be one pixel per resolution element.

To derive the range measurement error, we assume that the number of counts in each range bin in the histogram shown in Figure 3-1 follows a Poisson probability mass function with mean value n_j, where $j = 1..M$ is the index of the range bin. The number of range bins is $M \equiv R_w/dR$. For the range bin where the signal is located, $n_{sig} = N\, n_{pe}$. For the range bins that don't contain the signal, $n_j = N\, DCR\, (2dR/c)$. The range measurement error is

$$\Delta R = dR \cdot P + M/3 \cdot (1-P), \qquad (3\text{-}15)$$

where P is the probability that $n_{sig} > n_1, n_2, \ldots, n_M$. The value $dR \cdot P$ in equation (3-15) is the measurement error of the range when we correctly identify the peak. In this case, the range resolution is given by the range extent of the target. The value $M/3 \cdot (1-P)$ in equation (3-15) is the measurement error of the range when our peak estimator accidentally chooses a bin corresponding to dark count noise. When this happens, the value of the range error is huge and equal to $M/3$. Next, we evaluate P and derive $M/3$.

The value P is equal to prob$(n_1 < n_{sig}) \cdot prob(n_2 < n_{sig}) \cdot \ldots \cdot prob(n_M < n_{sig})$. The Poisson probability mass function for n_j (where $j = 1..M$) is defined as $pdf_j(k) = exp(-n_j)\, n_j^k / k!$ and its cumulative distribution function is $poisscdf_j(k)$. Therefore, $P = poisscdf_1(n_{sig}) \cdot poisscdf_2(n_{sig}) \cdot \ldots \cdot poisscdf_M(n_{sig}) =$

$$P = [poisscdf(n_{sig})]^M \tag{3-16}$$

since $n_1 = n_2 = \ldots = n_M$.

The factor of $M/3$ in equation (3-15) is derived as follows: Assume that we have M buckets that are placed in a line. A ball thrown at the buckets is equally likely to land in any one of them. If we through a black ball and a white ball into the buckets, what is the expected value of the distance between the balls? The white ball represents the location of the actual signal and the black ball represents the background noise bin. The expectation value is equal to the sum of the probability of the black and white balls landing in two particular bins $(1/M^2)$ times the distance between those bins. We consider all combinations of bins that the black and white balls can occupy. This expectation value is equal to

$$\begin{aligned}
&\frac{1}{M^2}\left[0+1+2+3+\ldots+(M-1)\right]+\\
&\frac{1}{M^2}\left[1+0+1+2+\ldots+(M-2)\right]+\\
&\frac{1}{M^2}\left[2+1+0+1+\ldots+(M-3)\right]+ .\\
&\ldots\\
&\frac{1}{M^2}\left[(M-1)+\ldots+3+2+1+0\right]
\end{aligned} \tag{3-17}$$

The first line of (3-17) corresponds to the black ball in the first bin and the white ball in the $1..M^{th}$ bins. The second line of (3-17) corresponds to the black ball in the second bin and the white ball in the *1..M*th bins. Each subsequent line, the black ball is moved over one bin to the right until it occupies the last bin. The sum of the upper right part of the matrix (the numbers above the zeros along the trace) is equal to the sum lower left part of the matrix. The sum of the upper right part of the matrix is equal to $M(M-1)/2 + (M-1)(M-2)/2 + (M-2)(M-3)/2 + \ldots + 1 = (M/6)\,(M^2-1)$. Therefore, the sum in (3-17) simplifies to $1/M^2 \cdot 2 \cdot (M/6)\,(M^2-1) = (M^2-1)/(3M)$. For all practical cases of interest, M>>1, and the sum in (3-17) simplifies to M/3.

Substituting (3-16) into equation (3-15) results in our final expression for the range resolution:

$$\Delta R = \frac{M}{3} + \left(dR - \frac{M}{3}\right)\left[poisscdf\left(n_{sig}\right)\right]^{M} . \qquad (3\text{-}18)$$

The range resolution is tabulated for several values of M, dR, n_j, and n_{sig} in table 3-1. The table shows that the range measurement error is limited by the range extent of the target, even when the return signal yields 1 photoelectrons per pulse. Using the value of ΔR = 15 cm is valid down to 1 photoelectrons per pulse.

Table 3-1: Range measurement error as a function of system parameters

ΔR	M	n_{sig}	n_j	dR
16.05 m	100	0.1	0.003	10 m
10.01 m	100	1.0	0.003	10 m
10.00 m	100	10	0.003	10 m
875.1 cm	100	0.1	0.003	15 cm
16.5 cm	100	1.0	0.003	15 cm
15.0 cm	100	10	0.003	15 cm

The derivation of the velocity measurement error is analogous to that of the range measurement error. Instead of range bins, we consider velocity bins. Analogous to the range measurement case, we find that to a good approximation the velocity error can be taken to be equal to the velocity measurement resolution of 2 cm/s. This velocity resolution corresponds to a transmit pulse width of 25 μs.

Table 3-2: Values used for Kalman Filter

$\Delta\varphi$	10 μrad	$\text{error}_{\text{azimuth}}$	10^{-10} rad^2
$\Delta\theta$	10 μrad	$\text{error}_{\text{elevation}}$	10^{-10} rad^2
ΔR	15 cm	$\text{error}_{\text{range}}$	225 cm^2
Δv	2 cm/s	$\text{error}_{\text{velocity}}$	4 cm^2/s^2

3.1.4 Effect of the "Joseph" form

One change that was made in our Kalman filter models is in the equation to determine the update of the error covariance matrix, $P_{k|k}$. Instead of the equation

$$P_{k|k} = (I - KH) * P_{k|k-1}, \tag{3-19}$$

found in most textbooks, we used the equation

$$P_{k|k} = (I - KH) * P_{k|k-1} * (I - KH)^T + K * R_k * K^T. \tag{3-20}$$

This form, called the "Joseph" form, is considered to be numerically more stable because it preserves positive definite-ness and symmetry in the covariance matrix [8]. In our simulations, the use of equation 3-20 centered the residual errors of the tracking algorithm better than when we used equation 3-19. Table 3-3 shows the statistics of one particular simulation of an unaffected free-flight trajectory. As can be seen the use of the "Joseph" form centers the residual errors around zero better, although there is a slight increase in the standard deviation of the error.

Table 3-3: The effect of using the "Joseph" form for the update of the error covariance matrix. Statistics are given of the range track residual errors.

	Minimum	Mean	Maximum	Standard Deviation
With "Joseph" form	-0.5550	0.0442	0.5894	0.1764
Without "Joseph" form	-0.3372	0.1722	0.7343	0.1696

3.2 Effect of Doppler Information in Measurement Model

Four pieces of information are obtained from the ladar system: azimuth, elevation, range and Doppler. Currently, many radar and ladar systems use just azimuth, elevation and range information when tracking an object. Because the capability of measuring Doppler information is available, it is important to understand whether Doppler information could improve the tracking accuracy.

To understand better the effects of Doppler, two separate Kalman filter algorithms were simulated in MATLAB: one with the measurement information in the form *{azimuth, elevation, range}* and another with *{azimuth, elevation, range, Doppler}*. Both models were used to track the trajectory of a free-flight target that follows a nearly parabolic trajectory. The trajectory is plotted in Figure 3-2 and was generated by a separate program that modeled the gravity as a function of altitude but did not include drag.

The Kalman filter time steps were set to $dt = 0.1$ s to match the update rate of the photon-counting ladar. Figure 3-3 shows the range residual errors between the model's predictions and the actual trajectory of the target, while Figure 3-4 shows the velocity residual errors. The spread of the range errors is essentially constant versus time. The range estimate is independent of time since the range accuracy is assumed to have a constant value. In addition, the range estimate does not depend on telescope pointing accuracy. On the other hand, Figure 3-4 of the velocity errors shows that the errors decrease linearly with time because the measurement errors of x and y (perpendicular to the line of sight) decrease with range as $S=R\theta$, where θ is the angular error of the telescope. Table 3-4 shows the statistics for the plots in Figure 3-3 and Figure 3-4.

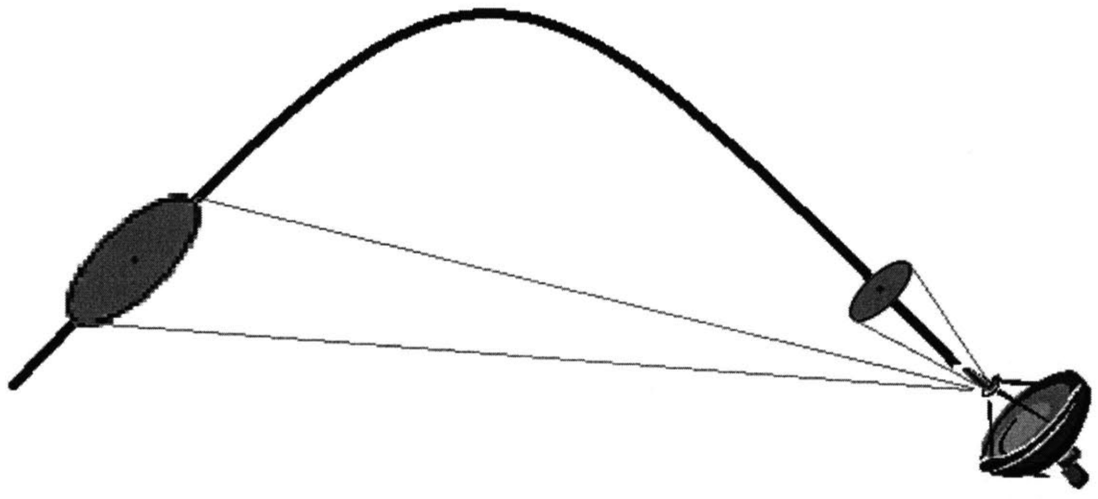
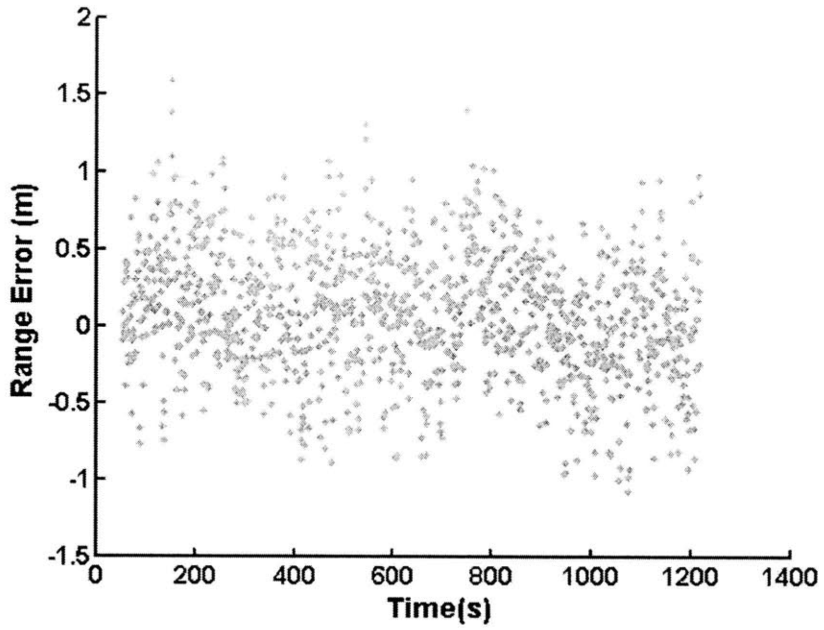
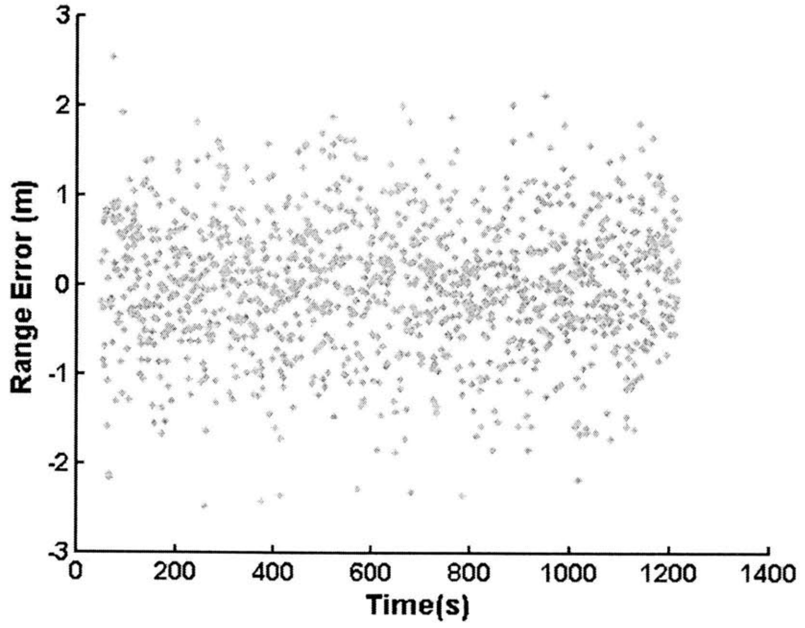


Figure 3-2: Example of parabolic trajectory, location of observer and measurement error ellipse for both far and near points on the trajectory.

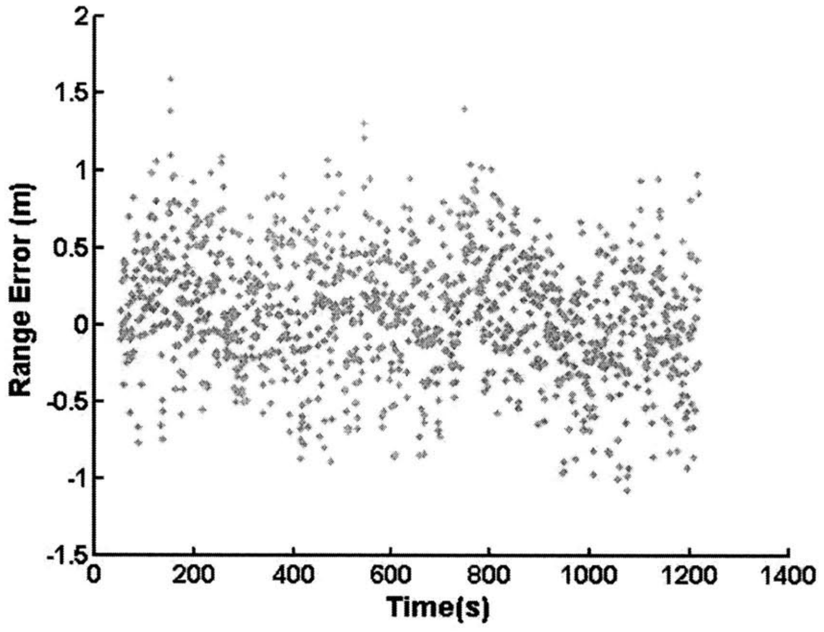


(a)

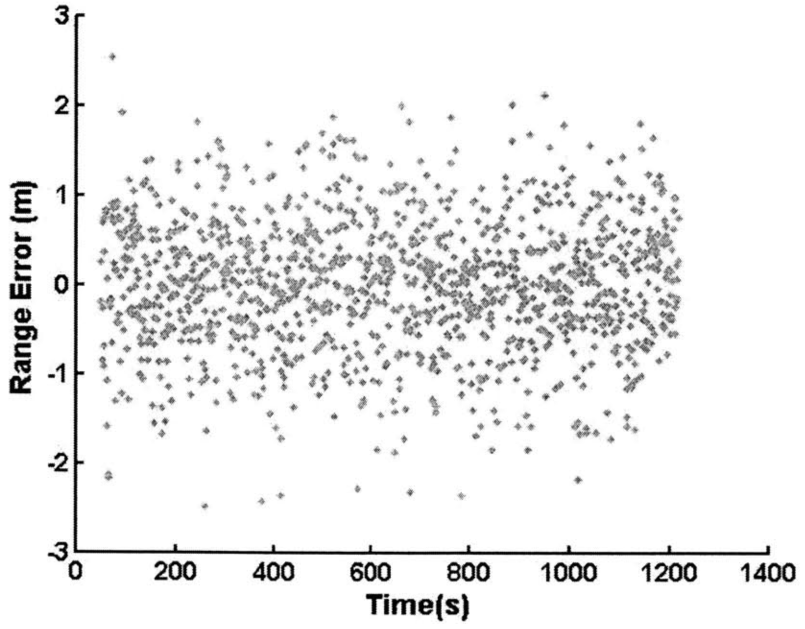


(b)

Figure 3-3: Residual range error (a) with Doppler information and (b) without Doppler information



(a)



(b)

Figure 3-3: Residual range error (a) with Doppler information and (b) without Doppler information

Table 3-4: Statistics on residual errors for range and velocity for the (a) model with Doppler information and the (b) model without Doppler information.

	Minimum	Mean	Maximum	Standard Deviation
Range residual error	-1.2337 m	0.0791 m	1.5924 m	0.4198 m
Velocity residual error	-4.0773 m/s	-0.0144 m/s	3.3523 m/s	0.9403 m/s

(a)

	Minimum	Mean	Maximum	Standard Deviation
Range residual error	-2.4780 m	0.0066 m	2.5367 m	0.7518 m
Velocity residual error	-3.9078 m/s	-0.0318 m/s	4.2040 m/s	1.1299 m/s

(b)

All velocities in this thesis will be magnitudes of velocities as opposed to velocities measured along line of sight, unless specifically noted. From these figures and table, one can see that the Doppler information only slightly improves the range track and the velocity track of the Kalman filter by reducing the standard deviation of the residual error. The standard deviation of the range error improves from 0.7518 m to 0.4128 m when using Doppler information. This indicates that just using range information is sufficient for tracking when the range measurement resolution is 15 cm. It is worth emphasizing that the Kalman filter is very accurate at predicting future states: the estimate of a future location (1 second into the future) is only 0.7518 m off after traveling 4 km (velocity of the target is 4 km/s for this trajectory).

Further software simulations were performed to better understand how the measurement noise covariance of the Doppler from the photon-counting ladar system affects the tracking ability of the Kalman filter. We wanted to better understand how the Doppler measurement resolution affected the tracking for both range and velocity. Multiple software simulations were performed with velocity measurement resolutions that ranged from 10^{-3} m^2/s^2 to 10 m^2/s^2. The velocity measurement resolution, Δv, is related to the Doppler measurement resolution, Δf, according to $\Delta f = 2\,\Delta v\,/\lambda$. The log of the velocity measurement resolution was plotted against the standard deviation of the range and Doppler estimate error. The standard deviation is evaluated over all time steps. Figure 3-5 and Figure 3-6 show the range and Doppler error, respectively. These plots show that the accuracy of the range and velocity track do depend on the accuracy of the Doppler information.

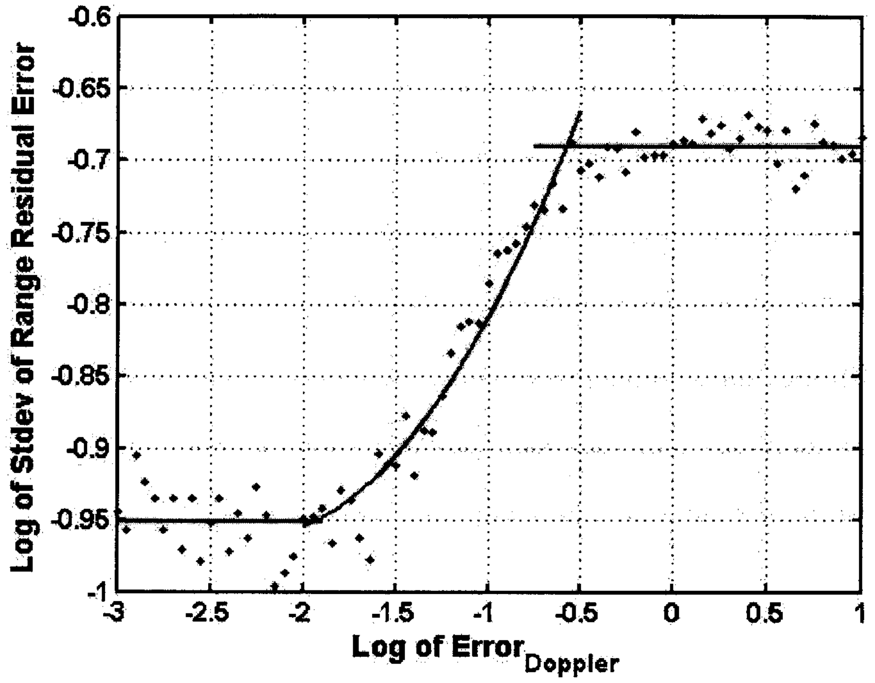


Figure 3-5: Standard deviation of the range errors from the Kalman filter as a function of the Doppler measurement resolution as seen in Eq. 3-14.

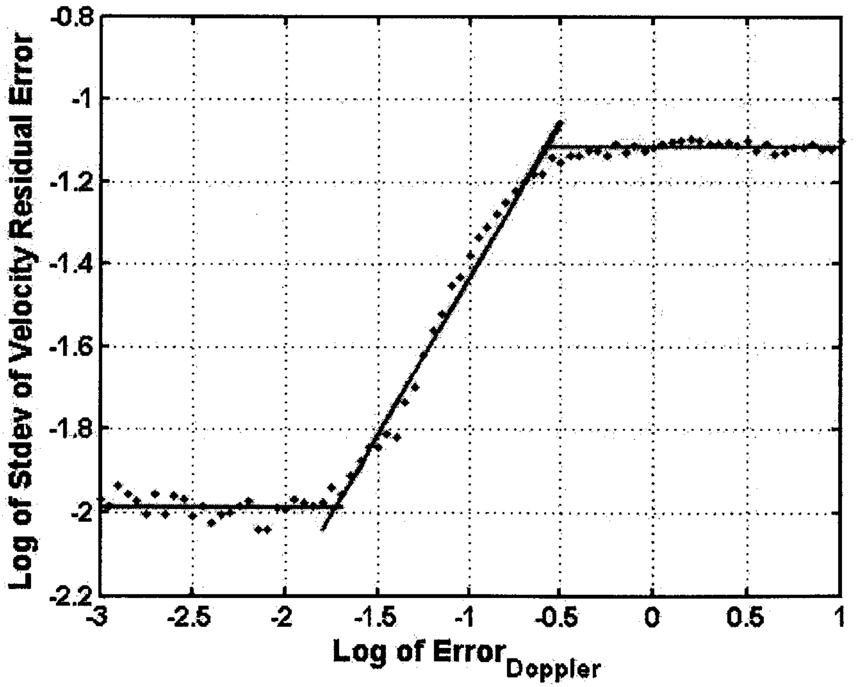


Figure 3-6: Standard deviation of the velocity errors from the Kalman filter as a function of the velocity measurement resolution as seen in Eq. 3-14..

When examining the error of range track estimates based on Doppler information, as shown in Figure 3-5, there are three regions of importance: the baseline values at very low Doppler error covariance, the plateau values at very high Doppler error covariance, and the transition period in between. When the Doppler measurement resolution is above 0.3162 m/s, the standard deviation of the residual errors of the range track no longer get worse. This characteristic is due to the fact that the velocity measurement resolution is so poor that the range measurements provide better velocity information than the Doppler measurements. For Figure 3-5, the range measurement resolution was kept constant at 15 cm.

On the opposite side, when the velocity measurement resolution is below 0.01 m/s, the standard deviation of the residual errors of the range track no longer get any better. This feature exists because the system noise of the model, S_{noise} and the range measurement errors were kept constant while the doppler error was variable. Because of the errors in the range measurement and the system noise inherent in the model, the accuracy of the overall range and velocity track can only a certain level, and will not be completely zero. As S_{noise} decreases, this saturated level will also decrease.

The transition period between these two regions has a quadratic form, where the standard deviation of the range errors scales quadratically with velocity measurement resolution. The quadratic form of this relationship is expected because range is the derivative of velocity along line of sight.

Similar observations can be made concerning the line-of-sight velocity error as a function of velocity measurement resolution (see Figure 3-6). When the velocity measurement resolution is above 0.316 m/s, the standard deviation of the velocity error of the velocity track no longer gets worse. This characteristic is because the Kalman filter is only using range information similar to the range track. Similar to the range track, when the Doppler error covariance is below -0.0158 m/s the Doppler track levels out. The region in between the two extremes is linear. This characteristic is expected because the Doppler information provided from the laser radar system should be the same as the Doppler state of the actual target.

In summary, Doppler information does not significantly improve the ability to track the state of the target, x_k. Since the range measurement resolution is very good, there is little to be gained from Doppler estimates to improve the accuracy of the track.

3.3 System Noise Parameterization

Using the Kalman filter model that includes the Doppler information, three different trajectories were used to determine the optimal value for the system noise parameter. As stated before, system noise characterizes the possible deviation of the target from the path that the Kalman filter has modeled. Thus, when the system noise parameter in the tracking algorithm model is large, the Kalman filter is able to account for deviations from the expected paths more easily; however the errors that the Kalman filter will allow also increase. This characteristic will be made clearer as we examine the effects of system noise on the tracking of three targets: a free-flight target with no external forces, a free-flight target with a booster that fires mid-flight, and a free-flight target that tumbles. As we examine the normal free-flight target, we will first examine the effect of system noise on the range track and then we will look at its effect of system noise on the velocity track.

3.3.1 Analysis of Free-Flight Trajectory

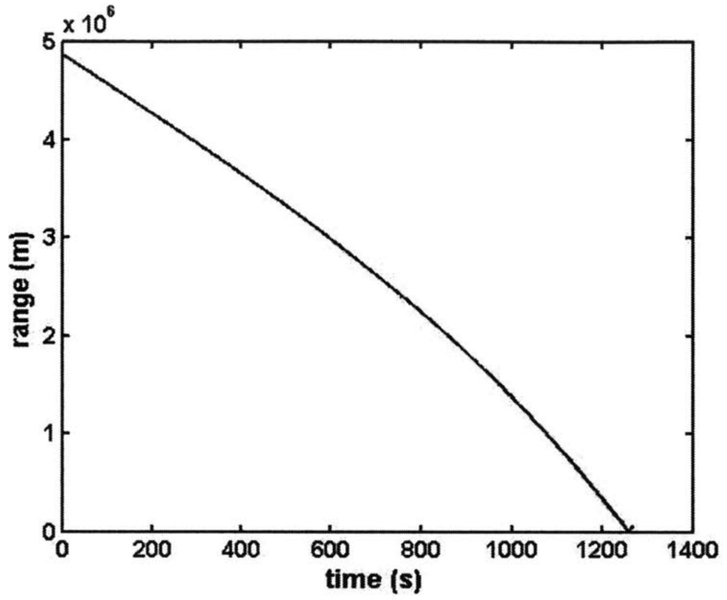
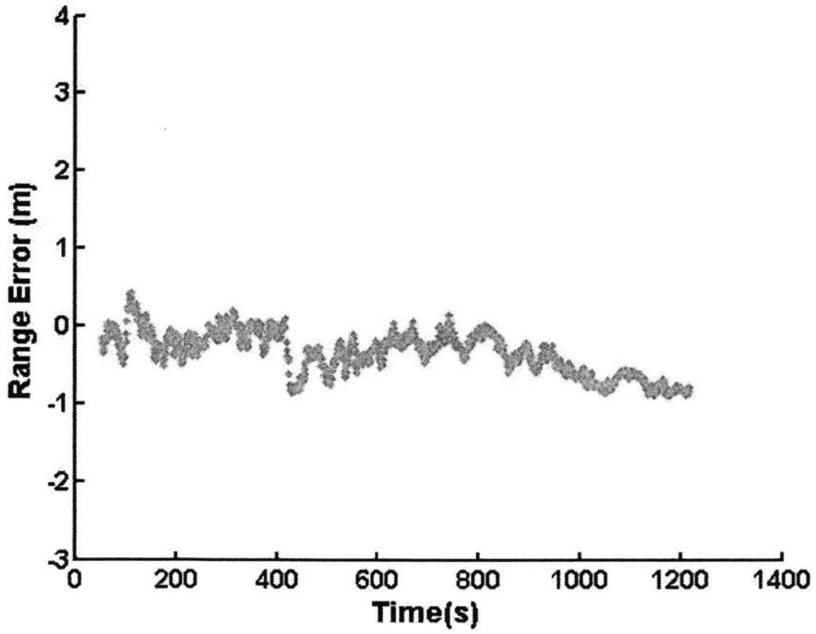
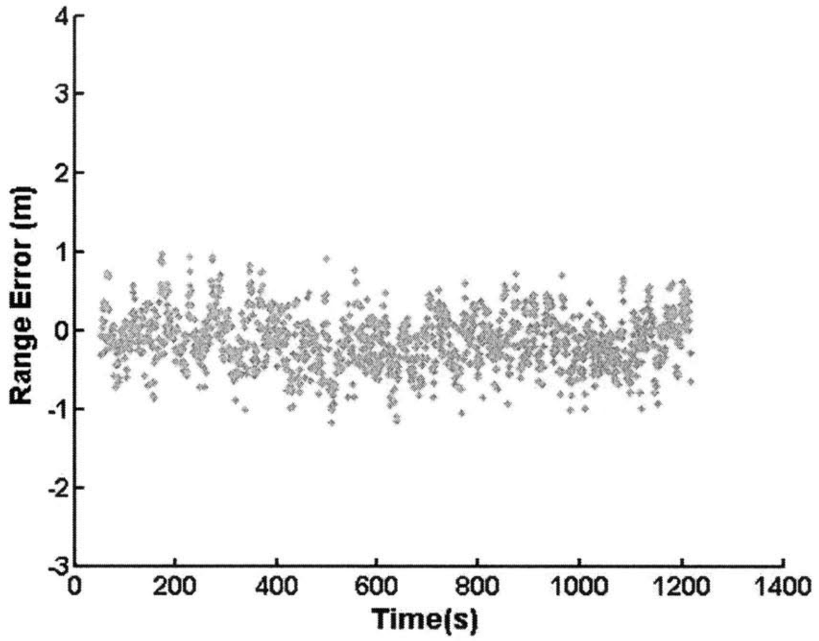


Figure 3-7: Range trajectory of free-flight target

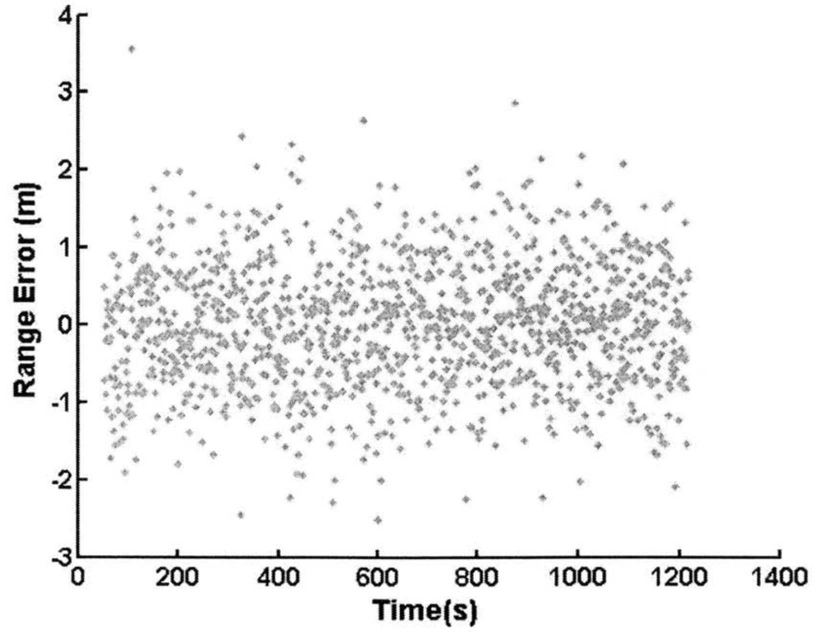
Figure 3-7 shows the trajectory of the target being tested. As one can see, the target begins at 4.8 million meters, and approaches the target at slightly less than 4 km/s.



(a)



(b)



(c)

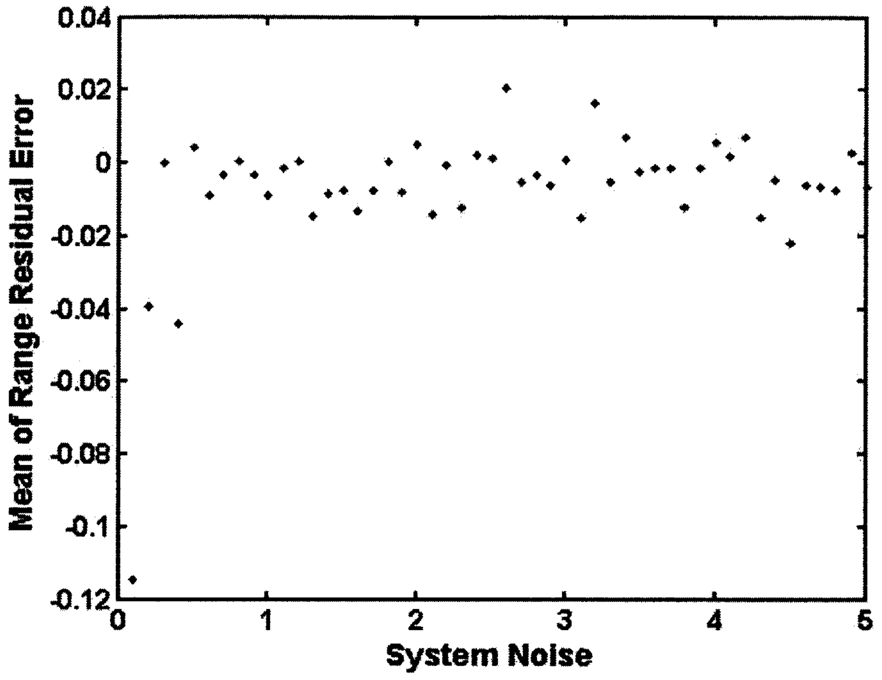
Figure 3-8: Range tracking residual error for system noise, S_{noise}, values at (a) 0.05 (b) 0.5 and (c) 5. Range error is equal to the actual range minus the estimated range at a given time step.

Table 3-5: The effect of system noise, S_{noise}, on the distribution of range residual error

System Noise	Min Error	Mean Error	Max Error	Std Dev of Error
0.05	-0.3146	-0.0532	0.2540	0.1015
0.5	-0.7147	-0.0053	0.5666	0.1941
5	-0.8927	-0.0035	0.9565	0.2960

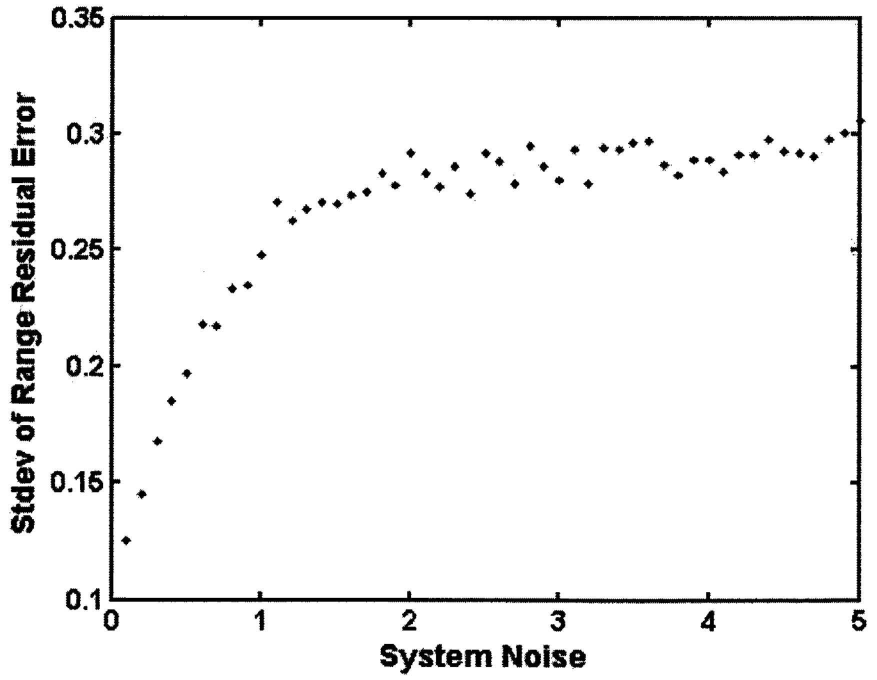
From Figure 3-8, one can see the effects of system noise, S_{noise}, at both extremes. Table 3-5 shows the statistics of each of these plots. When the system noise is low, the range residual error oscillates. This characteristic can be explained because the model's system noise is very small, any deviation of the measurement from the expected, causes the tracking algorithm to overstep the actual trajectory. The Kalman filter tries to match the measurements that it receives with the ballistic trajectory that it believes the object should be flying at. The system noise value defines a boundary around the expected trajectory of 'allowed' points, meaning that the measured values can fall in these areas and the ballistic trajectory would still be valid. When the system noise is too small, that area decreases in size, and thus the Kalman filter expects the measured data to follow the model. When it does not, the Kalman filter tries to reconstruct a new ballistic trajectory, still assuming the ballistic system model it's been given is still valid.

As the system noise increases, the deviations from the measurements still fall within the model's system noise range, and thus the entire track does not oscillate as quickly. However, as the system noise increases the standard deviation of the range residual error increases as well. Again, this is expected because as the model's system noise increases, the anticipated error increases, and thus greater deviations from the actual trajectory are allowed, allowing the track to be more lenient. In examining these characteristics of system noise, we wanted to minimize the standard deviation of the range residual error, while keeping the oscillation minimal. Figure 3-9 shows the relationship of the system noise to the mean and standard deviation of the range and velocity residual track errors.



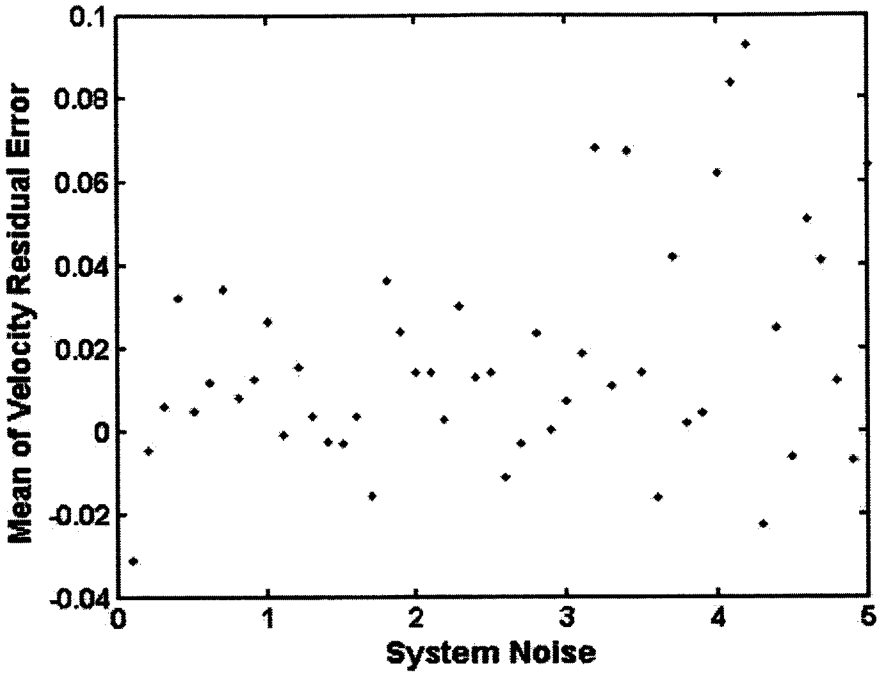
Mean of Range Residual Error
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
0
1
2
3
4
5
System Noise

(a)

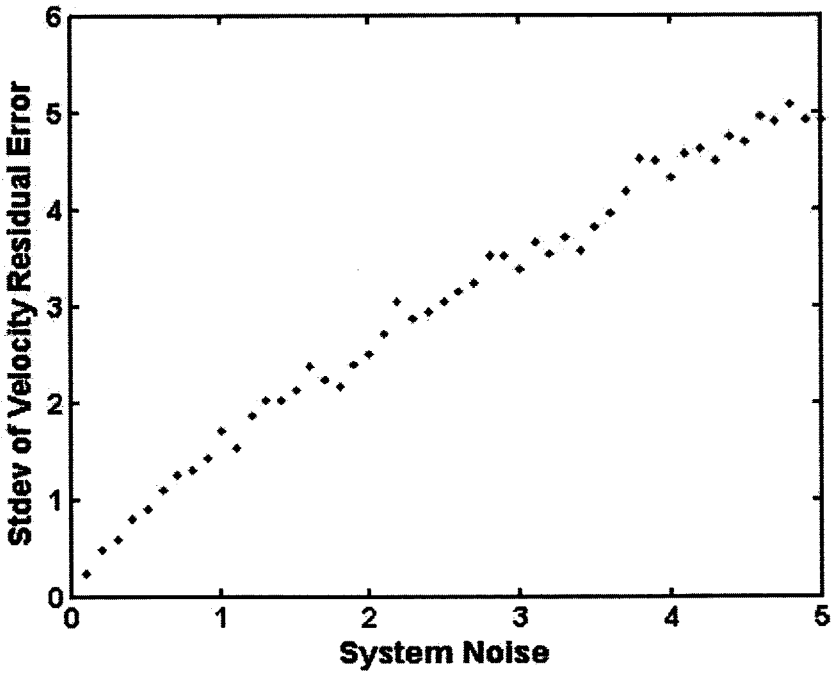


Stdev of Range Residual Error
0.35
0.3
0.25
0.2
0.15
0.1
0
1
2
3
4
5
System Noise

(b)



(c)

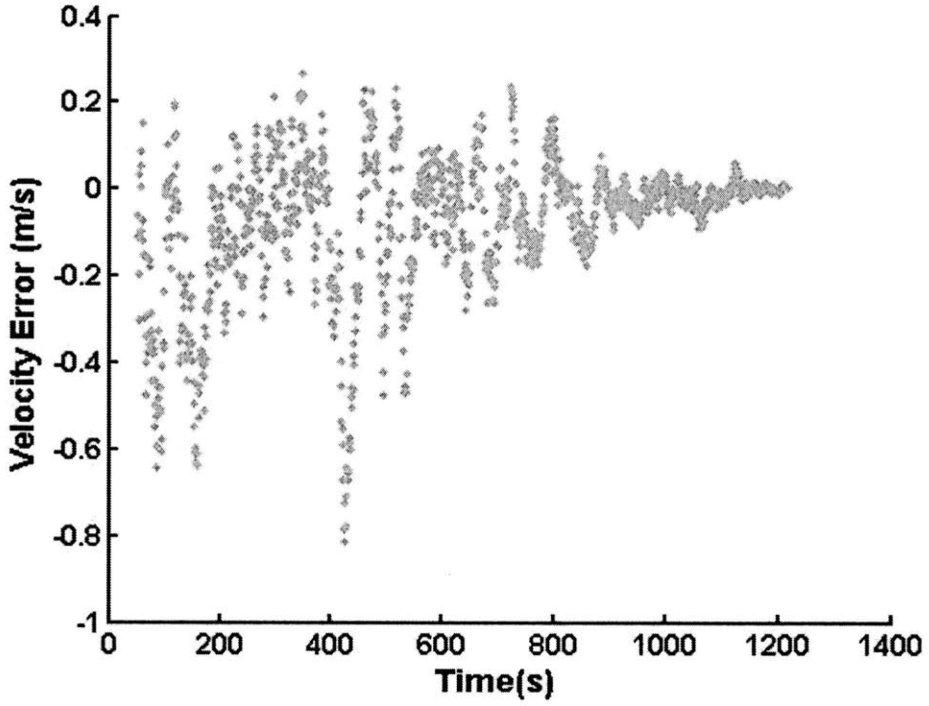


(d)

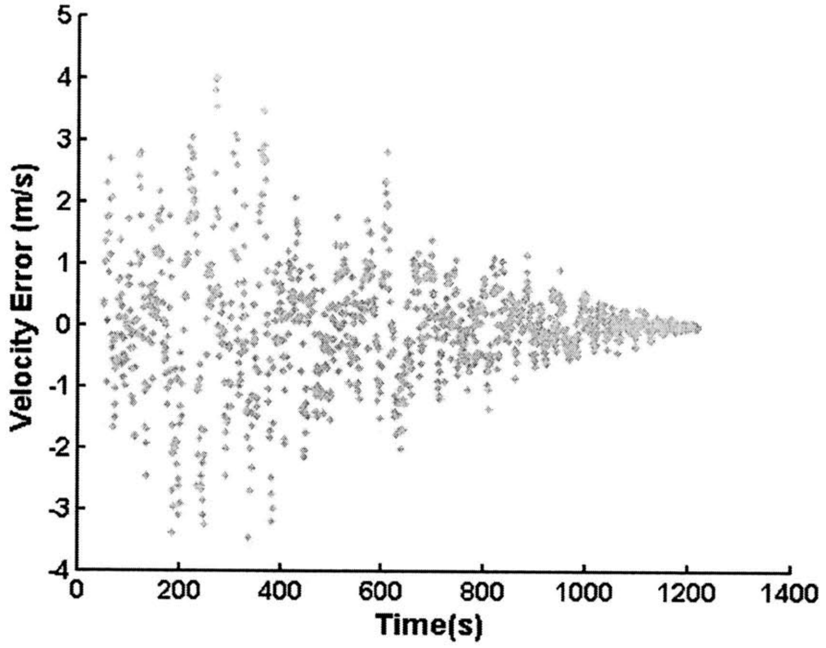
Figure 3-9: Relationship between system noise and range residual error (a) mean and (b) standard deviation, as well as the velocity residual error (a) mean and (d) standard deviation.

The mean of both of the tracks are roughly around zero. The velocity residual track errors begin moving around quite a bit. This could be an indication that our model is not completely correct, and that there is a slight biased in the model. The other possibility is that because we're measuring the magnitude of the entire velocity from two different locations (estimated target location and actual target location) a bias could exist. The standard deviations of both range and velocity are increasing, though the standard deviation of the range levels off, while the velocity error continues to increase. This characteristic will be explained later in this section.

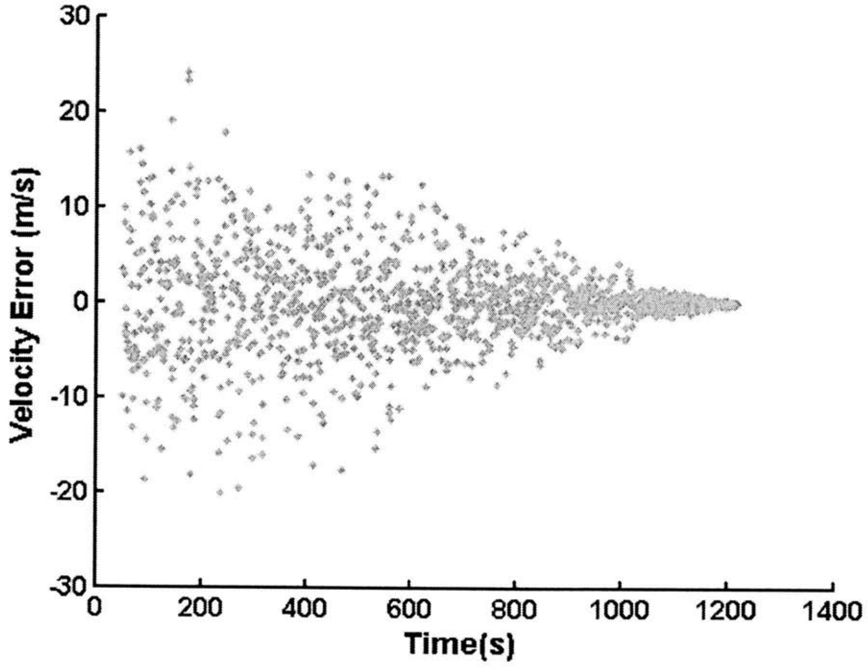
We thus chose the system noise set to $S_{noise} = 0.5$.



(a)



(b)



(c)

Figure 3-10: Velocity tracking residual error for system noise, S_{noise}, values at (a) 0.05 (b) 0.5 and (c) 5

Figure 3-10 shows the velocity tracking residual error at different system noises. Again, similar features are apparent, the oscillations at the low system noise levels, and the increase in standard deviations at higher system noise levels. One notable characteristic in the velocity tracking residual error that is not seen in the range tracking counterpart is the decreasing standard deviation. Because of the decreasing standard deviation, both the standard deviation as a whole, and the last 100 points, which appear to be stable have been shown in Table 3-6.

Table 3-6: The effect of system noise on the distribution of velocity residual error over (a) the entire time scale and (b) the last 100 points

System Noise	Min Error	Mean Error	Max Error	Std Dev of Error
0.05	-0.4124	-0.0032	0.3929	0.1275
0.5	-4.6246	-0.0129	4.1561	0.9496
5	-29.1056	-0.0067	23.6532	5.4527

(a)

System Noise	Min Error	Mean Error	Max Error	Std Dev of Error
0.05	-0.0239	-0.0013	0.0466	0.0133
0.5	-0.2387	-0.0165	0.1548	0.0665
5	-0.9656	-0.0072	0.9184	0.3561

(b)

Comparing these graphs to the trajectory, we can see that as the distance from the observer to the target decreases, the standard deviation of the residual decreases as well. This characteristic can be understood by realizing that the error of the velocity of the target is a combination of the error in both angle and range. When the target is further away from the observer, the effect of the angle error is much larger than when the target is closer. The observation that the standard deviation is proportional to the range also supports that hypothesis. The standard deviation does not react in the same way to the range information because the range information is not dependent on the angle measurements, but only the range information. This characteristic again supports the decision to choose a smaller system noise value so that the effect of the angle error is not as large.

3.3.2 Analysis of Free-Flight Trajectory with a Booster that Fires Mid-Flight

Figure 3-11 shows the difference in trajectories between the free-flight trajectory with a boost and the original trajectory to clearly show the discontinuity in the boosted trajectory. A 0.5 G force occurs at 470 seconds for 0.1 seconds in a direction parallel to the velocity of the object. The plot showing the difference in trajectories becoming negative is understood from the perspective that at 470 seconds, the target's velocity is pointing towards the observer, and thus the range of the nonlinear free-flight will be less than the original unaffected flight path.

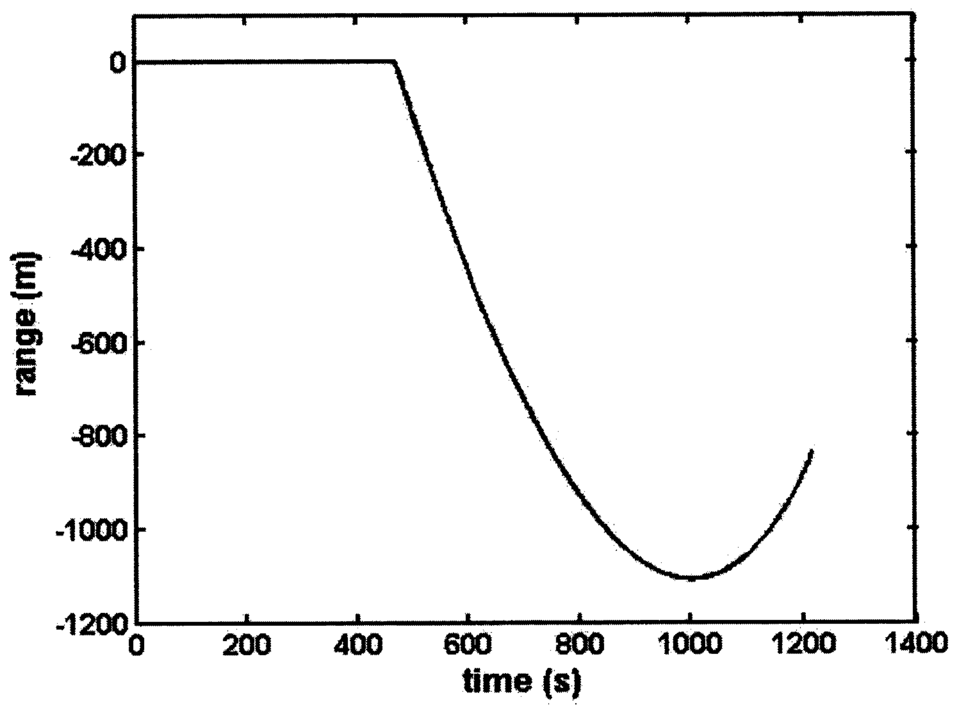
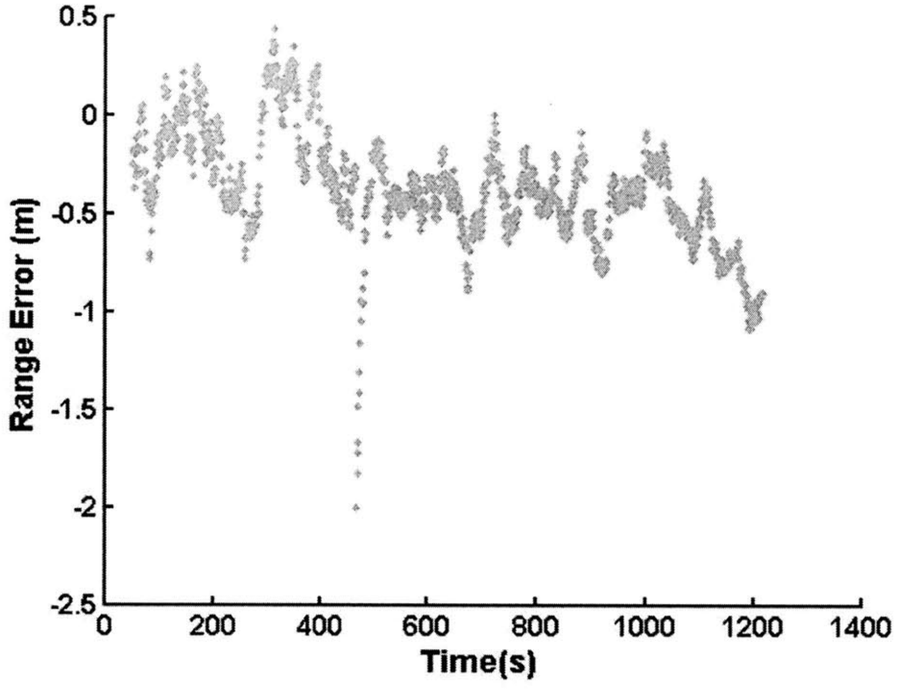
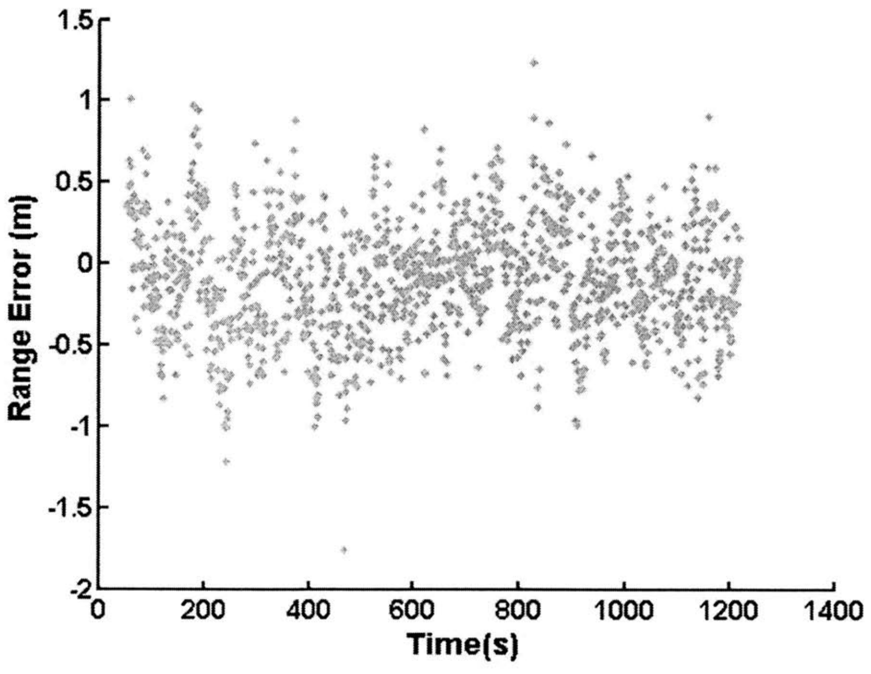


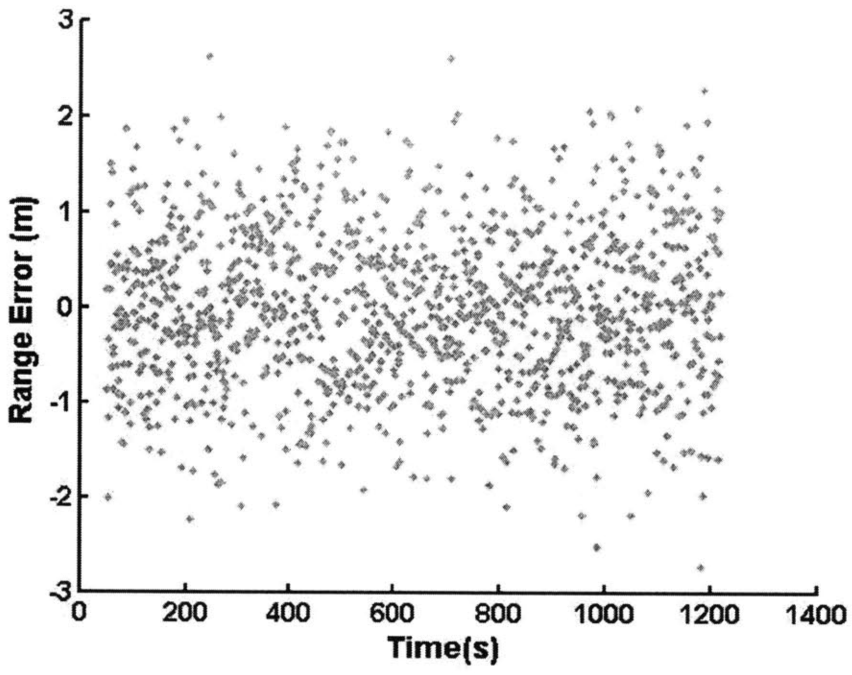
Figure 3-11: Difference between trajectory of free-flight target with a boost and trajectory of unaffected free-flight target


Range Error (m)
Time(s)
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
200
400
600
800
1000
1200
1400

(a)


Range Error (m)
Time(s)
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
200
400
600
800
1000
1200
1400

(b)



(c)

Figure 3-12: Range tracking residual error for system noise values at (a) 0.05 (b) 0.5 and (c) 5 with a perturbed free-flight path

Similar characteristics are observed in the perturbed flight path as that compared to the unaffected flight path: oscillations at lower system noise values and increasing standard deviation of residual error at higher system noise values. The difference between these plots and the unaffected range track residual error plots is the perturbation that is apparent in the Figure 3-12 (a). This perturbation is due to the sudden change in acceleration of the actual flight path, and the tracking algorithm's response time to the sudden change. As system noise increases, either the system is able to recover more quickly from the perturbation or the perturbation no longer falls outside of the expected range of interest of the ladar, because the deviation allowed is larger.

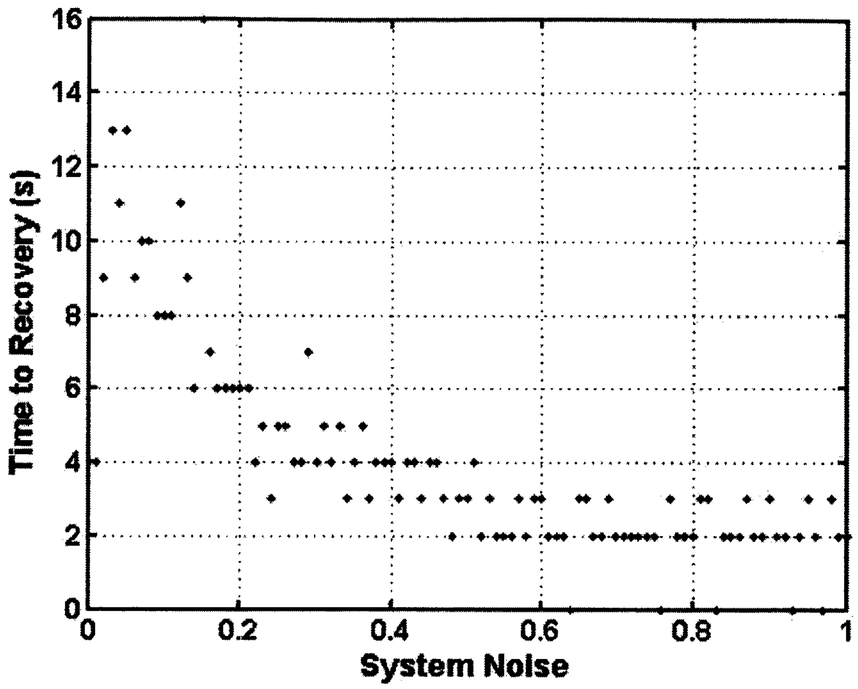
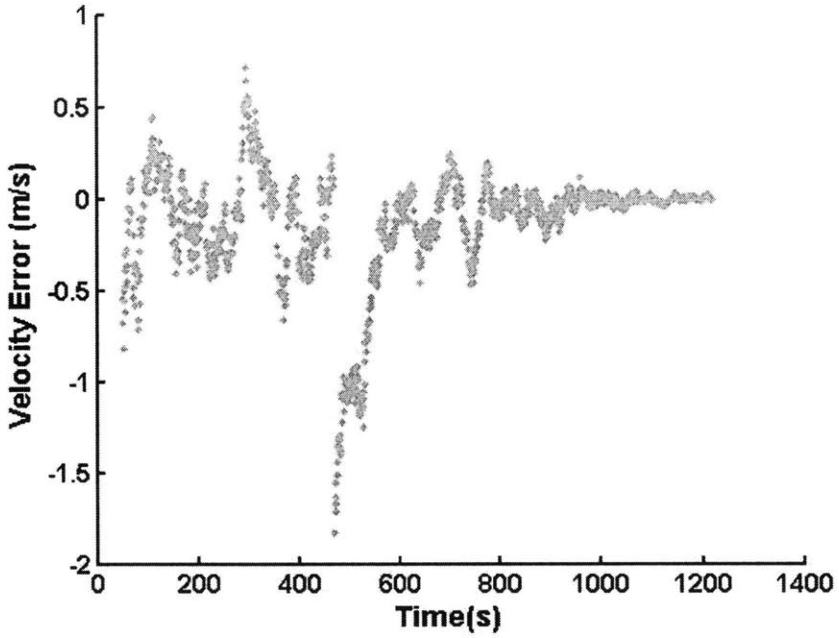
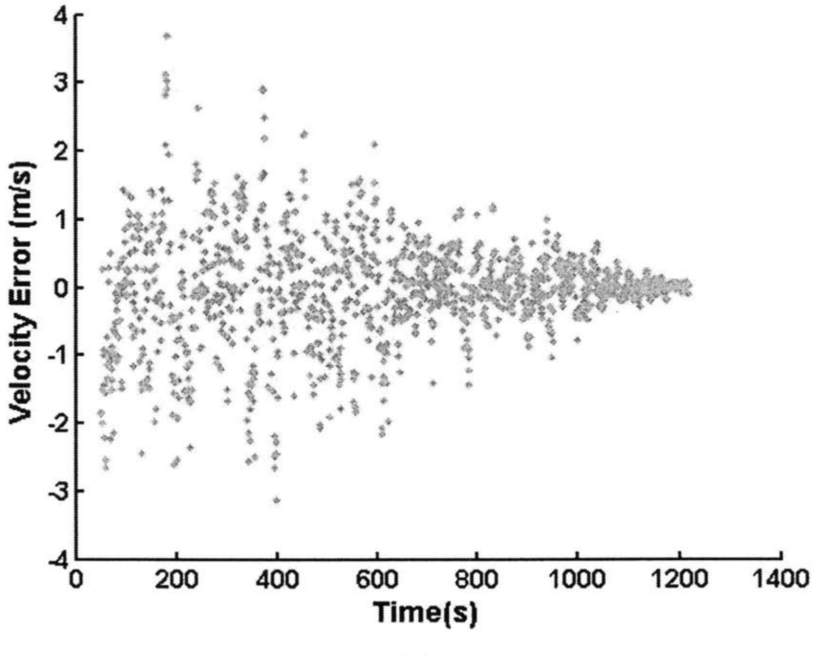


Figure 3-13: Response time for perturbation recovery in range tracking

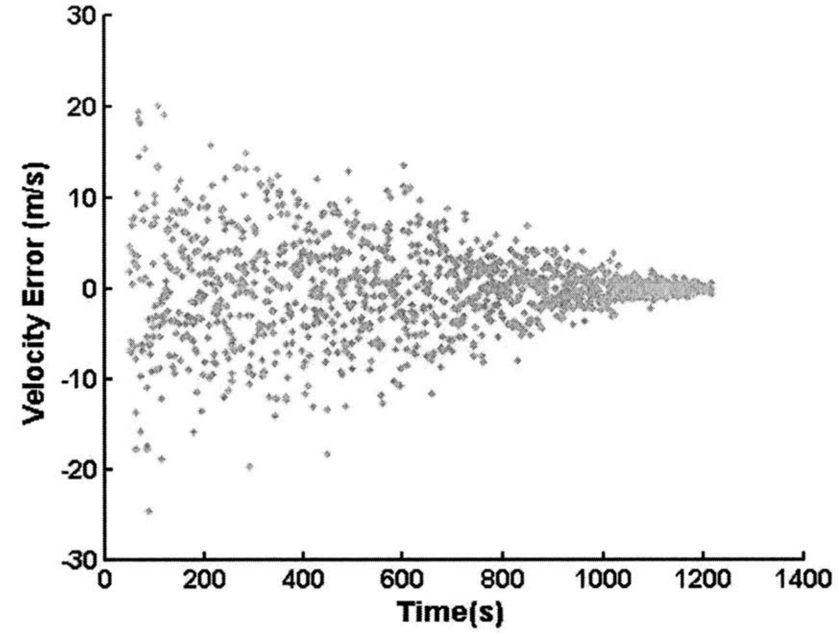
Figure 3-13 shows the time the Kalman filter model takes to recover from a perturbation as compared to the system noise parameter. We defined the signal to have recovered when it returns to within one standard deviation of the mean. We can conclude that with the perturbation given, the tracker will be able to recover the target's range within 2-4 time steps if the system noise is greater than 0.5.



(a)



(b)



(c)

Figure 3-14: Velocity tracking residual error for system noise values at (a) 0.05 (b) 0.5 and (c) 5 with a perturbed free-flight path

Figure 3-14 shows the velocity track residual errors of the three system noise models again. Again, the decreasing standard deviation is apparent in each of these models. Similarly, the perturbation is easily seen when the system noise is 0.05, but it quickly disappears into the rest of the noise when the system noise is 0.5 and 5. Figure 3-15 shows the response time for recovery for the velocity tracker. Again, we have defined the response time to be the time from the perturbation to when the signal returns to within one standard deviation of the mean. From what we can see here, the velocity track takes much longer to recover. If the system noise was greater than 0.5, the time to recovery would be between 0-14 seconds.

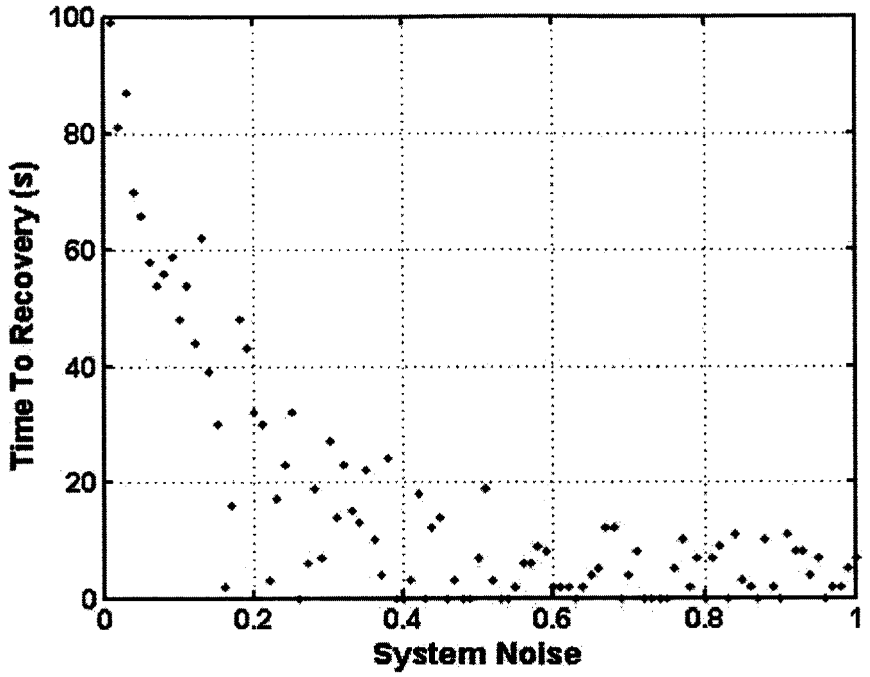


Figure 3-15: Response time for perturbation recovery in velocity tracking

3.3.3 Analysis of Tumbling Free-Flight Trajectory

Figure 3-16 shows the difference in trajectories between the tumbling free-flight trajectory and the original trajectory to show clearly the rotations. The tracking algorithm is tracking a point on the target 3 meters away from the center of mass, and tumbling at a rate of 3 degrees / second, or 120 seconds per cycle. The target is tumbling in the direction of the observer.

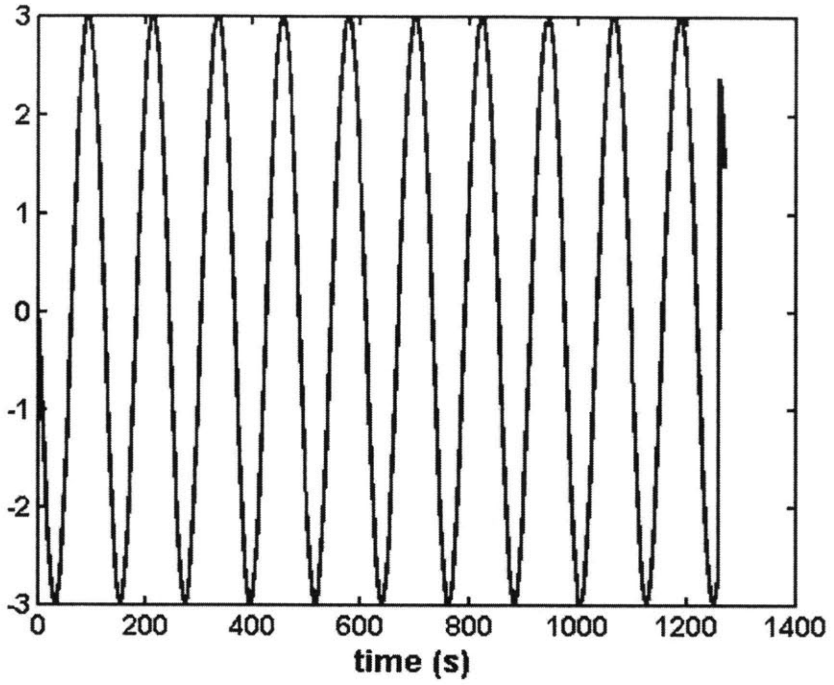
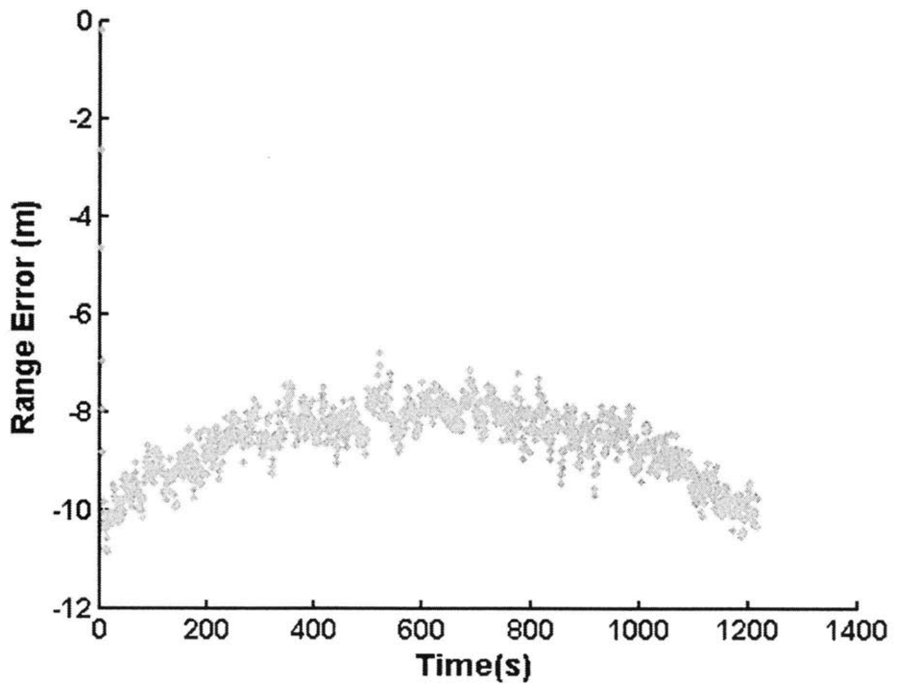
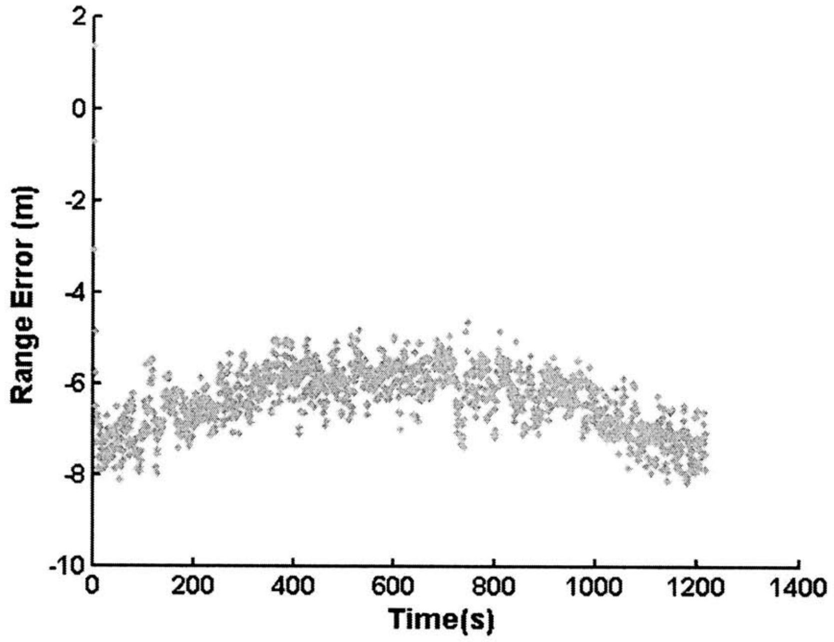


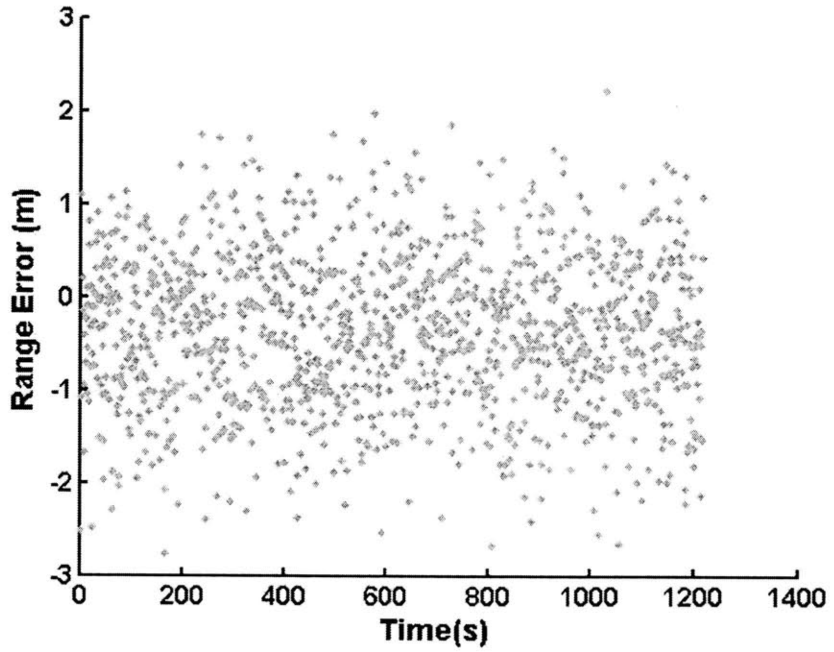
Figure 3-16: Difference between trajectory of tumbling free-flight target and trajectory of unaffected free-flight target



(a)

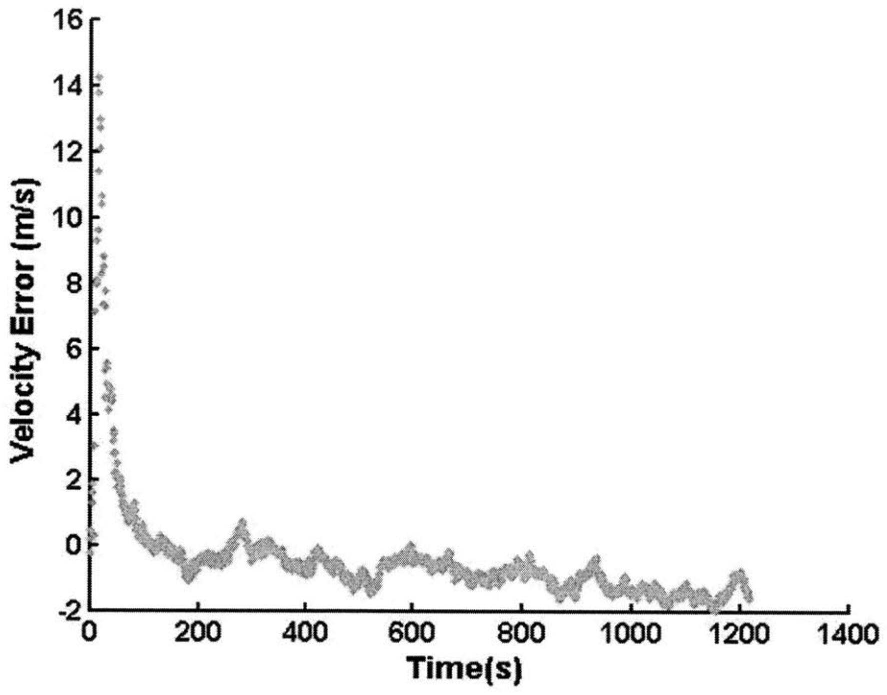


(b)



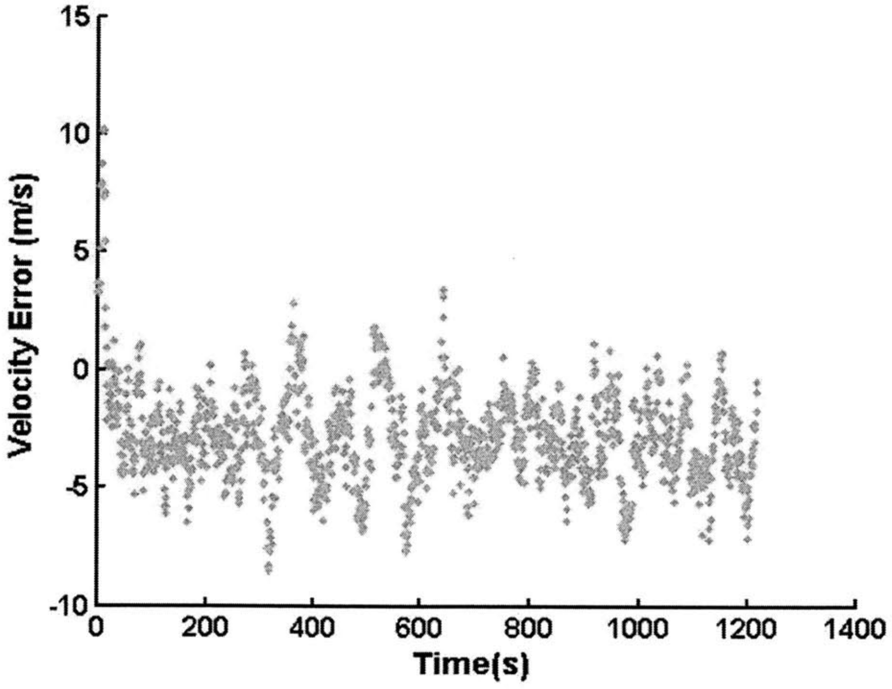
(c)

Figure 3-17: Range tracking residual error for system noise values at (a) 0.05 (b) 0.5 and (c) 5 with a tumbling free-flight path



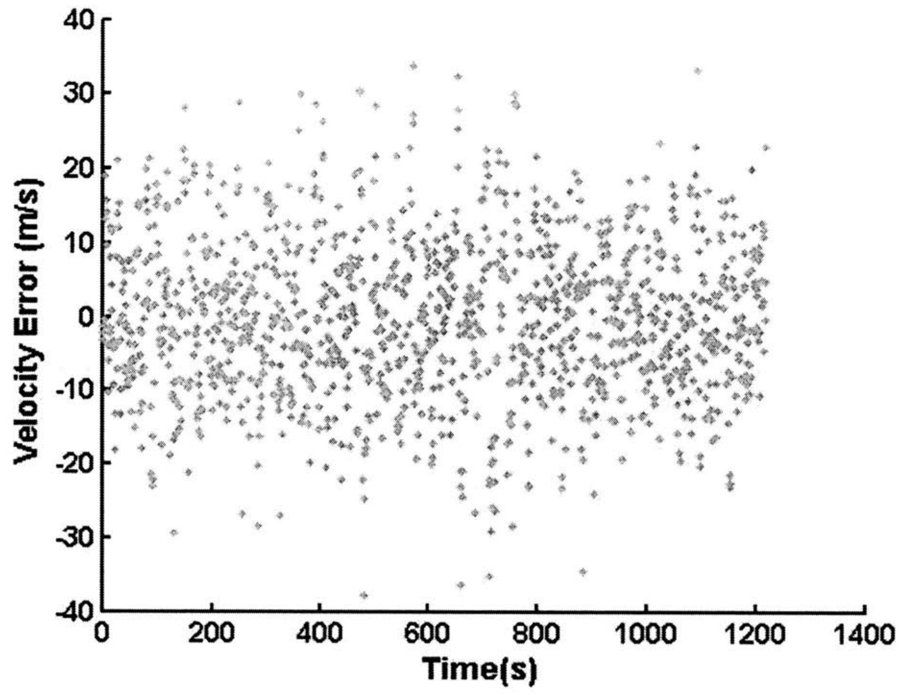
16
14
12
10
8
6
4
2
0
-2
0
200
400
600
800
1000
1200
1400
Velocity Error (m/s)
Time(s)

(a)



15
10
5
0
-5
-10
0
200
400
600
800
1000
1200
1400
Velocity Error (m/s)
Time(s)

(b)



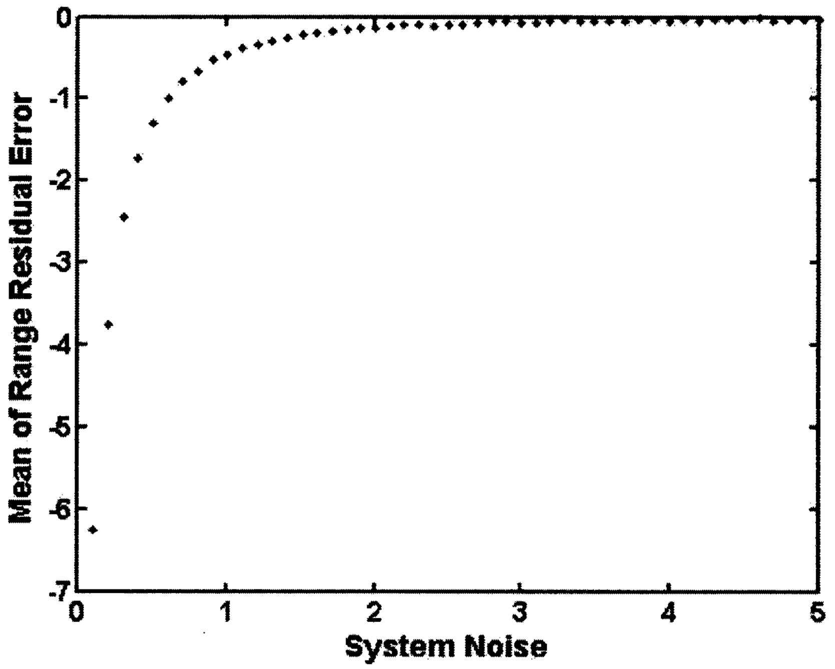
(c)

Figure 3-18: Velocity tracking residual error for system noise values, S_{noise}, at (a) 0.05 (b) 0.5 and (c) 5 with a tumbling free-flight path

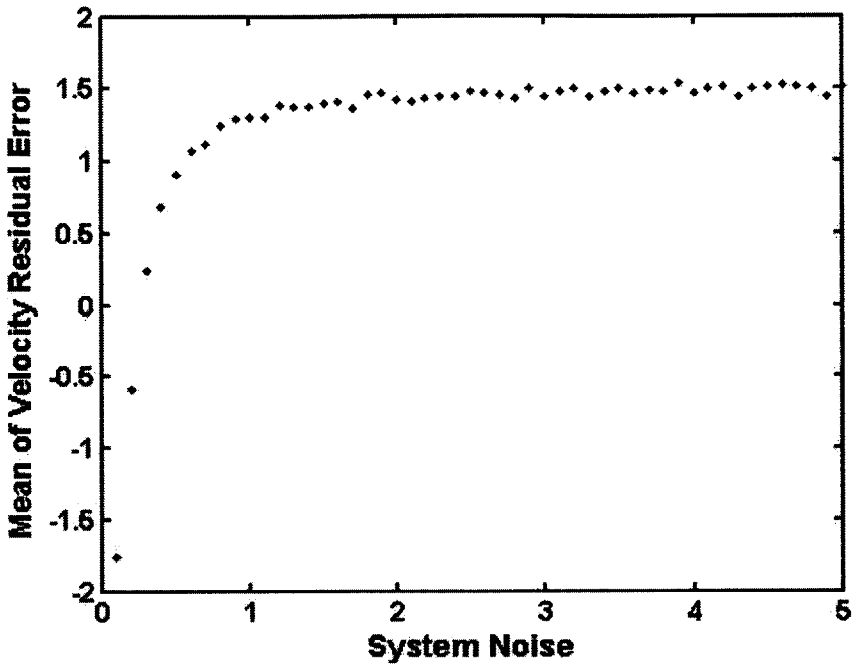
For the tumbling target, it is immediately noted that the residual error is more biased when the system noise is small than when the system noise is large. Furthermore, the residual error looks as if there is a parabolic curve that would fit the residual errors, implying that our model is incorrect, or missing some factor.

Within the Kalman filter-tracking algorithm, there is a system model, a model that attempts to match the parameters of the actual target in space. When the system noise is small, the Kalman filter believes that the actual target in space matches the model, defined within its parameters, even better. Thus, because the target is tumbling, there is an extra parameter that was not modeled at all in the tracking algorithms. The Kalman filter believes that the target is following a normal ballistic trajectory, and thus when the target begins following some other trajectory, it tries hard to match the information it receives with the model it believes is true. As the system noise increases, the Kalman filter becomes more lenient and allows for more errors. As it allows for more errors, the tumbling soon becomes just a part of the residual errors.

It is interesting to note that although the Kalman filter system model does not allow for tracking of the tumbling trajectories at low system noise, it does allow tracking for the trajectory with a boost. This observation can be explained in that the tumbling trajectory is constantly tumbling, and hence constantly deviating from the system model. In the case of the perturbed trajectory however, the deviation only occurs for 0.1 seconds, and the rest of the trajectory is determined by the same influences as the system model defined in the Kalman filter.



(a)



(b)

Figure 3-19: Value of system noise parameter compared to the mean of the (a) range residual error and (b) velocity residual error.

Figure 3-19 shows the relationship between system noise and mean of the range and velocity residual error for the tumbling free-flight path. Again, the velocity residual error is not exactly zero mean. This could be due to the model differing from the actual trajectory. Another possibility is that the method of measuring velocity is incorrect and causing a shift to occur. As one can see, in order to accurately track range and velocity of a target tumbling at 3 degrees / second, 3 meters from its center of mass, one needs to choose a system noise parameter of $S_{noise} \geq 1$.

In summary, we find that $S_{noise} = 0.5$ for free-flight trajectories with or without a boost and that $S_{noise} = 1$ for tumbling trajectories.

4. Tracking Experiments

We implement the tracking algorithm on the actual hardware using the optimal parameters derived in Section 3. The difficulty in testing a hardware implementation of a tracking system is that the range to target varies from several 1000's of kilometers down to 0 kilometers and the velocities can be -7.5 to +7.5 km/s for low earth orbit satellites. These experimental conditions cannot be easily done in a laboratory. Testing a ladar on actual targets is expensive since this requires laser safety clearance and access to highly subscribed tracking telescope facilities. To provide a realistic environment to the photon-counting ladar, we built a digitally programmable delay line to simulate the ranges of interest in hardware. We call this hardware our range simulator. This chapter will discuss this experimental setup, the timing considerations of the setup, and the results from the tracking simulations performed.

4.1 High-Level Design and Experimental Setup

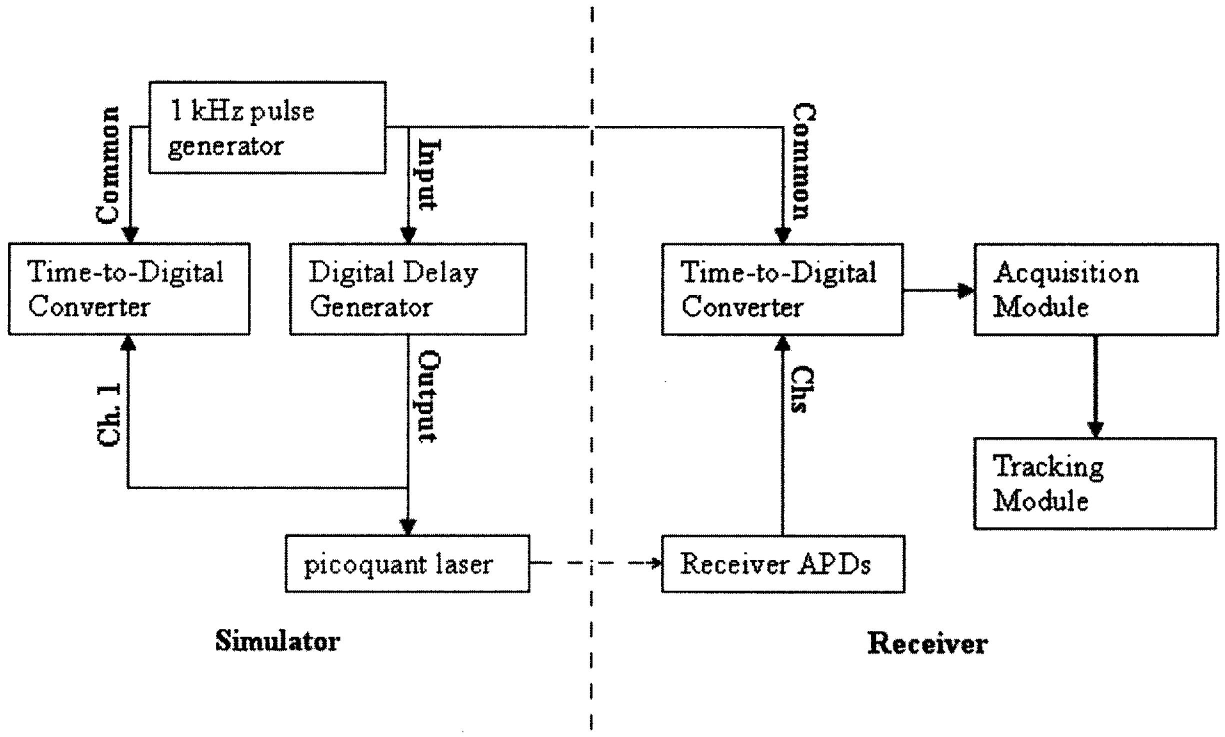


Figure 4-1: Layout of Tracking Algorithm Hardware and Experimental Setup- using a Time-to-Digital Converter (Acqiris TDC890), Digital Delay Generator (Greenfield Technology GFT1208), Picoquant laser (1064 nm), Receiver APDs (PerkinElmer SPCM-AQR-12 Si APD) and a 1 kHz pulse generator (Symmetricom XLi, cPI-SYNCCLOCK326U-Univ, Brandywine)

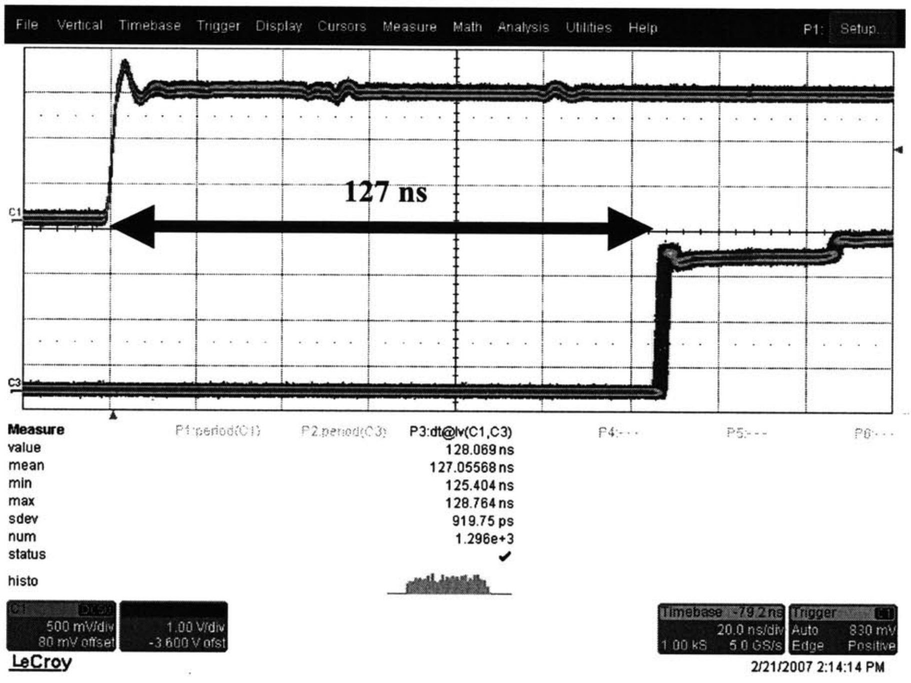
Figure 4-1 shows the general layout of the hardware implementation of photon-counting ladar's tracking system. There are two major components to the system: a hardware simulator, and the photon-counting ladar receiver. The actual ladar hardware would consist of a laser that fired a series of pulses towards the target. Each pulse would theoretically hit the target and bounce back towards the receiver. Because we do not have an actual target to track in space, the experimental hardware includes a simulator, which delays pulses for a set amount of time, and then fires them again afterwards. As can be seen from the Figure 4-1, a 1 kHz pulse generator produces a pulse every millisecond. The Digital Delay Generator (DDG) delays the given pulse for some amount of time and then triggers the picoquant laser. These three steps simulate the laser firing out towards space, hitting the target and bouncing back. The delay time set in the DDG is the same amount of time as the time it would take for a photon to travel from the transmitter to the target and back to the receiver.

After the laser pulse from the picoquant laser hits the receiver, the receiver time stamps the arrival time of the pulse. After a set number of pulses have been fired, the information from the receiver APD is passed along to the acquisition module. Because we are planning on working at low receive power levels, the purpose of the acquisition module is to determine the range of the target and how fast the target is moving. Once the module estimates the location of the target, it passes the range information into the Kalman filter-tracking module. The following sections will walk through each of the components of the experimental setup.

4.2 Range Simulator Design and Specifications

The first major component of the experimental setup is the simulation of the target. This component exists to recreate the delay of laser pulses as they fire from the transmitter, reflect from the target, and reach the receiver. The main hardware of the simulator is the DDG, which receives pulses, delays them for a period of time, and then fires them again. Three particular characteristics of the DDG were examined: the time jitter of the output of the DDG compared to the delay size, the propagation delay through the system, and the update time of the delay.

In order to determine the time jitter of the output from the Greenfield Technology Model GFT1208 digital delay generator, a 1-kHz pulse generator was connected to the input of the digital delay generator. Both the input and the output of the DDG were connected to an oscilloscope that was triggered by the signal to the input of the DDG. The timing jitter between the two traces were measured on a 500 MHz bandwidth oscilloscope. Figure 4-2 shows the scope trace of the input and output of the DDG when its delay is set to 100 ns. As one can see, the rising edge of the input pulse is always at the same point (because we trigger the scope from this trace), and the output pulse has a slight spread. The measured delay of 127 ns indicates that there is an additional 27 ns of propagation delay through the cables to the oscilloscope.



```
Measure   P1:period(C1)   P2:period(C3)   P3:dt@lv(C1,C3)      P4:---   P5:---   P6:---
value                                         128.069 ns
mean                                       127.05568 ns
min                                           125.404 ns
max                                           128.764 ns
sdev                                           919.75 ps
num                                             1.296e+3
status                                                 ✓
histo

C1 DC50               1.00 V/div          Timebase -79.2 ns   Trigger        C1
500 mV/div            -3.600 V ofst                 20.0 ns/div   Auto       830 mV
80 mV offset                              1.00 kS    5.0 GS/s    Edge     Positive
                                                          2/21/2007 2:14:14 PM
```

Figure 4-2: Oscilloscope screen shot of jitter from digital delay generator. The input pulse is on the left, and the output pulse is on the right. Statistics of the delay are underneath the oscilloscope plot.

Table 4-1: Comparison of delay jitter output

DDG Delay Value	Mean	Standard deviation
100 ns	0.20077 us	931.76 ps
1 us	1.10078 us	935.11 ps
10 us	10.10078 us	928.54 ps
100 us	100.10042 us	936.7 ps

From the measurements made, the jitter did not change as the delay size changed. One could safely assume then that in the range of delays being tested, the jitter of the digital delay generator will be constant. Table 4-1 shows a list of means and standard deviations for the delay generator output given a digital delay size.

The delay propagation through the system can also be seen in Table 4-1, based on the mean of the outputs. As one can see, there is approximately a 100.78 ns propagation time, meaning that it takes about 100 ns for the delay generator to receive a pulse and calculate the delay for that pulse, and then fire it again after the delay period. This means that distances below 15 m cannot be simulated at all with the given DDG.

Finally, the delay update time was measured again by setting the delay time and measuring the amount of time the computer took to compute the command of setting the delay. Table 4-2 shows the relationship between the delay size and the delay write time. As one can see, the delay write time is equivalent to the delay size until the delay size goes below 1 ms. The minimum delay write time is 1 ms.

Table 4-2: Delay write time compared to delay size

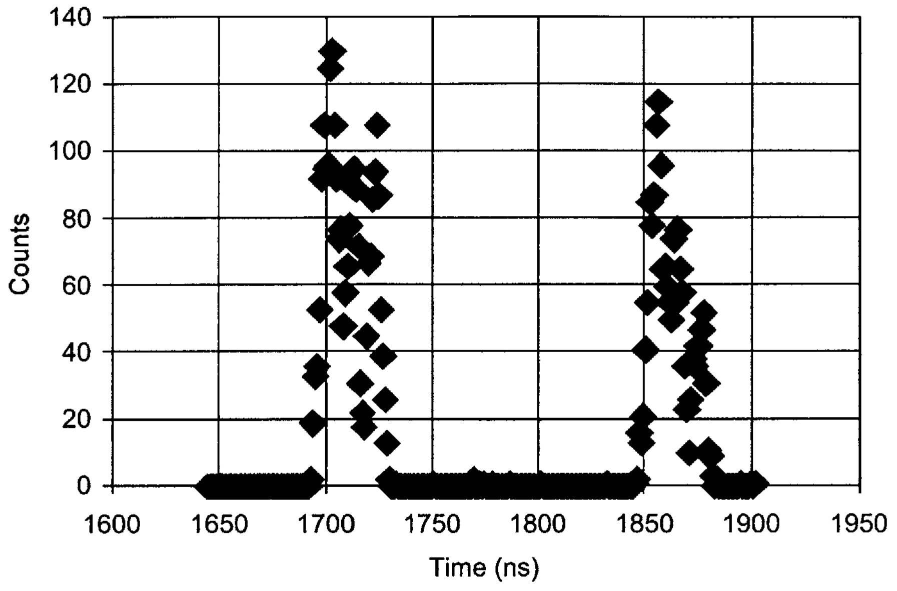
Delay Size	Average Delay Write Time	Standard Deviation of Delay Write Time
100 us	514 us	484 us
200 us	529 ns	482 us
300 us	508 us	483 us
400 us	504 us	487 us
500 us	538 us	503 us
600 us	531 us	558 us
700 us	523 us	490 us
800 us	541 us	628 us
900 us	527 us	584 us
1000 us	1067 us	984 us

In developing this system, the original intent was to have the DDG receive and buffer and fire pulses all at the same time. Unfortunately, the DDG did not have such capabilities. The DDG is unable to buffer the pulses. What this meant is that the DDG would have to receive a pulse, delay that one pulse, and fire it again before it was able to receive another pulse. This implies that multiple pulses could not be in the air at the same time. While the DDG was delaying a pulse, any pulses that arrive at the input channel will be ignored. This limitation restricts the delay size to be at most 1 ms, otherwise we would be ignoring some of pulses that are going at the 1 kHz rate which would throw off the simulation.

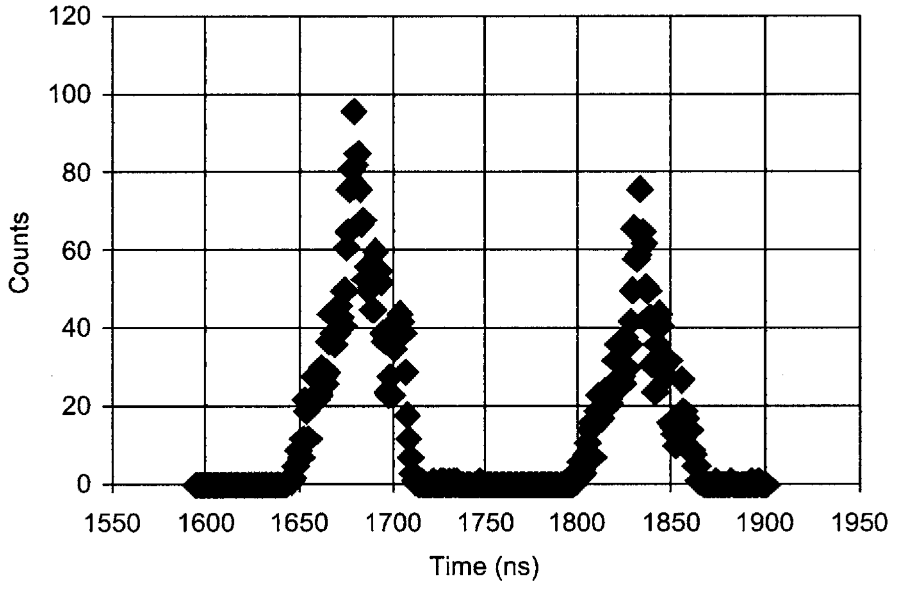
Because of this limitation, modifications had to be made to the simulations to allow for the testing of targets that are more than 1 ms, or 150 km, away from the observer. These modifications are shown in Figure 4-1. Rather than echoing the output of the microchip laser through the digital delay generator, we connect it directly to the 1 kHz heartbeat of the system. This removes the microchip jitter from the measurements, but in essence, still tests the hardware and the real-time tracking algorithms. The conversion from 1 ms to 150 km comes from the fact that if it took a laser pulse 1 ms to leave the transmitter, reflect from the target and hit the receiver, the target would have to be 150 km away, using 3E8 m/s as the rate light travels. Two sets of experiments were developed to get around the 1 ms delay limit. The first set of experiments involves using only the portions of the trajectory that lies within the 150 km of the observer. Because all of the trajectories are targets that are aimed to collide with the observer, the last 20-30 seconds of each trajectory fall within the 150 km range limit.

Figure 4-3 shows the histograms of the delay generator timing for a constant delay over time and for a linear ramp delay over time. In the moving delay shown in Figure 4-3(b), the target is moving at a constant speed of 15 cm/ms away from the observer. The data has been aligned so that it is as if the target is not moving at all. This information was important to derive the bin size for acquisition and tracking. As we can see from the plot, in order to capture all the points in the signal we need a bin size of 50 ns. The width of the first peak in Figure 4-3(a) is 30 ns, and the width of the first peak in Figure 4-3(b) is 50 ns.

It is interesting to note that when the delay is constant, the width of the bin size is a little smaller than when the delay is moving. This observation can be understood due to a few reasons. The 30 ns width of the peak for the constant delay is due to the differential delays in the receiver cable lengths and the jitter that is inherent in the system. The increase to 50 ns in the moving delay is due to the fidelity of the simulator, which at times holds the delay information for too long, and misses a delay update every few pulses. Furthermore, a double peak appears due to pickup of some other noise or an echo in the system.



(a)



(b)

Figure 4-3: Histogram of the delay generator timing for a (a) constant delay over 100 pulses and (b) linear ramp delay over time over 100 pulses

4.3 Receiver Design and Specifications

The picoquant laser pulses are aligned to the photon-counting ladar receiver, which consists of a 5x5 grid of single-photon detectors. Each of these detectors is connected to a time counter, which time stamps the photons as they are detected. All the time-stamps from all 25-detectors are combined together into one set of time stamps for each pulse. In other words, a single detector consists of 25 single-photon counting modules. One hundred pulses are combined together as the input to the acquisition module.

The receiver not only collects the pulses from the picoquant, but it registers counts at random times even when the detector is not illuminated. These stray counts are called dark counts. In addition, stray photons from room lights will increase the background count rate. Approximately 20% of the time stamps are dark and background room light counts when the attenuation of the picoquant is set to 0 dB (corresponding to 24 hits/pulse). As the attenuation of the laser increases, the received power decreases, and the signal-to-noise ratio drops.

Several measurements were made between the picoquant and the receiver to characterize the propagation delay spread and jitter between the transmitter and the receiver. These measurements were made for delays ranging from 100 us to 900 us. The number of hits (including dark counts) and the number of valid hits (total hits minus dark counts) are also noted below in Table 4-3. The average delay and the standard deviation of the delays are the statistics of just the valid hits. The standard deviation of the delays are mainly due to the spread of APD cable lengths. Each APD is connected to a different TDC channel. The cables between the APDs and the TDCs are of different lengths, and the entire spread due to the timing of cable lengths was measured to be around 2.4 ns, similar to what is shown in Table 4-3.

Table 4-3: Delay size compared to the delay between the triggering of the delay generator and the receiver.

Delay Size	# of hits	# of valid hits	Average Delay	Std. Dev. of Delay
100 us	483	338	101.3467 us	2.450815 ns
200 us	499	350	201.3462 us	2.413843 ns
300 us	493	344	301.3455 us	2.515932 ns
400 us	479	336	401.3448 us	2.382767 ns
500 us	488	347	501.3443 us	2.483941 ns
600 us	499	349	601.3433 us	2.503105 ns
700 us	490	351	701.3429 us	2.450327 ns
800 us	477	335	801.3422 us	2.520336 ns
900 us	517	350	901.3416 us	2.582794 ns

From these results, one can say that the delay from the trigger of the delay generator to the receiver APDs is about 1.344 us with a standard deviation of 2.478 ns. This delay of 1.344 us comes from two locations: (1) pulse width of the delay generator output and (2) propagation delay of the cables. Because the firing of the laser triggers off of the falling edge, the propagation width of the digital delay generator is factored into this delay. The propagation width of the digital delay generator is 1 us. The remaining 344 ns comes from the propagation delay of the cables and the propagation delay of the digital delay generator itself.

The TDC cards collect 100 pulses before sending the 32-bit time stamps of the 100 pulses to the acquisition module that runs on the server CPU. Thus, any acquisition / tracking methods or algorithms that need to be taken care of, need to be taken care of before another 100 pulses are sent. Because the pulse period of the laser is 1 ms, the computation time for acquisition and tracking must update within less than 100 milliseconds.

4.4 Acquiring the Target

Once the time stamps from 100 pulses are captured and sent from the receiver APDs, the acquisition module uses all 100 pulses to determine a range of target along line of sight and the magnitude of its velocity. One of the goals of the photon-counting ladar program is to be able to track a target with less than 1 photoelectron per pulse. As stated before, the ability to work at a sub-photon level allows one to get more information about a target at a greater distance because the power of the laser decreases by a factor of R^{-4}, where R is distance from the observer to the target.

In order to work at the sub-photon level, the tracking algorithm takes the time stamps from 100 pulses and analyzes all 100 pulses for range information. Because we're at a sub-photon level, it's possible that very few photons actually arrive for each pulse, and thus by using time stamps from several pulses, we may be able to strengthen the signal-to-noise ratio. The motion of the target will manifest in the data as a time delay from pulse to pulse. The Doppler shift or velocity along the line of sight is recovered by numerically delaying each received pulse by an amount determined by the estimated target velocity.

An acquisition module, written in C++, was developed to take 100 pulses, and determine the appropriate time shift, to find the actual range of the target.

```
for velocity = -5 km/s to 5 km/s
      for all 100 time stamps
            add time stamps + pulse # * shift to histogram
            find peak in histogram
      end
end
find highest peak of all peaks
using the histogram with the highest peak, return the time stamp
      containing the highest peak
```

Figure 4-4: Pseudocode of Acquisition module

As can be seen from Figure 4-4, the algorithm tests a number of possible velocities, determines the highest count of time stamps for each velocity, and then chooses the velocity with the highest peak. The velocity that gives rise to the highest peak corresponds to the correct velocity of the target.

In the actual development of this section of code, the timing of the system was a major consideration. The input to the Kalman filter is updated every 100 ms hence the acquisiton module code must efficiently compute the pseudocode in Figure 4-4 in less than 100 ms. There were three main decisions that needed to be made concerning the coding aspect of the target acquisition: how to design the storage array for the histograms, how to take care of garbage collection afterwards, and how to efficiently find the peak of the histogram.

The question of storage array for the histograms arises due to the fact that the program is working with a few pulses, and a large range of possible values (e.g. 50 ps bins with 1 ms span = $2 * 10^7$ bins). There were some considerations made to use a hash table and store time stamps as the keys, and the frequency a particular time stamp occurred as the values. A number of different methods were tried including hash tables, maps, and regular arrays. Nothing worked fast enough to check all of the possible velocity values except for a large storage array allocating a separate cell for each possible time stamp. Thus, if the program were analyzing 1 ms worth of data, with time stamps at 1 ns, then the program would use a 1,000,000-cell array. Because it's a regular array, the program would have instant access to each time bin, and items could be stored with O(1) time.

The use of such a large array led to the immediate problem of garbage collection. The program needs to clear the histogram between each velocity test. Running through the entire array and setting all the cells to zero, used more than the 100 ms that was allotted in order to maintain real-time acquisition. In the end, it was decided to keep a cleanup array on the side. This array would keep track of all the time stamps that have been used in the histogram, and when the histogram needed to be cleaned up, the program would run down this list of indices and set only these cells to zero. All of the time stamps received were placed into the cleanup array, even if there were repeats. The decision to ignore the repeats was made due to the idea that there would not be that many pulses, that the cleaning of repeated pulses would take a substantial amount of time. In fact, it would probably take more time to search through the cleanup array to determine if a time stamp already existed, than it would to incorporate it into the array again. In test runs, with about 4 time stamps per pulse, 100 pulses, the time to clean the array dropped by three orders of magnitude, from 300 ms to .3 ms. When running with real data from the receiver APDs, the acquisition module received at most 1200 hits per 100 pulses, and was able to search through 88 possible shifts in 70 ms on average. Because the shifts should only be between -33.33 ns / ms and 33.33 ns / ms, we decided to test shifts from -33 ns/ms to 33 ns/ms, incrementing by 0.75 ns.

The last area that required some optimization was the portion of code that determined the peak of the histogram. Originally, the program would store all the time stamps into the histogram first, and then determine the maximum. In order to avoid going through the entire array again and comparing values, the program checks the maximum as values are inputted into the histogram. Thus, whenever the program increases the frequency of a bin in the histogram, it checks to see if the size of that bin is bigger than the maximum so far. In this manner, the search for the maximum peak occurs as the system is inputting values into the array, making the search happen in O(n) time, where n is the number of values being stored.

4.5 Target Tracking

Once the acquisition module determines a range for the start of the 100 pulse sequence, as well as a velocity of target along line of sight, it begins to feed these values into the Kalman filter. In order to ensure that the results from the target tracking are not due to the Kalman filter implementation, we also store the ranges from the 100 pulse sequences. After the entire trajectory has been analyzed, we can use the range measurements and run them through the MATLAB model, and compare the results from the MATLAB model and the actual Kalman filter implementation. In this manner, we are able to determine how accurate the Kalman filter implementation is, and also be able to test whether or not the Kalman filter can truly track a target in real-time.

4.5 Timing Requirements

One of the key components to be able to compare accurately the Kalman filter results to truth data is the timing of the entire system. It is imperative to be able to follow a pulse through the entire system, from when it enters the digital delay generator to when it hits the receiver, to when it gets converted to a range measurement which then feeds into the Kalman filter and then becomes a prediction. Only in this manner are we going to be able to compare the predicted track to the actual track and determine how accurate our system is.

There are two clock counters on the both the simulator and the receiver. There is an IRIG time card and the time-to-digital converter (TDC). The IRIG time card returns a time stamp indicating the number of hours, minutes, seconds, milliseconds and microseconds, has gone by since the start of the year. The timing card is synchronized to a GPS time signal. The time-to-digital converter measures the time between any two pulses on two separate channels, with a resolution of 50 picoseconds. By setting the IRIG time card to be the baseline of the time-to-digital converter, we are able to obtain a resolution of 50 picoseconds for all of our measurements. To be able to compare the times between the simulator and the receiver both of the systems need to be synced to the same clock. Both of the TDCs are connected to the same 1 kHz pulse and moreover the IRIG time cards are also connected to the same clock. Thus, both the simulator and the receiver are clocked together and are completely synchronized.

Although we may be able to accurately time stamp each of the values before the digital delay generator, after the digital delay generator and from the receiver APDs, we still need to be able to match up which pulse went into the digital delay generator with which pulse came out from the digital delay generator and which pulse hit the receiver. Much of this information is found by measuring the average delay of the system and the jitter of the delay. Unfortunately, because the delay of the digital delay generator is changing, we are unable to use this method to match pulses entering the digital delay generator with the pulses exiting the digital delay generator. However, because the input and output signals of the digital delay generator are clean and can be directly plugged into a TDC, there should only be one pulse per channel per ms, and thus we can match the first pulse that enters with the first pulse that exits the digital delay generator, etc.

Matching the pulses that are fired from the digital delay generator to the pulses that hit the receiver is a little bit harder. In this situation, it is necessary to compare the time stamps of the digital delay generator and the time stamps from the receiver in order to determine what the average delay time is. This delay is constant because there is no variable in between the picoquant laser and the receivers and as seen in the measurements made in previous sections.

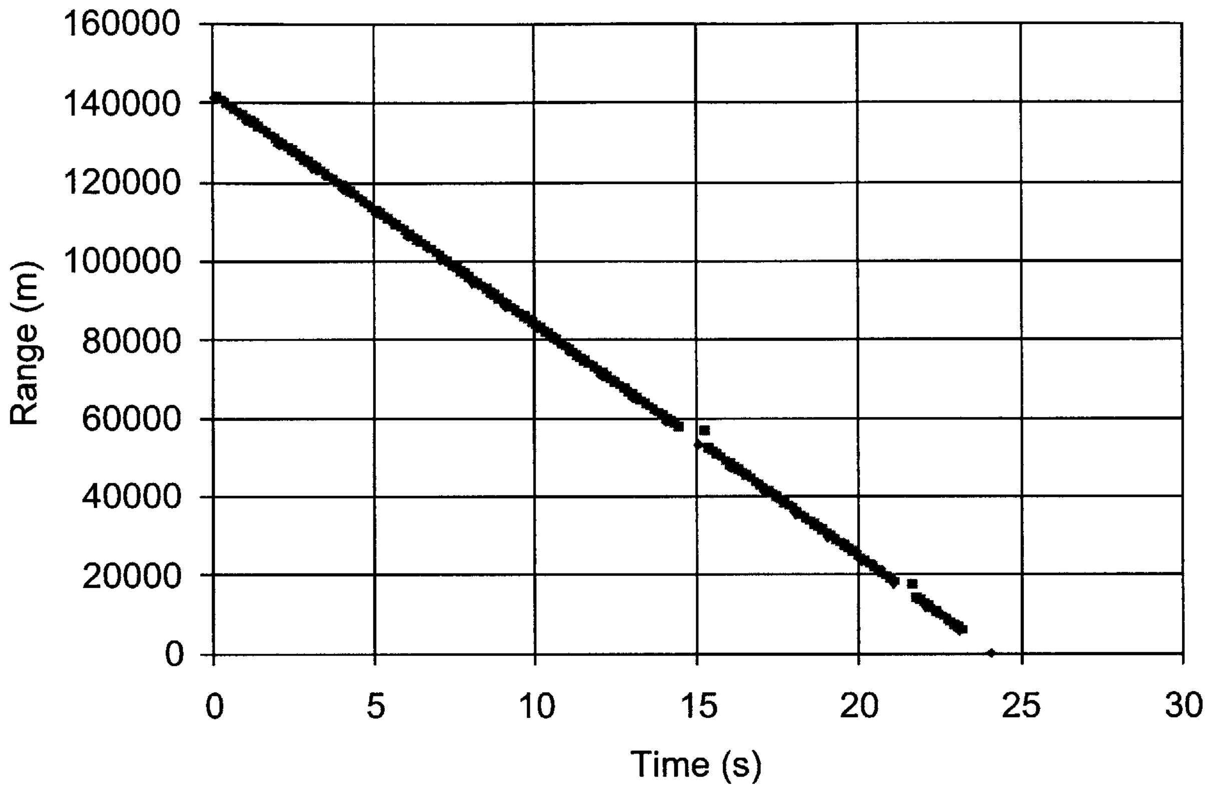


Figure 4-5: Plot of calculated range from both simulator (blue) and receiver (red).

Figure 4-5 shows the plot of the calculated range from both the simulator and the receiver at different time steps. As one can see, the receiver and acquisition module are able to accurately measure the location of the target. There are one or two errors due to the double peak that occurs in the measurement data.

4.7 Experimental Results

Once the delay and the jitter was characterized from the experimental setup, the error covariance was recalculated for range and Doppler. In the setup, there is an error covariance of 7.5 m^2 for the range measurement and 7.5 m^2/ s^2 for the Doppler measurement. These values are different than the ones used in the simulations (Table 3-2). These values were determined based upon the resolution of the acquisition module as well as the 50 ns peak width seen in Figure 4-3(b). These two values were set in the measurement model, and we began testing different trajectories on the setup.

4.7.1 Analysis of Unaffected Trajectory

Figure 4-6 shows the unaffected free-flight trajectory that we used in the setup. All future graphs (Figure 4-7 to 4-10) begin from t=0, which is equivalent to t=1235 seconds in this particular graph.

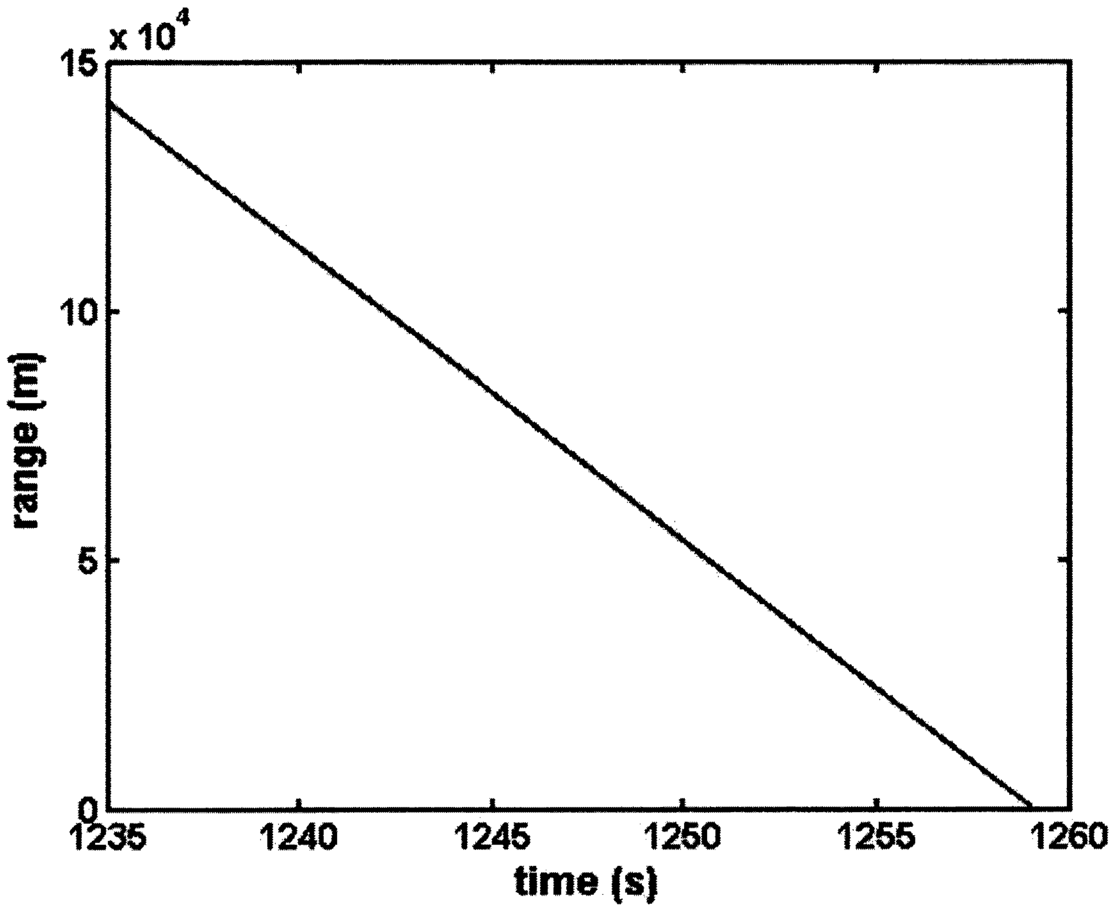


Figure 4-6: A segment of the unaffected free-flight trajectory used in the setup

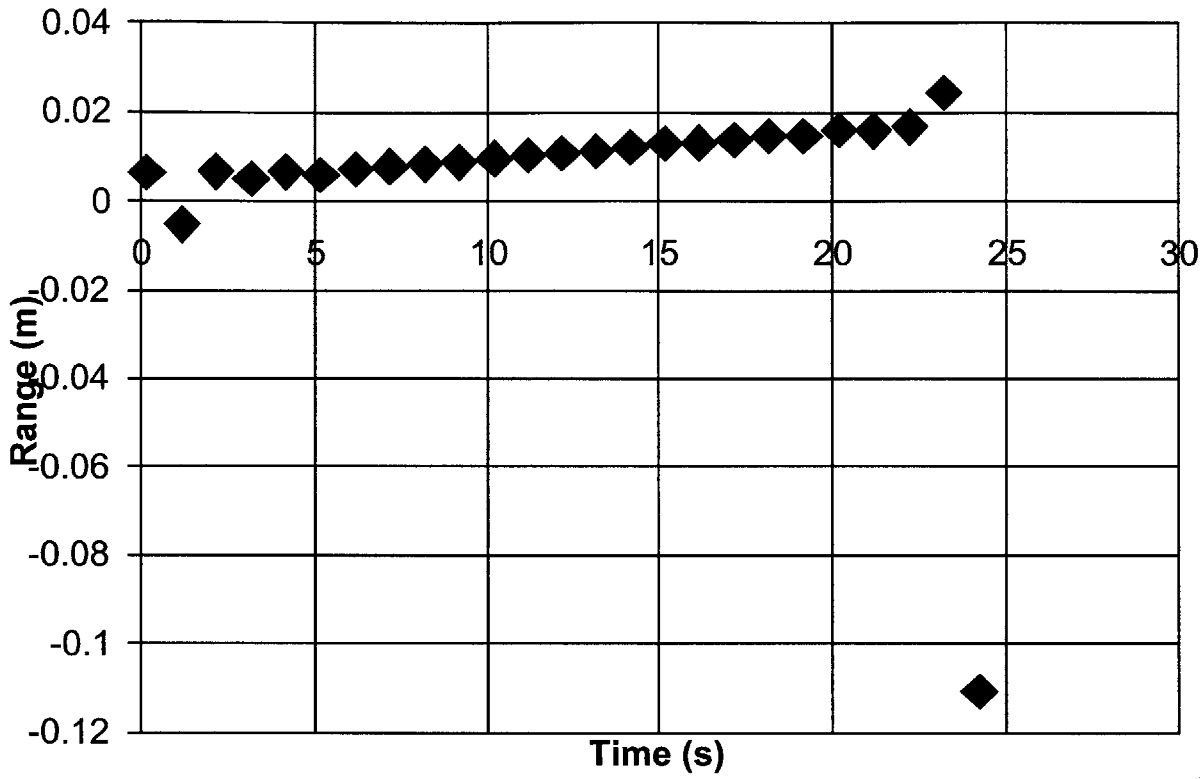


Figure 4-7: Range residual errors between the measurements going into and the predictions from the real-time Kalman filter for the unaffected free-flight trajectory,

i.e. $(z_{k+1} - x_{k|k})\,|_{\text{projected along line of sight}}$

These are the results that we obtain when we calculate the residual error of the output of the Kalman filter to the inputs of the Kalman filter. As we can see from Figure 4-7, the output of the Kalman filter is able to track the target measurements to within 2 cm. The error of the last point is because parts of the last data point included information after the end of the trajectory. In fact, the standard deviation of these points lie within 8.601 mm, and the mean of the residuals is 7.64 mm. S_{noise} was set to be 50 in these simulations. There are a couple of observations to be made with regards to this plot. First, there does appear to be a slightly deterministic slant in the tracking as the target seems to be drifting upwards. We believe that this deterministic shifting could be due to a missing variable in our model. Because the shift looks very deterministic, it would appear that the tracking is consistently propagating the track not perfectly, and thus that the system model that we had put together is not accurately portraying the actual trajectory. In the next few plots, the same shift will appear again, and thus strengthens the argument that there could be a problem with the model.

Secondly, the residual range error here is much more accurate than what the simulations had predicted. The predictions were about 1 meter, whereas in our experiments we show a 2 cm residual range error. This increase in accuracy is due to the acquisition module, which uses 100 pulses to determine the range at a given pulse. Thus, the accuracy of the measurements after the acquisition module is much better than the accuracy we had originally calculated.

There was one major change to the code that had to be developed in order to achieve such accuracy. Originally, we had allowed the receiver to use the Kalman filter whenever it had received 100 ms worth of data points. We were able to get many more data points on the graph. However, because of hardware constraints and hardware noise, the receiver was unable to collect and write out to disk 100ms worth of data in 100 ms. There was some variability in the amount of time necessary to write out the data. This problem was compounded with the fact that the acquisition module took nearly 100 ms to compute the peak and the shift.

This variability from time step to time step, caused errors because the Kalman filter needs to know the time difference from the current time step to the next time step. This time difference is used by the Kalman filter to predict the location of the target at the next time step. If the time difference was incorrect, then the track would begin drifting off because our Kalman filter's model assumes a constant *dt*. We reconfigured the program so that the receiver would only collect 100 ms of data on the second. This corrected our *dt* and made it constant, thus allowing proper predictions to be made from time step to time step.

4.7.2 Analysis of Free-Flight Trajectory with a Booster that Fires Mid-Flight

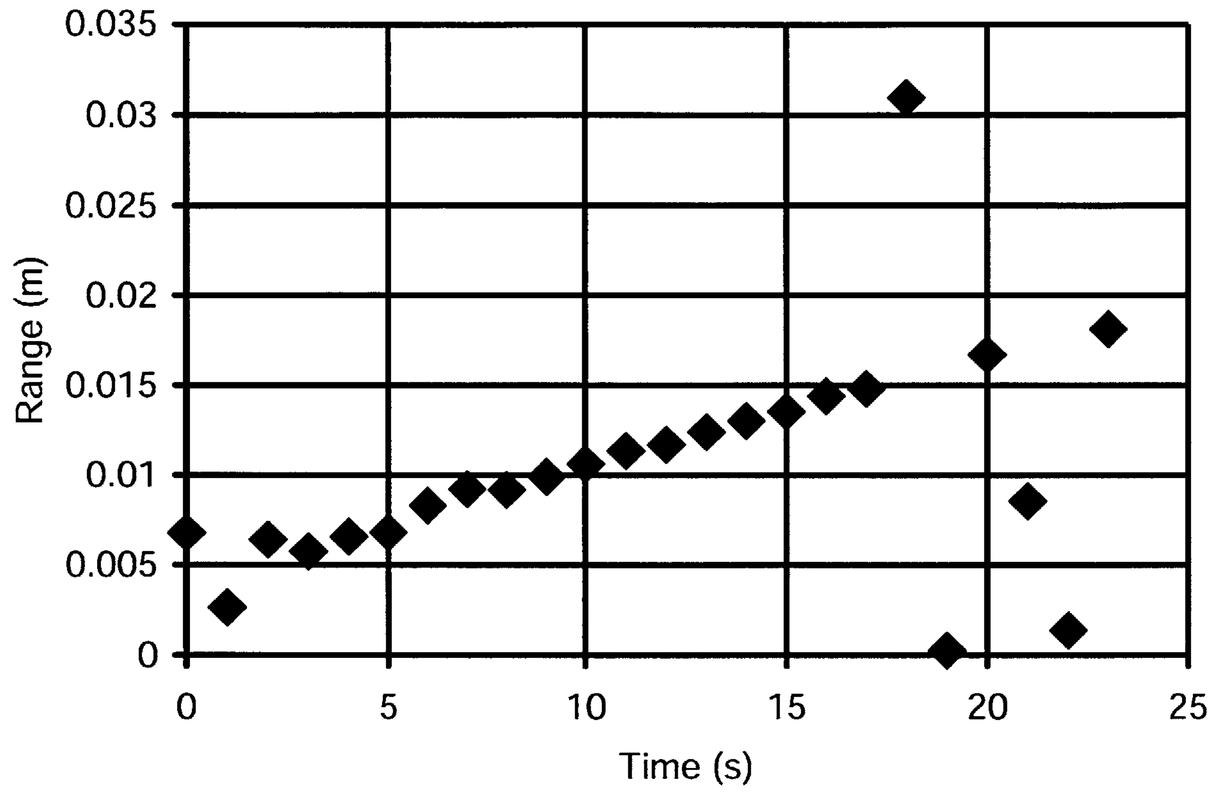


Figure 4-8: Range residual errors of the real-time Kalman filter for the nonlinear free-flight trajectory with optics comparing actual trajectory to predicted trajectory,

i.e. $(z_{k+1} - x_{k|k})\,|_{\textit{projected along line of sight}}$

Figure 4-8 shows the range residual errors of the real-time Kalman filter for the nonlinear free-flight trajectory. A 0.5 G force occurs at 5 seconds for 0.1 seconds in a direction parallel to the velocity of the object. As can be seen, the perturbation does not affect the tracking of the target very much. This can be understood when considering the fact that this nonlinearity causes the target to move at most two meters from its original location. Furthermore, it is possible that the acquisition module's determination of the result based upon 100 pulses mitigates the single-point nonlinearity. The average of the residual error is 1.0378 cm and the standard deviation is 6.304 mm. The deterministic shifting upwards disappears around 18 seconds. This increase in jitter is probably due to the perturbation included into the trajectory. The reason that it is delayed by 13 seconds is probably because the perturbation was initially attributed to system noise. Once the residual between the actual and the expected reached a certain point, the Kalman filter predictions began jumping back in order to acquire the target yet again.

4.7.3 Analysis of Tumbling Free-Flight Trajectory

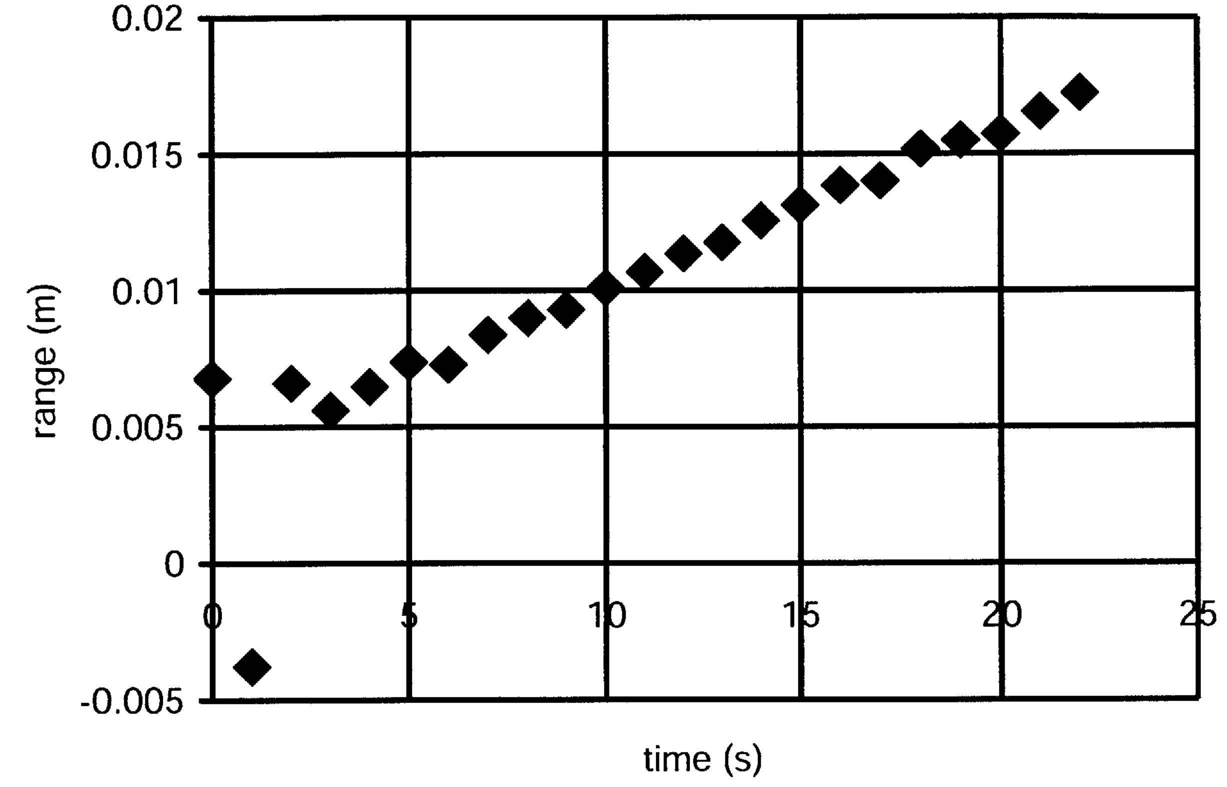


Figure 4-9: Range residual errors of the real-time Kalman filter for the tumbling free-flight trajectory with optics comparing actual trajectory to predicted trajectory,

i.e. $(z_{k+1} - x_{k|k})\,|_{\mathit{projected\ along\ line\ of\ sight}}$

Figure 4-9 shows the range residual errors of the real-time Kalman filter for the tumbling free-flight trajectory. As shown here, again, the tracking algorithm is able to accurately follow the target. Similar to the simulations shown before, the tracking algorithm is tracking a point on the target 3 meters away from the center of mass, and tumbling at a rate of 3 degrees / second, or 120 seconds per cycle. The target is tumbling in the direction of the observer. When we remove the beginning points, the average of the residual error is 1.05 cm, and the standard deviation is 4.7 mm. These values are similar to the unaffected free-flight trajectory. This is suggesting that the Kalman filter's system noise is large enough to accommodate for these values.

4.7.4 Analysis of Laser Strength on Tracking Capability

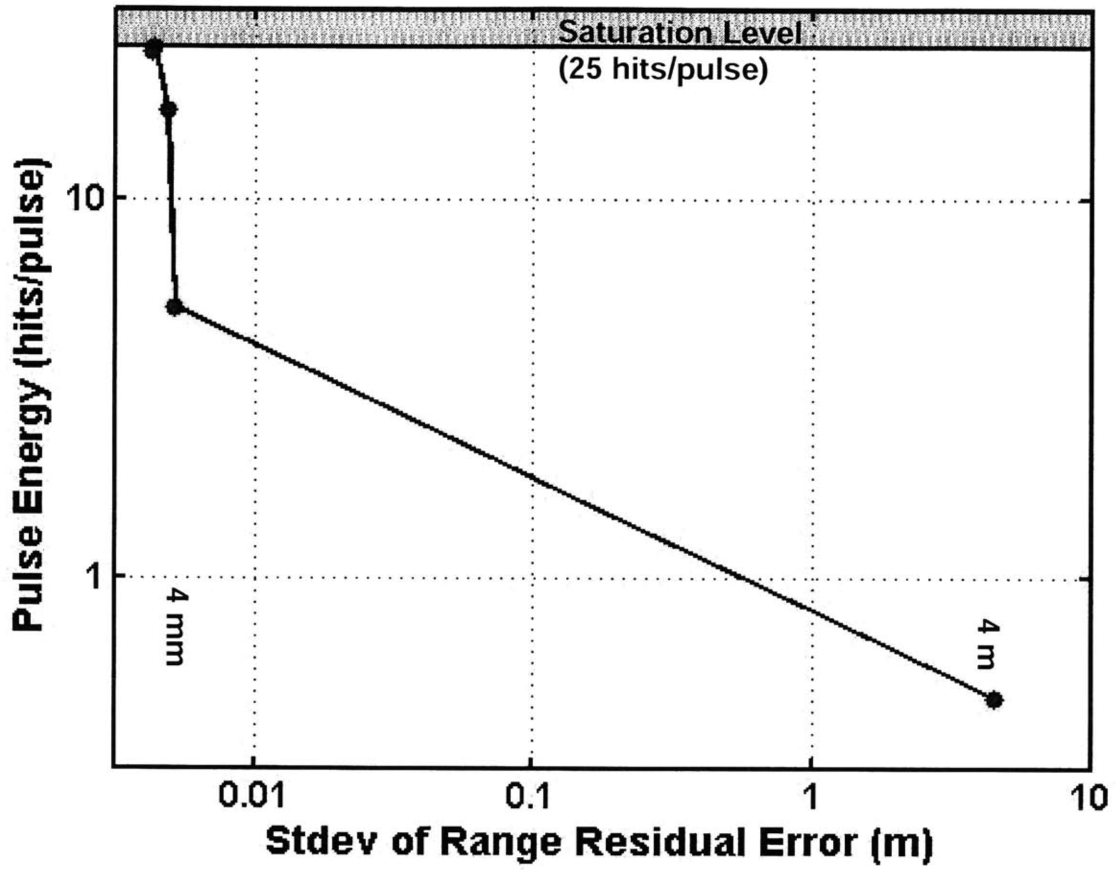


Figure 4-10: Range residual errors of the real-time Kalman filter for the unaffected free-flight trajectory with different energy (hits/pulse) levels

Figure 4-10 shows the results from tracking a simulated target at a number of different energy levels. The measurements were made by placing an attenuator on the laser, and increasing the attenuation by 10 dB for each measurement. Thus, the data point on the far left is at 0 dB, and the data point on the far right was measured at 40 dB. As we can see, the acquisition module is capable of determining the signal from the noise when the receiver receives at least five hits/pulse. One can see that the more energy returning, the better the accuracy. One thing to note is that there is a saturation level on the detectors. We currently have 25 detectors, and thus we expect that there will be a saturation level of 25 hits/pulse. This saturation level explains why we do not see an increase in hits/pulse when we decrease the attenuation past a certain point.

When the signal gets to less than 1 hit/pulse, the signal is no longer detectable above the noise. The acquisition module is not always able to find the signal perfectly, and thus we get errors on the order of a few meters.

We can use the radar equation [9] to transition from pulse energy to distance. The radar equation is

$$n_{pe} = n_{tx} \frac{\sigma}{A} \frac{D^2}{4R^2} \eta , \tag{4-1}$$

where n_{pe} is the number of hits/pulse, n_{tx} is the number of photons fired from the transmitter per pulse, σ is the cross-sectional area of the target, A is the illumination region of the laser at the target, D is the diameter of the mirror in the ladar, R is the distance from the ladar to the target, and η is the system efficiency of both the detectors and the optics. We can assume that the energy per pulse would be around 100 mJ, which converts to $5.35 * 10^{-17}$ photons/pulse when using a 1064 nm laser beam. We assume that the cross-section of the target is roughly 0.01 m^2 at a range of 10^6 meters. At this range, we can assume that the illumination region of the laser (A) is 1 m^2. We also set the diameter of the lens to be 1 meter. The system efficiency is approximately $\eta = 0.02$ (10% detection efficiency and 20% optical throughput). Altogether, we calculated that at a range of 1 megameter, with these particular system specifications, the receiver would receive 27 pe's/pulse.

Using these numbers, we can say that at five pe's/pulse, we are able to track a target at 1.5 mega meters to within 1 cm of the target's actual location.

5. Conclusions and Further Work

This thesis has demonstrated four main accomplishments: the determination of the importance of Doppler information with regards to tracking both target range and velocity (see Fig 3-5 and 3-6); the determination of the optimal system noise parameter; the design, construction and characterization of an experimental setup for the purpose of simulating targets at a distance from tens of meters to megameters; and the design and implementation of a real-time tracking algorithm for the photon-counting ladar program.

Concerning the importance of Doppler information in tracking target range and velocity, this thesis has shown that the current Doppler information is useful for tracking targets in both range and velocity. Furthermore, as the accuracy of the Doppler information increases the range accuracy will also increase with a quadratic relationship to the accuracy of the Doppler information, while the velocity tracking will increase with a linear relationship. Moreover, we have found that there is a lower bound in terms of tracking accuracy regardless of how accurate the Doppler information will get. This lower bound exists due to the error of the range information and system noise parameters.

Currently, the system runs at 1 Hz. This value was set to match some of the simulations developed in MATLAB. In reality, this value could be set much lower. The time to write out data and acquire the range and Doppler information currently takes around 200 ms. Thus, the system could potentially run at 5 Hz. The write to disk could potentially be further optimized, and the computation power of the receiver system increased, in order to achieve much faster update rates.

This thesis has demonstrated a working implementation of a real-time tracking algorithm for the photon-counting ladar program when the target is within 150 km. From the results shown the tracking program is able to track a target to within 7.5m in range. These values allow for certain levels of nonlinearity and tumbling motion as well.

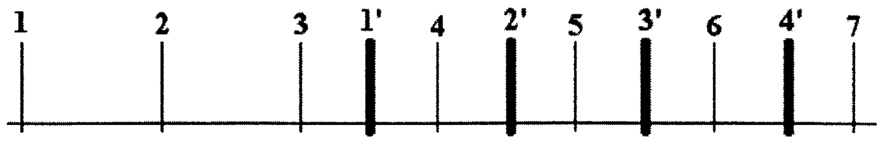


Figure 5-1: Example of the digital delay generator simulating multiple pulses in the air. The thinner lines with a number above it represent the input pulses to the digital delay generator. The thick lines with a number prime above them represent the output pulses to the digital delay generator.

To be able to allow the simulator to simulate ranges over 150 km, some post-processing will be required. Figure 5-1 shows an example of the digital delay generator simulating multiple pulses in the air. The input pulses are coming in at 1 kHz, and the digital delay generator is set to 2.5 ms. If the digital delay generator had worked as expected, the above figure is what we would expect, that pulses would begin outputting at a rate of 1 kHz beginning 2.5 ms after the first pulse was inputted into the DDG. Because the DDG does not buffer the pulses, what actually happens is that pulses 2 and 3 are ignored while pulse 1 is being delayed. Figure 5-2 shows an example of what actually happens. As can be seen, many pulses are actually dropped in between.

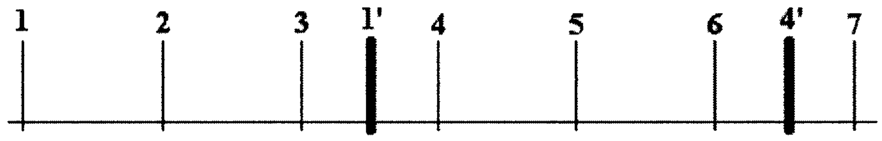


Figure 5-2: Example of what actually happens with the digital delay generator. Again thin lines represent input pulses and thick lines represent output pulses.

To get around this situation and be able to simulate pulses that go beyond the 150 km, we noticed that if we renumbered the output pulses, the delays would be less than 1 ms.

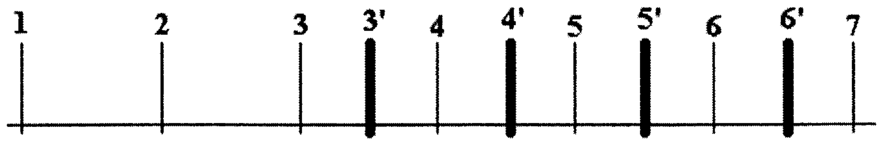
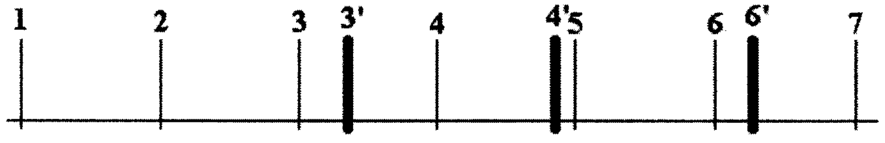


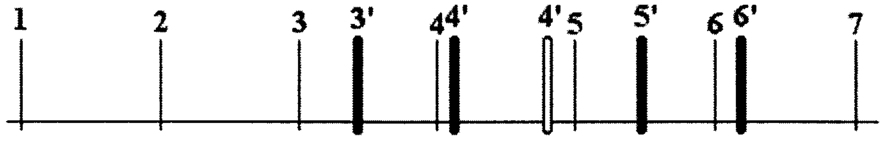
Figure 5-3: Example of renumbered output pulses

By renumbering the output pulses, we are able to recreate the original intended set of output pulses with delays less than 1 ms. So now, instead of 2.5 ms delays, we set the digital delay generator to delay the input pulses by 0.5 ms delays. The trick to this method though is that this method requires some bookkeeping. The number of whole milliseconds needs to be kept track of for each output pulse. Afterwards, when the receiver detects all the pulses, it can add back the whole milliseconds to recover the original delay sizes.

This second set of experiments requires a lot of accurate bookkeeping. It is important to know how many whole milliseconds to pad each of the delays, to recover the original delay sizes. One method of recovering the original delay sizes is to construct a table so that for each pulse there is an entry that stores the whole millisecond value. Unfortunately, this method requires very accurate time stamping and photon counting, because if even one photon is missed, the entire table becomes offset. To get around this problem, we have decided that the receiver will be fed an initial whole millisecond value corresponding to the accurate whole millisecond value of the first delay size. From there, the receiver will keep track of its own whole millisecond value, recovering the original delay sizes on its own. The whole millisecond value will not change unless the pulses begin to cross boundaries. If the pulses cross the millisecond value boundary, there are two possible scenarios: the delay size lengths until it cross a millisecond boundary, or the delay size shortens until it crosses the shorter boundary.



(a)



(b)

Figure 5-4: Examples of delay size (a) increasing past the millisecond boundary and (b) decreasing past the millisecond boundary.

Figure 5-4 shows an example of both cases. As we can see, if the delay size lengthens until it crosses a millisecond boundary, there will be one millisecond window when there is no output pulse. From this observation, we can deduct that if no peak is observed in one window, then the whole millisecond value stored in the receiver would increase by one. In the other case, where the delay size shortens past the whole millisecond boundary, one pulse will be dropped regardless. This occurs because for any one input pulse, we're only able to obtain one output pulse. Thus, one pulse will be lost regardless using this method for decreasing delay sizes.

There is no worry that the delay sizes will change so much that it is difficult to determine whether the delay size has increased or decreased. The fastest a target should be able to move is 5 km/s and corresponds to a digital delay generator shift of 33.3 ns per millisecond. This change in delay size should make it easy to determine whether the target is moving closer or farther away.

There is one concern with this method, and that is that there is a dead zone around each millisecond boundary. This is a characteristic of the TDC cards: they cannot time stamp a pulse if it sits too closely to the millisecond boundary. This is probably because there is some processing time surrounding when the common channel triggers the TDC to start the next millisecond count. This should not be too big of a concern, as long as the post-processing is able to deal with the fact that it may drop a few pulses every time a millisecond boundary is passed.

Another addition to the simulator could be to simulate not only the delay time, but also the pulse strength. The further away the target is, the weaker the energy in the return signal. The power of the return signal is roughly R^{-4}, where R is the distance from the observer to the target. Thus, the simulator also controls the attenuation of the signal based upon the distance. This adjustment is done through the same program that runs the digital delay generator. The thesis thus far has created a uniform decrease in power across the board, showing that at certain power levels the tracker is still functional. However, in reality, the power levels will be constantly changing and thus it would be interesting to see how the tracker is affected by the changing power level of the return pulse.

Further research and implementations can still occur based upon the work already done. One of the interesting sections of further research would be the design and implementation of other nonlinear filters instead of the extended Kalman filter to test how robust and accurate the other filters could become at low photon counting levels. Moreover, as more tests are performed on the tracking algorithms, there may come a point when the tracking algorithm may be removed from the experimental setup and be tested in the field.

Appendix A: Software Simulation Code

runme2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% runme2.m script
% Jonathan Watson MITLL G38 #6755 1/11/2005
% Joshua Chang, MITLL G106 Modified for Photon-Counting Laser Radar Summer 2006
%
% This .m file is the "heart" of the 6-DOF ladar algorithm testbed,
% namely this is where all of the dynamic calculations take place.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all;

% Initializations
rand('state',round(sum(clock*100)));
randn('state',round(sum(clock*100)));
global switchboard

switchboard.rungekutta = 1;        % Runge-Kutta integration of state yes/no
switchboard.step_size  = 20;       % integration step size, t = DT/step_size
switchboard.round_earth = 1;       % We want to propagate states with a ballistic
                                   % model
switchboard.rotating_earth = 1;

path(path,'Franz Functions');
path(path,'Rotations');
path(path,'Filters');

SetUpGlobals;

% Length of one time step
DT=1;

% path to trajectory file
path = 'Vandenburg_to_Kwaj2.mat';
```

```
% Process noise tuning parameter
system_noise = 0.05;

load(path);

observer_state(1, 6) = 0;

%% Begin
for TE = 1:length(target_state) - 50
  if TE==1                          % Initializations required on time step 1
    % Observer rotation matrix points directly at target at time step 1
    observer.dcm(:,:,1) = ...          %rotate to where objects ARE
      sight_along_vector(target_state(1, 1:3)-...
      observer_state(1,1:3),[0 0 0],[0 0 1]);
    xhat_minus(1,:)= target_state(1,1:6);
    Cov_minus=diag([10 10 10 1 1 1]).^2;

    % Convariance of measurement noise
    AngResError = 3e-6;   % Resolution Error
    AngPtgError = 10e-6;  % Telescope Pointing Error
    TotAngError = sqrt(AngResError ^ 2 + AngPtgError ^ 2);
    RngError = 0.15;
    VelError = 0.02;
    R=diag([TotAngError TotAngError RngError VelError]).^2;

    % Measurement noise from ladar
    AngError=TotAngError*randn(1,2) * 2;
    rangeError=RngError*randn(1) * 2;
    velocityError = VelError * randn(1);

    % Filtering step
    [xhat_plus_temp,Cov_plus]= IKalman1Ladar(observer_state(1,1:6)',...
      observer.dcm(:,:,TE),observer.dcm(:,:,TE), observer_state(1,1:6)',...
      target_state(TE,1:6)', xhat_minus(TE,:)', R, Cov_minus,...
      [AngError(:)',rangeError, velocityError],1);
    xhat_plus(TE,1:6)=xhat_plus_temp;

     % Prediction step
  [xhat_minus_temp,Cov_minus] = kf_2_acceleration(xhat_plus(TE,:)',...
    gravity(xhat_plus(TE,1:3)), Cov_plus, DT, system_noise);
  xhat_minus(TE+1,1:6)=xhat_minus_temp;

  Cov_plus_list(TE,:,:)=Cov_plus;

  % Transform ECI filter result into BCC
  R_true = [observer.dcm(:,:,TE),zeros(3);zeros(3),observer.dcm(:,:,TE)];
  true_rotated(TE,1:6)= R_true *
    (target_state(TE,1:6)'-observer_state(1,1:6)');
  [bcc_err(TE,:),cov_diag_bcc(TE,:),theta1,cov_bcc] =...
    eci2bcc(true_rotated(TE,:),xhat_plus(TE,:)',observer_state(1,1:6)',...
    squeeze(Cov_plus_list(TE,:,:)),1,observer.dcm(:,:,TE));
  [bcc_err_minus(TE,:),cov_diag_bcc_minus(TE,:),theta1_minus,
    cov_bcc_minus] = eci2bcc(true_rotated(TE,:),xhat_minus(TE,:)', ...
    observer_state(1,1:6)', squeeze(Cov_minus),1,observer.dcm(:,:,TE));
  theta1_list(TE,:)=theta1(:)';
else
  % Observer rotation matrix points directly at target at time step 1
  observer.dcm(:,:,TE) = ...          %rotate to where objects ARE
     sight_along_vector(target_state(TE, 1:3)-...
     observer_state(1,1:3),[0 0 0],[0 0 1]);

  %% Calculate Ladar measurement and noise
  AngError=TotAngError*randn(1,2) * 2;
  rangeError=RngError*randn(1) * 2;
  xhat_plus(TE,1:6)=zeros(1,6);
  xhat_minus(TE+1,1:6)=zeros(1,6);
  [xhat_plus_temp,Cov_plus]= IKalman1Ladar(observer_state(1,1:6)',...
     observer.dcm(:,:,TE),observer.dcm(:,:,TE),...
     observer_state(1,1:6)', target_state(TE,1:6)',...
     xhat_minus(TE,:)', R, Cov_minus,...
     [AngError(:)',rangeError, velocityError],1);
  xhat_plus(TE,1:6)=xhat_plus_temp;

  [xhat_minus_temp,Cov_minus] = kf_2_acceleration(xhat_plus(TE,:)',...
     gravity(xhat_plus(TE,1:3)), Cov_plus, DT, system_noise);
  xhat_minus(TE+1,1:6)=xhat_minus_temp;
```

```
    Cov_plus_list(TE,:,:)=Cov_plus;


    % Transform ECI filter result into BCC
    R_true = [observer.dcm(:,:,TE),zeros(3);zeros(3),observer.dcm(:,:,TE)];
    true_rotated(TE,1:6)= R_true *
      (target_state(TE,1:6)'-observer_state(1,1:6)');
    [bcc_err(TE,:),cov_diag_bcc(TE,:),theta1,cov_bcc] =...
       eci2bcc(true_rotated(TE,:),xhat_plus(TE,:)',observer_state(1,1:6)',...
       squeeze(Cov_plus_list(TE,:,:)),1,observer.dcm(:,:,TE));


    [bcc_err_minus(TE,:),cov_diag_bcc_minus(TE,:),theta1_minus,
       cov_bcc_minus] = eci2bcc(true_rotated(TE,:),xhat_minus(TE,:)',
       observer_state(1,1:6)', squeeze(Cov_minus),1,observer.dcm(:,:,TE));


    theta1_list(TE,:)=theta1(:)';
  end
end
```

IKalman1Ladar.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IKalman1a.m
% Jonathan Watson G.38 #6755 10/22/2004
% Joshua Chang, MITLL G106 Modified for Photon-Counting Laser Radar Summer 2006
%
% state vector is [x;y;z;xdot;ydot;zdot]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [bestguess,ErrMat] =
  IKalman1Ladar(xs,Tib,TibTr,xp,TruthC,futurestate,R,ErrMatN,Err,vis)


Id = eye(6);                  %6x6 identity
x = futurestate;
p = ErrMatN;                  %input is futurestate from previous time step


TibX = Tib(1,:); TibY = Tib(2,:); TibZ = Tib(3,:);
TibTrX = TibTr(1,:); TibTrY = TibTr(2,:); TibTrZ = TibTr(3,:);
```

```
Bx = TibX*(x(1:3)-xs(1:3));
By = TibY*(x(1:3)-xs(1:3));
Bz = TibZ*(x(1:3)-xs(1:3));
Vz = TibZ*(x(4:6)-xs(4:6));

h = [atan((TibTrX*(TruthC(1:3)-xp(1:3)))/(TibTrZ*(TruthC(1:3)-xp(1:3))));
    atan((TibTrY*(TruthC(1:3)-xp(1:3)))/(TibTrZ*(TruthC(1:3)-xp(1:3))));
    sqrt((TibTrX*(TruthC(1:3)-xp(1:3)))^2 + (TibTrY*(TruthC(1:3)-xp(1:3)))^2 +
      (TibTrZ*(TruthC(1:3)-xp(1:3)))^2)
    TibTrZ*(TruthC(4:6)-xp(4:6))];

hhat = [atan(Bx/Bz);atan(By/Bz);(Bx^2+By^2+Bz^2)^(1/2); Vz];

H = [1/(1+Bx^2/Bz^2)*1/Bz,0,-Bx/Bz^2*1/(1+Bx^2/Bz^2),0,0,0;
    0,1/(1+By^2/Bz^2)*1/Bz,-By/Bz^2*1/(1+By^2/Bz^2),0,0,0;
    1/(Bx^2+By^2+Bz^2)^(1/2)*Bx,1/(Bx^2+By^2+Bz^2)^(1/2)*By,
      1/(Bx^2+By^2+Bz^2)^(1/2)*Bz,0,0,0;
    0,0,0,0,0,1
    ]*[Tib,zeros(3);zeros(3),Tib];

if vis
    y = h+[Err(1);Err(2);Err(3); Err(4)];                  %measurement matrix
else
    y = hhat;
end

K = p*H'*inv(H*p*H'+ R);                    %Kalman gain
X = x+K*(y-hhat) ;                          %update state estimate matrix
P = (Id-K*H)*p*(Id-K*H)'+K*R*K';            %update covariance matrix

bestguess(1,:) = X;                         %current state estimate
ErrMat = P;
```

kf_2_acceleration.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% kf_2_acceleration.m
% Jonathan Watson G.38 #6755 10/22/2004
% Joshua Chang, MITLL G106 Modified for Photon-Counting Laser Radar Summer 2006
%
% state vector is [x;y;z;xdot;ydot;zdot]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [futurestate,ErrMatN] = kf_2_acceleration(Obj,Acc,ErrMat,DT,S_noise)

J = eye(3);                 %3x3 identity
F = zeros(6); F(1:3,4:6) = J;
Qc = zeros(6); Qc(4,4) = S_noise^2; Qc(5,5) = S_noise^2; Qc(6,6) = S_noise^2;
S = [-F Qc; zeros(6) F'];
Cc = expm(S*DT);            %continuous to discrete system noise matrix conversion
Phi = Cc(7:12,7:12)';       %system matrix
Qk = Phi*Cc(1:6,7:12);      %discrete system noise matrix

X = Obj;                    %input state vector
P = ErrMat;                 %input error matrix

PhiA = [eye(3)*DT^2/2;eye(3)*DT];

participant.Cd = [0.5; 0.5; 0.5];
participant.mass = 500;
participant.area = 0.1225 * pi;

drag = atmospheric_drag(X(1:6)', participant);
x = Phi*X + PhiA*(Acc)';    %future state prediction
p = Phi*P*Phi' + Qk;        %future error matrix prediction

ErrMatN = p;                         %output error matrix
futurestate(1,1:6) = x(1:6,1)';      %output state vector
```

Appendix B: Hardware Simulation Code

Wrapping Function

```
/**
 * Tracking function for photon-counting ladar
 * Inputs:
 *  temp_memory - resulting buffers from the receiver, in which data is
 *                stored with IRIG time stamps separating them.
 *  dt - time difference of previous time step
 *
 * Joshua Chang (jchang@ll.mit.edu)
 */
void find_range_from_data_buffer(char * temp_memory, unsigned long dt)
{
      // Acquire range
      double shift;
      int peak;
      int timebin = acquisition(temp_memory, 1, shift, peak);

      // Converting time_bins to range
      float range = (float)timebin * .15;
      float Doppler = (float)shift * .15 * 1000;

      // Packaging range into a measurement matrix
      TNT::Matrix< double > measurement(4, 1);
      double angle1 = 0;
      double angle2 = 0;
      measurement[0][0] = angle1;
      measurement[1][0] = angle2;
      measurement[2][0] = range;
      measurement[3][0] = Doppler;

      TNT::Matrix< double > xhat_plus(6, 1);
      // Running kalman filter
      runme2(xhat_minus, Cov_minus, observer_state, measurement, xhat_plus);
}
```

Acquistion.cpp

```
/**
 * Acquisition functions for the AARDI ladar acquisition system.
 *
 * Joshua Chang (jchang@ll.mit.edu)
 */

#include "acquisition.h"
#include "acqiris.h"
#include <iostream>
#include "windows_specific_headers.h"

static short counts[20000000]; // the histogram table, we're creating
                               // the table assuming 50 ps, though we may
                               // not use all of it.

/**
 * Takes an array of time stamps, and a time shift.  It then
 * makes a copy of the array and applies the time shift to each pulse.
 *
 * Time shift must be in a velocity ... m/ms
 *
 * Variables:
 *   array - the time stamps, each pulse, or set of time stamps
 *           are separated by 0's.
 *   shift - the time shift that each pulse is shifted by.
 *   timebin - the particular time bin that has the maximum peak
 *   numbins - the size of the time bin in ns
 *
 * Ouput: the frequency of the maximum peak.
 */
float timeshift(void * array2, double shift, int& timebin, double numbins)
{
  int pulse = 0;         // keeps track of which pulse we're looking at
  float max = 0;         // keeps track of the maximum frequency value so far
  int limit = 0;         // keeps track of how many cells have values in them
  int cleanup[100000];   // keeps track of the indices of the cells that have
                         // frequency of more than one.
```

```
int block_number = 0;
const int heart = 1;

// Looping through the time stamps

for(int block_number = 0; block_number < NUM_TC890_CARDS-1; block_number++)
{
  void * local_array = &((char *)array2)
                      [(ALLOCATED_MEMORY_PER_BUFFER_PER_CARD)*block_number];
  for (int counter = 0;; counter++)
  {
    int sample = ((int *)local_array)[counter];
    int flag = (sample & 0x80000000) >> 31;
    int channel = (sample & 0x70000000) >> 28;
    int time_stamp = sample & 0xFFFFFFF;

    if(flag == 1 || channel == 7)
    {
      break;
    }
    else if (flag == 0 && channel == 0)
    {
      pulse = time_stamp;
    }
    else if (flag == 0 && (channel > 0 && channel < 7))
    {
      // This checking portion of temp ensures that the
      // time pulses stay within the 1 ms loop
      if(heart)
      {
        // converting pico second bins to nanosecond bins
        int temp = time_stamp / 20;

        temp = temp - pulse * shift;
        if (temp < 0)
        {
          temp += 1000000 / numbins;
        }
        temp = temp % (int)(1000000 / numbins);
```

```
          // this is way to keep track of which cells need cleaning.
          // it saves time, so that we don't need to re-zero the entire array.
          if (counts[temp] == 0)
          {
            cleanup[limit] = temp;
            limit++;
          }
          // increments the cell that is keeping track of the histograms.
          counts[temp]++;
          // so that we don't have to go through the entire histogram later,
          // we can determine as we go along whether the adding of the new time
          // stamp, exceeds the maximum value, and if so, save the values.
          if (counts[temp] > max)
          {
            max = counts[temp];
            timebin = temp;
          }
        }
      }
      else
      {
        std::cout << "Eh ... Flag: " << flag << "\t Channel: " << channel << "\t
            Block #: " << block_number << std::endl;
      }
    }
  }

  // clean up counts list using the indicies that we have stored in cleanup.
  for (int i = 0; i < limit; i++)
  {
    counts[cleanup[i]] = 0;
  }

  return max;
}
```

```
/**
 * Takes an array of time stamps and determines which time bin should
 * contain the actual target.  The function currently runs through many
 * time shifts and calculates the highest peak for each one, and chooses
 * the highest peak for all of the time shifts calculated.
 *
 * Variables:
 *   array - the time stamps, each pulse, or set of time stamps
 *           are separated by 0's.
 *   numbins - the size of the time bins in ns
 *
 * Ouput: the time bin with the highest frequency count.
 */
int acquisition(void * array2, double numbins, double& shift, int& peak)
{
  float max = 0;          // max frequency of each time_shift attempt
  shift = 0;              // keep track of the best shift so far, not
                          // needed, but good for checking
  int timebin = 0;        // the best time bin so far

  LARGE_INTEGER tic, toc, ticksPerSecond, toc2;
  float  duration;

  // get the high resolution counter's accuracy
  QueryPerformanceFrequency(&ticksPerSecond);
  QueryPerformanceCounter(&tic);

  // cycles through a set of possible time stamps
  for (double counter = -50; counter < -25; counter += 1)
  {
    int temp_timebin;
    // determines the peak at each time stamp
    float peak2 = timeshift(array2, counter, temp_timebin, numbins);

    // compares the peak of the time stamp to the highest peak so far
    if (peak2 > max)
    {
      max = peak2;
      shift = counter;
      timebin = temp_timebin;
    }

  }
  QueryPerformanceCounter(&toc);

  duration = (float)(toc.QuadPart - tic.QuadPart) /
      (float)ticksPerSecond.QuadPart;
  peak = max;
  return timebin;
}
```

KalmanFilter.cpp

```
/**
 * Kalman Filter and Matrix functions for the ladar tracking system.
 *
 * Joshua Chang (jchang@ll.mit.edu)
 */

#include <math.h>
#include "KalmanFilter.h"
#include "tnt.h"

#define pi 3.14159265358979

using namespace std;

/*
 * Returns the inverse of the given matrix (only for a 4x4 matrix)
 *
 * Variables:
 *   input - a 4x4 matrix
 *
 * Ouput: a 4x4 matrix that is the inverse of the input.
*/
TNT::Matrix< double > inverse (TNT::Matrix<double> input) {
  TNT::Matrix< double > answer(4, 4);
  double aa = input[0][0], ab = input[0][1], ac = input[0][2], ad = input[0][3];
  double ba = input[1][0], bb = input[1][1], bc = input[1][2], bd = input[1][3];
  double ca = input[2][0], cb = input[2][1], cc = input[2][2], cd = input[2][3];
  double da = input[3][0], db = input[3][1], dc = input[3][2], dd = input[3][3];

  double constant = aa*bb*cc*dd-aa*bb*cd*dc-aa*cb*bc*dd+aa*cb*bd*dc+aa*db*bc*cd-
    aa*db*bd*cc-ba*ab*cc*dd+ba*ab*cd*dc+ba*cb*ac*dd-ba*cb*ad*dc-ba*db*ac*cd+
    ba*db*ad*cc+ca*ab*bc*dd-ca*ab*bd*dc-ca*bb*ac*dd+ca*bb*ad*dc+ca*db*ac*bd-
    ca*db*ad*bc-da*ab*bc*cd+da*ab*bd*cc+da*bb*ac*cd-da*bb*ad*cc-da*cb*ac*bd+
    da*cb*ad*bc;

  answer[0][0] = (bb*cc*dd-bb*cd*dc-cb*bc*dd+cb*bd*dc+db*bc*cd-db*bd*cc);
  answer[0][1] = (-ab*cc*dd+ab*cd*dc+cb*ac*dd-cb*ad*dc-db*ac*cd+db*ad*cc);
  answer[0][2] = (ab*bc*dd-ab*bd*dc-bb*ac*dd+bb*ad*dc+db*ac*bd-db*ad*bc);
  answer[0][3] = (-ab*bc*cd+ab*bd*cc+bb*ac*cd-bb*ad*cc-cb*ac*bd+cb*ad*bc);
  answer[1][0] = (-ba*cc*dd+ba*cd*dc+ca*bc*dd-ca*bd*dc-da*bc*cd+da*bd*cc);
  answer[1][1] = (aa*cc*dd-aa*cd*dc-ca*ac*dd+ca*ad*dc+da*ac*cd-da*ad*cc);
  answer[1][2] = (-aa*bc*dd+aa*bd*dc+ba*ac*dd-ba*ad*dc-da*ac*bd+da*ad*bc);
  answer[1][3] = (aa*bc*cd-aa*bd*cc-ba*ac*cd+ba*ad*cc+ca*ac*bd-ca*ad*bc);
  answer[2][0] = (ba*cb*dd-ba*cd*db-ca*bb*dd+ca*bd*db+da*bb*cd-da*bd*cb);
  answer[2][1] = (-aa*cb*dd+aa*cd*db+ca*ab*dd-ca*ad*db-da*ab*cd+da*ad*cb);
  answer[2][2] = (aa*bb*dd-aa*bd*db-ba*ab*dd+ba*ad*db+da*ab*bd-da*ad*bb);
  answer[2][3] = (-aa*bb*cd+aa*bd*cb+ba*ab*cd-ba*ad*cb-ca*ab*bd+ca*ad*bb);
  answer[3][0] = (-ba*cb*dc+ba*cc*db+ca*bb*dc-ca*bc*db-da*bb*cc+da*bc*cb);
  answer[3][1] = (aa*cb*dc-aa*cc*db-ca*ab*dc+ca*ac*db+da*ab*cc-da*ac*cb);
  answer[3][2] = (-aa*bb*dc+aa*bc*db+ba*ab*dc-ba*ac*db-da*ab*bc+da*ac*bb);
  answer[3][3] = (aa*bb*cc-aa*bc*cb-ba*ab*cc+ba*ac*cb+ca*ab*bc-ca*ac*bb);
  for (int a = 0; a < 3; a++) {
    for (int b = 0; b < 3; b++) {
      answer[a][b] = answer[a][b] / constant;
    }
  }
  return answer;
}

/*
 * Transform quaternion into rotation matrix (direction cosine matrix).
 *
 * Inputs: matrix dimensions 1 x 4
 * Outputs: matrix of dimensions 3 x 3
 */
```

```
TNT::Matrix< double > q2dcm (TNT::Matrix<double> q) {
  TNT::Matrix< double > dcm(3, 3);

  dcm[0][0] = q[0][0] * q[0][0] - q[0][1] * q[0][1] - q[0][2] * q[0][2]
      + q[0][3] * q[0][3];
  dcm[0][1] = 2 * (q[0][0] * q[0][1] + q[0][3] * q[0][2]);
  dcm[0][2] = 2 * (q[0][0] * q[0][2] - q[0][3] * q[0][1]);

  dcm[1][0] = 2 * (q[0][1] * q[0][0] - q[0][3] * q[0][2]);
  dcm[1][1] = - q[0][0] * q[0][0] + q[0][1] * q[0][1] - q[0][2] * q[0][2]
      + q[0][3] * q[0][3];
  dcm[1][2] = 2 * (q[0][1] * q[0][2] + q[0][3] * q[0][0]);

  dcm[2][0] = 2 * (q[0][2] * q[0][0] + q[0][3] * q[0][1]);
  dcm[2][1] = 2 * (q[0][2] * q[0][1] - q[0][3] * q[0][0]);
  dcm[2][2] = - q[0][0] * q[0][0] - q[0][1] * q[0][1] + q[0][2] * q[0][2]
      + q[0][3] * q[0][3];

  return dcm;
}

/*
 * multiplies two quaternions conforming to the [scalar, VECTOR] notation.
 *
 * Inputs: 1x4 quaternion vector, 1x4 quaternion vector
 * Outputs: 1x4 quaternion vector product
 */

TNT::Matrix< double > multiplyQs(TNT::Matrix< double > q1,
      TNT::Matrix< double > q2) {
  // equations from mathworld.com
  TNT::Matrix< double > q1b(3, 1), q2b(3, 1);
  q1b[0][0] = q1[1][0]; q2b[0][0] = q2[1][0];
  q1b[1][0] = q1[2][0]; q2b[1][0] = q2[2][0];
  q1b[2][0] = q1[3][0]; q2b[2][0] = q2[3][0];

  TNT::Matrix< double > temp(3, 1);
  temp = cross(q1b, q2b);
  TNT::Matrix< double > product(4, 1);
  product[0][0] = q1[0][0] * q2[0][0] - dot(q1b, q2b);
  product[1][0] = q1[0][0] * q2[1][0] + q2[0][0] * q1[1][0] + temp[0][0];
  product[2][0] = q1[0][0] * q2[2][0] + q2[0][0] * q1[2][0] + temp[1][0];
  product[3][0] = q1[0][0] * q2[3][0] + q2[0][0] * q1[3][0] + temp[2][0];
  double normPoint = sqrt(product[0][0] * product[0][0] + product[1][0]
      * product[1][0] + product[2][0] * product[2][0] + product[3][0]
      * product[3][0]);

  product[0][0] /= normPoint;
  product[1][0] /= normPoint;
  product[2][0] /= normPoint;
  product[3][0] /= normPoint;

  return product;
}

/*
 * Generates quaternions conforming to the [scalar, VECTOR] notation.
 *
 * Inputs: 1x3 angle vector
 * Outputs: 4x1 quaternion vector
 */

TNT::Matrix< double > makeQs(TNT::Matrix< double > ang) {
  TNT::Matrix< double > q(4, 1);

  double rot1st = ang[0][0] / 2;
  double rot2nd = ang[0][1] / 2;
  double rot3rd = ang[0][2] / 2;

  double sy = sin(rot3rd), cy = cos(rot3rd);
  double sp = sin(rot2nd), cp = cos(rot2nd);
  double sr = sin(rot1st), cr = cos(rot1st);

  q[0][0] =  cr * cp * cy + sr * sp * sy;
  q[1][0] =  sr * cp * cy - cr * sp * sy;
  q[2][0] =  cr * sp * cy + sr * cp * sy;
  q[3][0] = -sr * sp * cy + cr * cp * sy;
  TNT::Matrix< double > output(4, 1);
  double con = pow(q[0][0] * q[0][0] + q[1][0] * q[1][0] + q[2][0] * q[2][0]
                   + q[3][0] * q[3][0], -0.5);
  output[0][0] = con * q[0][0];
  output[1][0] = con * q[1][0];
  output[2][0] = con * q[2][0];
  output[3][0] = con * q[3][0];

  return output;
}

/*
 * Quaternion rotation sequence using euler parameters.  Produces a 3x3
 * rotation matrix.  b is the current vector found from input Ang
 * (about x, y, z).
 *
 * Inputs:
 *     a - 1x3 body line of sight vector
 *     b - 1x3 angle vector for rotation
 * Outputs:
 *     Rot - 3x3 rotation matrix
 *     q - 1x4 quaternion vector
 */

void Xe (TNT::Matrix<double> a, TNT::Matrix<double> Ang,
      TNT::Matrix<double> &ROT, TNT::Matrix<double> &q) {
  // find vector coordinates from angles
  TNT::Matrix< double > b(3, 1);

  b[0][0] = sin(pi/2 + Ang[0][0]) * sin(Ang[1][0]);
  b[1][0] = cos(pi/2 + Ang[0][0]);
  b[2][0] = sin(pi/2 + Ang[0][0]) * cos(Ang[1][0]);

  // find axis of rotation
  TNT::Matrix< double > n(3, 1);
  n = cross(a, b);

  TNT::Matrix< double > nhat(3, 1);
  double normPoint = sqrt(n[0][0] * n[0][0] + n[1][0] * n[1][0]
        + n[2][0] * n[2][0]);
    nhat[0][0] = n[0][0] / normPoint;
    nhat[1][0] = n[1][0] / normPoint;
    nhat[2][0] = n[2][0] / normPoint;

    // rotation angle
    double normab = sqrt(a[0][0] * a[0][0] + a[1][0] * a[1][0] + a[2][0]
        * a[2][0]) * sqrt(b[0][0] * b[0][0] + b[1][0] * b[1][0] + b[2][0]
        * b[2][0]);
    double theta = atan2(normPoint / normab, dot(a, b) / normab);

    // find quaternions
    TNT::Matrix< double > makeQinput(3, 1);
    makeQinput[1][0] = Ang[2][0];
    TNT::Matrix< double > q1 = makeQs(makeQinput);

    TNT::Matrix< double > q2(4, 1);
    q2[0][0] = cos(theta / 2);
    q2[1][0] = nhat[0][0] * sin(theta / 2);
    q2[2][0] = nhat[1][0] * sin(theta / 2);
    q2[3][0] = nhat[2][0] * sin(theta / 2);

    // multiply quaternions
    q = multiplyQs(q1, q2);

    TNT::Matrix< double > qnew(4, 1);
    qnew[0][0] = q[1][0];
    qnew[1][0] = q[2][0];
    qnew[2][0] = q[3][0];
    qnew[3][0] = q[0][0];
    ROT = q2dcm(qnew);

    return;
}
```

```
/*
 * Quaternion rotation sequence using euler parameters.
 *
 * Inputs: 1x3 angle vector for rotation
 * Outputs: 3x3 rotation matrix, 1x4 quaternion vector
 */
void Xe2(TNT::Matrix< double > Rotation, TNT::Matrix< double > &ROT,
      TNT::Matrix < double > &q) {
  q = makeQs(Rotation);
  TNT::Matrix< double > qnew(4, 1);
  qnew[0][0] = q[1][0];
  qnew[1][0] = q[2][0];
  qnew[2][0] = q[3][0];
  qnew[3][0] = q[0][0];
  ROT = q2dcm(qnew);
  return;
}

/*
 * Rotates observer to point along specified vector using a single-axis
 * rotation method, and also incorporates additional commanded rotations on top
 * of this initial alignment.
 *
 * Inputs: Vector - 6x1 vector, P_Ang 1x3 vector, los 1x3 vector
 * Outputs: 3x3 rotation matrix
 */

TNT::Matrix< double > sight_along_vector (TNT::Matrix<double> Vector,
      TNT::Matrix<double> P_Ang, TNT::Matrix<double> los) {
  double Xa = Vector[0][0];
  double Ya = Vector[1][0];
  double Za = Vector[2][0];

  double normPoint = sqrt(Xa * Xa + Ya * Ya + Za * Za);
  TNT::Matrix< double > np(3, 1);
  np[0][0] = Xa / normPoint;
  np[1][0] = Ya / normPoint;
  np[2][0] = Za / normPoint;
  TNT::Matrix< double > ang(3, 1);
  ang[0][0] = -asin(Ya / normPoint);
  ang[1][0] = atan2(Xa, Za);
  ang[2][0] = 0;

  TNT::Matrix< double > RM1(3, 3);
  TNT::Matrix< double > q1(1, 4);
  Xe(los, ang, RM1, q1);

  TNT::Matrix< double > RM2(3, 3);
  TNT::Matrix< double > q2(1, 4);
  Xe2(P_Ang, RM2, q2);

  TNT::Matrix< double > RotMat(3, 3);
  RotMat = RM2*RM1;

  return RotMat;
}

/*
 * gravity function
 * MATLAB code by Jonathan Watson
 *
 * Simple round-earth gravity model
 * Inputs: 1x3 position vector (eci)
 * Outputs: 1x3 acceleration vector
 */
TNT::Matrix< double > gravity (TNT::Matrix< double > location) {
  TNT::Matrix< double > Acc(3, 1);
  // gravitational constant
  float MU = (float) 3.986004415e14;
  float NL = std::sqrt(location[0][0] * location[0][0] + location[1][0]
           * location[1][0] + location[2][0] * location[2][0]);

  if (NL != 0) {
    float AC = -MU / (NL * NL);                      // keplerian gravity
    Acc[0][0] = location[0][0] / NL * AC;            // acceleration vector
    Acc[1][0] = location[1][0] / NL * AC;
    Acc[2][0] = location[2][0] / NL * AC;
  }

  return Acc;
}

/*
 * State Estimate Prediction
 *
 * Variables:
 *   Obj - xhat_plus from IKalman1Ladar
 *   Acc - acceleration of target
 *   ErrMat - Covariance matrix
 *   dt - time difference in seconds
 *   S_noise - System noise of the model
 *   futurestate - output of the new xhat
 *   ErrMatN - output of the new covariance matrix
 */
void kf_2_acceleration (TNT::Matrix< double > &Obj, TNT::Matrix< double > &Acc,
  TNT::Matrix< double > &ErrMat, float dt, float S_noise,
  TNT::Matrix< double > &futurestate, TNT::Matrix< double > &ErrMatN) {

//    defining Phi
  TNT::Matrix< double > Phi(6, 6);
  for (int c = 0; c < 6; c++) {
    Phi[c][c] = 1;
  }
  Phi[0][3] = dt;
  Phi[1][4] = dt;
  Phi[2][5] = dt;

//    defining Qk
  TNT::Matrix< double > Qk(6, 6);
  for (int d = 0; d < 3; d++) {
    Qk[d][d]     = dt * dt * dt * S_noise * S_noise / 3;
    Qk[d][d + 3] = dt * dt * S_noise * S_noise / 2;
    Qk[d+3][d]   = dt * dt * S_noise * S_noise / 2;
    Qk[d+3][d+3] = dt * S_noise * S_noise;
  }
```

```
//    defining PhiA
  TNT::Matrix< double > PhiA(6, 3);
  for (int e = 0; e < 3; e++) {
    PhiA[e][e] = dt * dt / 2;
    PhiA[e+3][e] = dt;
  }

//    defining futurestate
  futurestate = Phi * Obj + PhiA * Acc;

//    std::cout << "and defining ErrMatN\n" << std::endl;
  ErrMatN = Phi * ErrMat * TNT::transpose(Phi) + Qk;
}

/*
 * State Estimate Updates
 * based on MATLAB code from Jonathan Watson
 *
 * Variables:
 *   xs - observer state
 *   Tib - pointing rotation matrix from the observer to target
 *   futurestate - xhat plus from kf_2_acceleration
 *   R - measurement error
 *   ErrMatN - covariance matrix
 *   y - measurement
 *   bestguess - output xhat update
 *   ErrMat - output covariance matrix update
 */
void IKalman1Ladar(TNT::Matrix< double > &xs, TNT::Matrix< double > &Tib,
  TNT::Matrix< double > &futurestate, TNT::Matrix< double > &R,
  TNT::Matrix< double > &ErrMatN, TNT::Matrix< double > &y,
  TNT::Matrix< double > &bestguess, TNT::Matrix< double > &ErrMat) {

//   defining Tibs
  TNT::Matrix< double > x(futurestate);
  TNT::Matrix< double > p(ErrMatN);

  TNT::Matrix< double > TibX(1, 3);
  TNT::Matrix< double > TibY(1, 3);
  TNT::Matrix< double > TibZ(1, 3);
  for (int counter = 0; counter < 3; counter++) {
    TibX[0][counter] = Tib[0][counter];
    TibY[0][counter] = Tib[1][counter];
    TibZ[0][counter] = Tib[2][counter];
  }

//   defining Bs and Vs
  TNT::Matrix< double > xmxs(3, 1);
  TNT::Matrix< double > xmxs2(3, 1);

  for (int counter = 0; counter < 3; counter++) {
    xmxs[counter][0]  = x[counter][0]     - xs[counter][0];
    xmxs2[counter][0] = x[counter + 3][0] - xs[counter + 3][0];
  }

  TNT::Matrix< double > Bxmat = TibX * xmxs;
  TNT::Matrix< double > Bymat = TibY * xmxs;
  TNT::Matrix< double > Bzmat = TibZ * xmxs;
  TNT::Matrix< double > Vzmat = TibZ * xmxs2;

  double Bx = Bxmat[0][0];
  double By = Bymat[0][0];
  double Bz = Bzmat[0][0];
  double Vz = Vzmat[0][0];

//   defining Hhat
  TNT::Matrix< double > hhat(4, 1);
  hhat[0][0] = std::atan(Bx / Bz);
  hhat[1][0] = std::atan(By / Bz);
  hhat[2][0] = std::sqrt(Bx * Bx + By * By + Bz * Bz);
  hhat[3][0] = Vz;

//  defining H
  TNT::Matrix< double > H(4, 6);
  H[0][0] = 1 / (1 + (Bx * Bx) / (Bz * Bz)) * 1 / Bz;
  H[0][2] = -Bx / (Bz * Bz) * 1 / (1 + (Bx * Bx) / (Bz * Bz));
  H[1][1] = 1 / (1 + (Bx * Bx) / (Bz * Bz)) * 1 / Bz;
  H[1][2] = -By / (Bz * Bz) * 1 / (1 + (By * By) / (Bz * Bz));
  H[2][0] = Bx / sqrt(Bx * Bx + By * By + Bz * Bz);
  H[2][1] = By / sqrt(Bx * Bx + By * By + Bz * Bz);
  H[2][2] = Bz / sqrt(Bx * Bx + By * By + Bz * Bz);
  H[3][5] = 1;

  TNT::Matrix< double > Hrotate(6, 6);
  for (int a = 0; a < 3; a++) {
    for (int b = 0; b < 3; b++) {
      Hrotate[a][b]         = Tib[a][b];
      Hrotate[a + 3][b + 3] = Tib[a][b];
    }
  }

  H = H * Hrotate;

//   defining K
  TNT::Matrix< double > K(6, 4);
  K = p * TNT::transpose(H) * inverse(H * p * TNT::transpose(H) + R);

//   bestguess
  bestguess = x + K * (y - hhat);
  TNT::Matrix< double > Id(6, 6);
  for (int c = 0; c < 6; c++)
    Id[c][c] = 1;

//   ErrMat
  ErrMat = (Id - (K * H)) * p * (TNT::transpose(Id - (K * H))) +
      K * R * TNT::transpose(K);
}

/*
 * This function is the heart of the Kalman filter algorithm.  It does one
 * iteration of the Kalman filter update and predict steps.  The wrapping
 * function keeps track of passing the values from one time step to the next
 * time step.
 * based on MATLAB code from Jonathan Watson
 *
 * Variables:
 *   dt - time difference
 *   xhat_minus - old state of the target
 *   Cov_minus - old covariance matrix of the target
 *   observer_state - observer's current location
 *   Measurement - measurements from the observer
 *   xhat_plus_temp - output of new state of the target
*/

void runme2(TNT::Matrix< double > &xhat_minus, TNT::Matrix< double > &Cov_minus,
      TNT::Matrix< double > &observer_state, TNT::Matrix< double > &Measurement,
      TNT::Matrix< double > &xhat_plus_temp) {
  float system_noise = 50;          // Process noise tuning parameter

  TNT::Matrix< double > dcm(3, 3);
  TNT::Matrix< double > P_Ang(1, 3);
  TNT::Matrix< double > los(3, 1);
  los[2][0] = 1;

  dcm = sight_along_vector(xhat_minus - observer_state, P_Ang, los);

  // Covariance error of measurement noise
  float AngResError = 3e-6);        // Resolution error
  float AngPtgError = 10e-6);       // Telescope Pointing Error
  float TotAngError = std::sqrt(AngResError * AngResError +
    AngPtgError * AngPtgError);
  float RngError = 0.15;
  float VelError = 0.02;

  TNT::Matrix< double > R(4, 4);
  for (int a = 0; a < 4; a++) {
    for (int b = 0; b < 4; b++)
      R[a][b] = 0;
  }
  R[0][0] = TotAngError * TotAngError;
  R[1][1] = TotAngError * TotAngError;
  R[2][2] = RngError * RngError;
  R[3][3] = VelError * VelError;

// Filtering Step
  TNT::Matrix< double > Cov_plus(6, 6);
  IKalman1Ladar(observer_state, dcm, xhat_minus, R, Cov_minus,
    Measurement, xhat_plus_temp, Cov_plus);

// Prediction step
  TNT::Matrix< double > xhat_minus_temp(6, 1);
  TNT::Matrix< double > Acc(3, 1);
  Acc = gravity(xhat_plus_temp);
  kf_2_acceleration(xhat_plus_temp, Acc, Cov_plus, 1 system_noise,
    xhat_minus_temp, Cov_minus);
  xhat_minus = xhat_minus_temp;
}
```

List of References

[1] Jiang, Leaf A., Dauler, Eric A., Chang, Joshua T, "Photon-number resolving detector with 10 bits of resolution," Physical Review A. Pending.

[2] Luu, J.X. and Jiang, L.A, "Saturation Effects in Heterodyne Detection with Geiger-mode InGaAs avalanche photodiode detector arrays," Appl. Opt. 45, 16:3798-3804, 2006.

[3] Kalman, R.E., "A New Approach to Linear Filtering and Prediction Problems." Journal of Basic Engineering, 83: 35-45, 1960.

[4] Andrieu, Christophe, de Freitas, Nando, Doucet, Arnaud, Jordan, Michael I. "An Introduction to MCMC for Machine Learning." September 10, 2001. Kluwer Academic Publishers, Printed in the Netherlands, 2001

[5] Julier, Simon; Uhlmann, Jeffrey K. "A New Extension of the Kalman Filter to Nonlinear Systems."

[6] Enders, Robert H., Shapiro, Jeffrey H., "Laser radar tracking theory." Laser radar III; Proceedings of the Meeting, Boston, MA, Sept 6, 7, 1988.

[7] Zarchan, Paul and Musoff, Howard, Fundaments of Kalman Filtering: A Practical Approach, American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia. 2000.

[8] Stengel, R.F. Optimal Control and Estimation, Dover Publications, New York, NY. 1994

[9] Jelalian, A.V. Laser Radar Systems. Artech House, Portland, OR. 1992.