A REAL-TIME ALGORITHM FOR INTEGRATING DIFFERENTIAL SATELLITE AND INERTIAL NAVIGATION INFORMATION DURING HELICOPTER APPROACH

A Thesis

Presented to

the Faculty of the Graduate School

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Aeronautical Engineering

by

Ty Hoang

April 1994

AUTHORIZATION FOR REPRODUCTION
OF MASTER'S THESIS

I grant permission for the reproduction of this thesis in its entirety or any of its parts, without further authorization from me.

_____
Signature

April 1994
Date

ABSTRACT

A Real-Time Algorithm For Integrating Differential Satellite and Inertial Navigation Information During Helicopter Approach

by

Ty Hoang

A real-time, high-rate precision navigation Kalman filter algorithm is developed and analyzed. This Navigation algorithm blends various navigation data collected during terminal area approach of an instrumented helicopter. Navigation data collected include helicopter position and velocity from a global position system in differential mode (DGPS) as well as helicopter velocity and attitude from an inertial navigation system (INS). The goal of the Navigation algorithm is to increase the DGPS accuracy while producing navigational data at the 64 Hertz INS update rate. It is important to note that while the data was post flight processed, the Navigation algorithm was designed for real-time analysis.

The design of the Navigation algorithm resulted in a nine-state Kalman filter. The Kalman filter's state matrix contains position, velocity, and velocity bias components. The filter updates positional readings with DGPS position, INS velocity, and velocity bias information. In addition, the filter incorporates a sporadic data rejection scheme. This relatively simple model met and exceeded the ten meter absolute positional requirement.

The Navigation algorithm results were compared with truth data derived from a laser tracker. The helicopter flight profile included terminal glideslope angles of 3, 6, and 9 degrees. Two flight segments extracted during each terminal approach was used to evaluate the Navigation algorithm. The first segment recorded small dynamic maneuver in the lateral plane while motion in the vertical plane was recorded by the second segment. The longitudinal, lateral, and vertical averaged positional accuracies for all three glideslope approaches are as follows (mean ± two standard deviations in meters): longitudinal (-0.03 ± 1.41), lateral (-1.29 ± 2.36), and vertical (-0.76 ± 2.05).

ACKNOWLEDGMENTS

The expertise and patience of Dr. Stanley F. Schmidt are greatly appreciated in teaching this novice designer the inner workings of the Kalman filter. Thank you Dr. Daniel J. Biezad for giving me the opportunity to do my best and for being flexible with the direction of this project. Thank you Mr. Harry N. Swenson for your motivational support and confidence. With gratitude to Mr. David N. Kaufmann, for his technical inputs and aid with the continuity between the two projects. And a special thanks to the engineers and computer support personnel at the Guidance and Navigation Branch of NASA Ames Research Center for their inputs and support of this project. Lastly, I like to thank my parents Minh & Tinh Hoang, and my family whom emotional and financial supports are often taken for granted, but not this time.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER 1

1 Introduction 1

Previous Works 1

Problem Definition 2

2 Flight Test Data Collection 3

Test Equipment and Material 3

Flight Test Profile 5

3 Flight Data Reduction And Verification 8

Data Reduction 8

Analysis of the Navigation Data 11

INS Data Reduction 11

DGPS Data Reduction 12

Data Manipulation 16

Analysis of the Truth Data 16

INS Data Reduction 16

Laser Data Reduction 16

Data Manipulation 17

Data Verification and Selection 17

4 Overview And Design Of The Kalman Filter 20

Kalman Filter Theory 20

Design of the Navigation Filter 25

Design of the Truth Filter 26

5 Navigation Algorithm Results 27

6 Summary and Recommendations 34

Summary 34

Recommendations 35

REFERENCES 36

A Navigation Algorithm Written in Matlab Code 37

B Truth Algorithm Written in Matlab Code 48

C Locations of Navigation Components 58

LIST OF TABLES

Table Page

4.1 The Nine State Navigation Filter.....25

4.2 The Six State Truth Filter.....26

5.1 Navigation Filter and Time History Results - 3, 6, and 9 Degree Samples.....32

LIST OF FIGURES

Figure Page

2.1 UH-60A RASCAL Flight Test Helicopter 3
2.2 Airborne System 4
2.3 Ground Based Systems 4
2.4 Flight Test Profile at Crows Landing Runway 35 5
2.5 Sample Laser Ground Track Profile 6
2.6 Vertical Descent Profile 7
2.7 Sample Laser Vertical Descent Profile - 6° Glideslope 7
3.1 Sample Ground Track Plot of Raw Laser and DGPS Data 9
3.2 Sample Vertical Descent of Raw Laser and DGPS Data 9
3.3 Sample Ground Track Plot of Truth and Navigation Data 10
3.4 Sample Vertical Descent Plot of Truth and Navigation Data 10
3.5 WGS-84 Ellipsoid and ECEF Reference Frame 13
3.6 Sample Laser Ground Track Profile with Data Drop-out 18
3.7 Sample Laser Vertical Descent Profile with Data Drop-out 18
3.8 Sample Laser Ground Track Profile Showing Small Dynamic Maneuvers 19
4.1 Kalman Filter Loop 22
4.2 Modified Kalman Logical Loop 22
4.3 System Block Diagram of the Navigation Filter 23
4.4 Algorithm Flow Diagram 24
5.1 Representative Navigation and Truth Ground Track Profile 27
5.2 Representative Navigation and Truth Vertical Descent Profile 28
5.3 3° Longitudinal Errors 29

Figure Page

5.4 3° Lateral Errors....29
5.5 3° Vertical Errors....30
5.6 3° Ground Track Profile - Raw Laser Data....31
5.7 Statistical Results Showing Mean ± 2s for 3°, 6° and 9° Flights....31

CHAPTER 1

INTRODUCTION

Previous Works

Considerable attention has been given to the integration of inertial navigation systems (INS) and the differential global positioning systems (DGPS) via the Kalman filter to provide precision navigation information. Work has been done on both fixed-wing [1-3] and rotary wing aircraft [4-7] using precision (P) and course acquisition (C/A) code. DGPS research at NASA Ames Research Center was initiated in the early 1980's using a Sikorsky SH-3G helicopter. The objective of the tests was to evaluate the use of DGPS to support helicopter terminal approach operations.

The helicopter was equipped with an early research DGPS system. Final approach positioning accuracy was 5.2 ± 8.0 meters (mean $\pm 2\sigma$) laterally and 5.0 ± 4.0 m vertically with radar altimeter enhancement. Since then, commercial GPS receivers have made significant improvements in positional accuracy. A recent NASA rotary wing project using commercial DGPS yield the following non real-time result during final approach. Navigation accuracy resulted in -0.79 ± 2.74 m laterally and -2.03 ± 3.54 m vertically [4].

Currently, NASA Ames Research Center and the U.S. Army Aeroflightdynamics Directorate are developing a research rotorcraft, the Rotorcraft-Aircrew Systems Concepts Airborne Laboratory (RASCAL). The RASCAL's UH-60A Black Hawk helicopter was used as the flight test vehicle which will be modified in stages to support flight research of advanced guidance, control and pilot display programs. Among the requirements of these programs is to provide navigational information that has a minimum of 10 meter absolute accuracy with a sample rate of 20 Hz [8]. It is this navigation requirement that the current thesis sets out to address.

Problem Definition

The purpose of this research is to provide a real-time navigation algorithm integrating DGPS and INS navigation data. The goal is to design a simple algorithm that would not tax the performance of the onboard computer, thereby displaying timely and accurate readings. Real-time implies that computational time requirements are less than clock time during program execution. Although work accomplished by Kaufmann [4] meets RASCAL's positional requirement, it encountered two important short comings: 1) the DGPS update rate was only at 2 Hz and 2) the DGPS time-lag was not implemented in real-time. The current navigation algorithm addresses and resolves both short comings encountered in Kaufmann's research, in addition to providing real-time positional updates.

CHAPTER 2

FLIGHT TEST DATA COLLECTION

Test Equipment and Material

To satisfy RASCAL's navigation requirement, a C/A-code global positioning system with differential upload capability was installed on the UH-60A. The contribution of these two components makes a differential global positioning system (DGPS). An additional ground-based differential uplink GPS system was located at a pre-surveyed site. The data from the DGPS receiver and navigation information from the inertial navigation system (INS) was sent to the onboard Data Acquisition Computer (DAC). The DAC collects GPS position data at 2 Hz and INS Euler angle and velocity data at 64 Hz.

The flight tests were conducted at Crows Landing located approximately 50 miles east of the Moffett Field Naval Air Station. Data acquisition was divided into two major systems: airborne and ground-based. The airborne system includes the RASCAL UH-60A helicopter which was equipped with an Ashtech Model XII 12 channel C/A code GPS receiver, a Maxon/Ashtech SM 3010 VHF telemetry uplink receiver, a Litton LN-93 ring laser gyro inertial navigation system (barometric altimeter aided), a laser reflector on each side of the fuselage, and an 80486 data acquisition computer. Figure 2.1 shows the flight test vehicle with laser, GPS, and differential uplink placements. A schematic of the airborne system integration is shown in Figure 2.2.

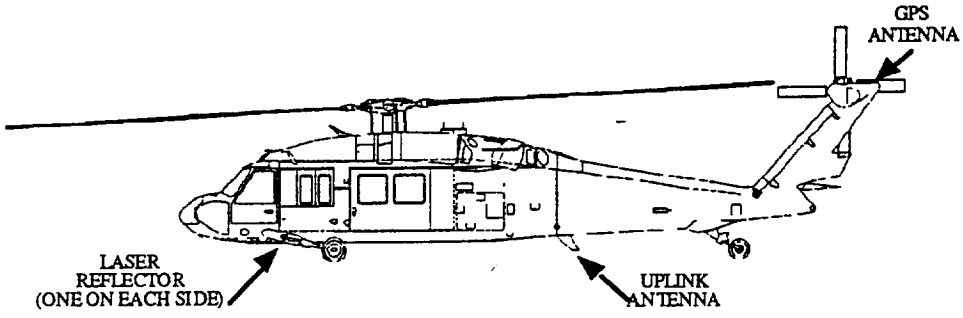


Figure 2.1 UH-60A RASCAL Flight Test Helicopter [4]

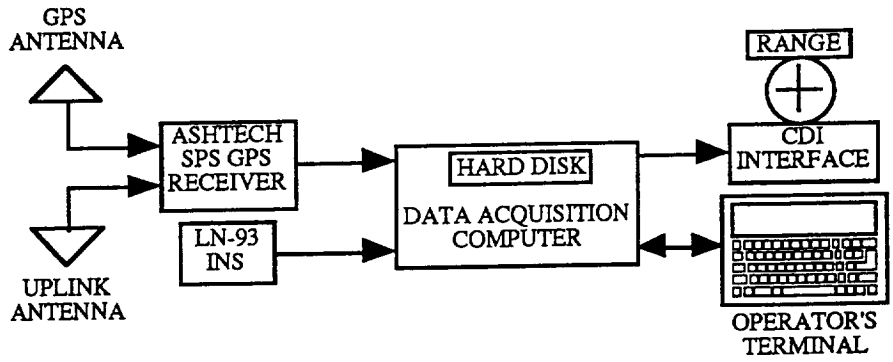


Figure 2.2 Airborne System [4]

The ground-based system consists of two components. The first component includes the same Ashtech GPS receiver and a Maxon/Ashtech telemetry uplink transmitter providing the differential correction. The second component, a laser tracker system located at Crows Landing NAS provides precision truth data. The two components of the ground-based systems are illustrated in Figure 2.3

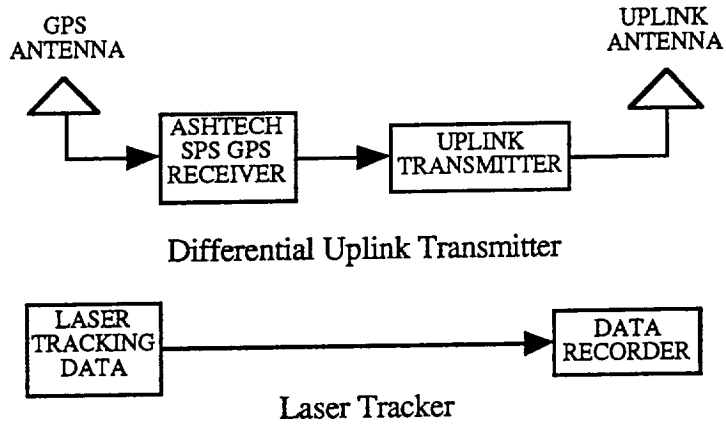
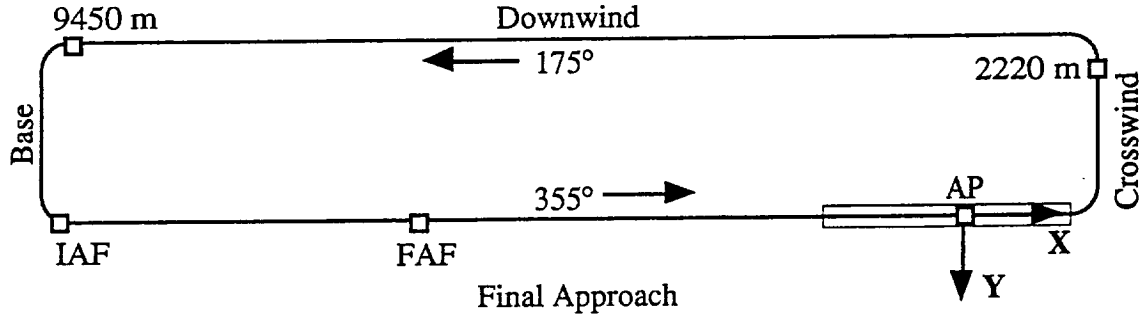


Figure 2.3 Ground Based Systems [4]

Bouncing off the reflector mounted on the side of the helicopter, the laser tracker provides precise range, azimuth, and elevation truth data at 100 Hz. Laser range accuracy is nominally ± 0.3 m out to approximately 9 km while azimuth and elevation accuracy are nominally ± 0.2 mrad. Altogether, DGPS data is collected at 2 Hz, INS data at 64 Hz, and laser data at 100 Hz.

Flight Test Profile

The flight test helicopter collects data by flying in a rectangular pattern. Figure 2.4 illustrates the pattern as seen from above Runway 35 at Crows Landing. Starting at the Aim Point (AP), the helicopter flies crosswind, downwind, around the base and up the final approach segment to finish the flight profile. Five important locations are identified on the figure. Data collection is initiated at approximately 9450 meters (X-axis) down of the AP. At the Initial Approach Fix (IAF) point, the helicopter is on altitude, on course, and on speed. The helicopter then intercepts the appropriate glideslope at the Final Approach Fix (FAF) and descends toward the AP. At the AP, the helicopter arrests its rate of descent, levels off, and flies crosswind. Data collection is terminated about 2200 meters after the AP. Afterward, the helicopter flies downwind to complete the run and restarts the process. The AP is the origin of the runway coordinate system (RCS), which has the X-axis (longitudinal) pointing along the runway centerline with the Y-axis (lateral) pointing right of runway and the Z-axis (vertical) pointing down, normal to the runway. The length of the Base segment is 1852 meters.



Note: all values indicate X distances

Figure 2.4 Flight Test Profile at Crows Landing Runway 35

Compared to Figure 2.4, a laser data representation of the base and final approach segment is plotted in Figure 2.5. The IAF and FAF points are located at -9960 and -5560 meters respectively along the X axis. The AP is located at the origin of the figure.

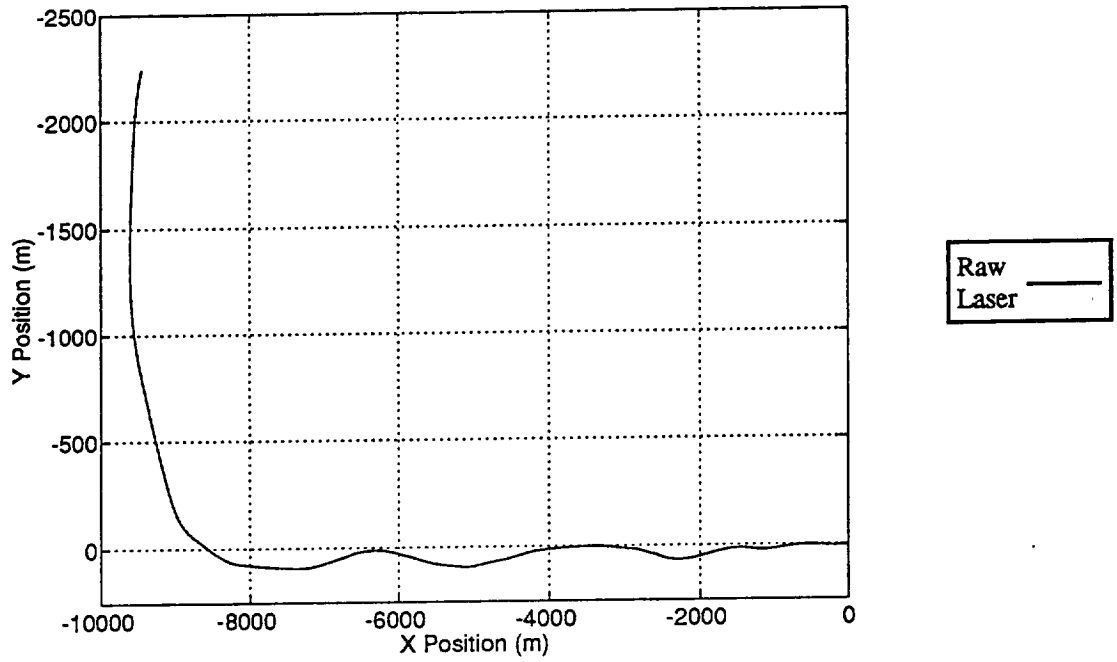


Figure 2.5 Sample Laser Ground Track Profile

The flight path between the IAF and AP point is composed of two maneuvers. The helicopter flies a "straight and level" approach from the IAF to the FAF point, as shown in Figure 2.6. The descent portion is carried out between the FAF and AP point. This project investigates the standard 3 degree approach as well as various other glideslope angles (steeper 6 and 9 degree). By holding the distance between the IAF and FAF constant during the straight and level flight segment, each glideslope angle had its unique approach altitude. For the 3, 6, and 9 degree glideslopes, the approach altitudes are 340, 640, and 910 meters respectively. Once again, the descent segment as recorded by the laser tracker is shown in Figure 2.7.

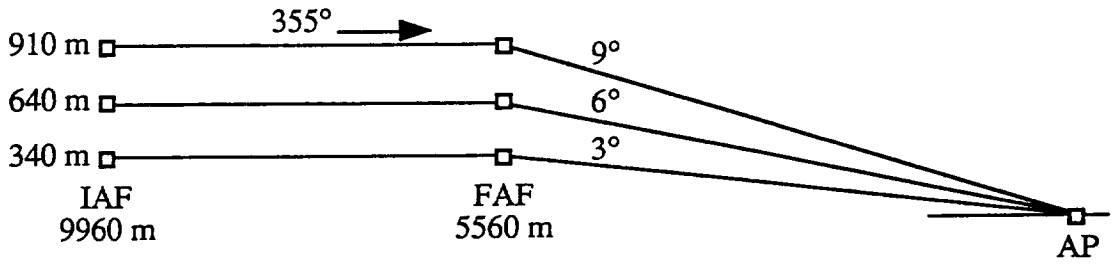


Figure 2.6 Vertical Descent Profile

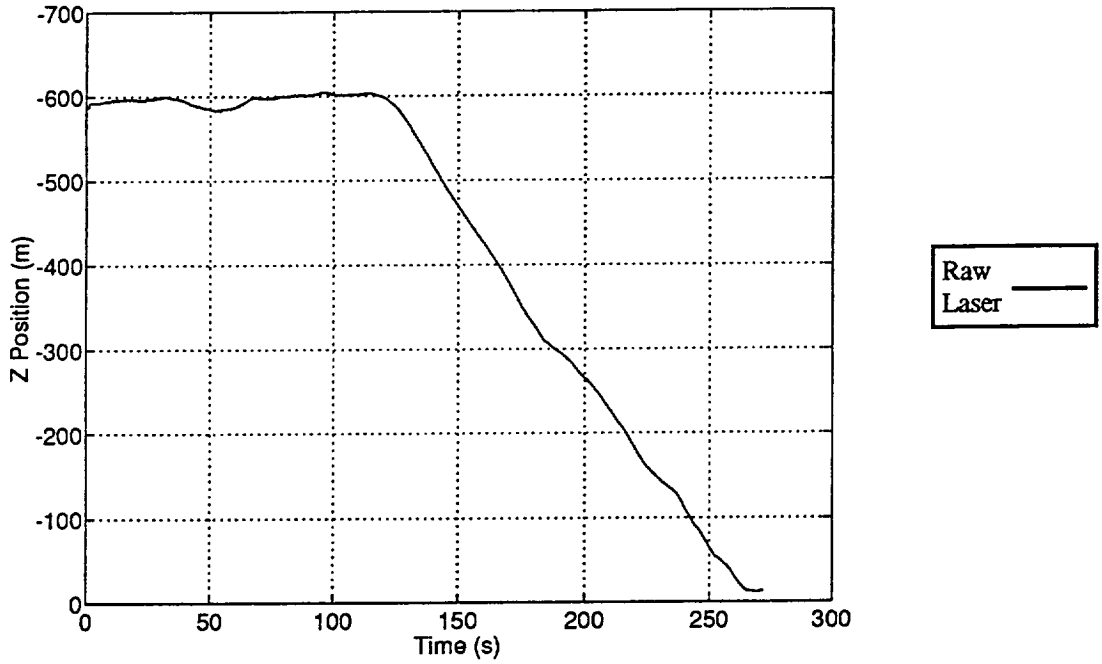


Figure 2.7 Sample Laser Vertical Descent Profile - 6° Glideslope

Unlike the ground track profile, the vertical descent profile is plotted against time instead of position to show a better representation of the level flight segment. Starting the descent around 600 m, the figure represents a 6 degree approach glideslope. With reference to the Figure 2.5, the turn from the Base to Final Approach segment occurs between 40 and 60 seconds. Although the plots show data from the Base segment up to the AP, the algorithm analyzes data only from the IAF to the AP locations.

CHAPTER 3

FLIGHT DATA REDUCTION AND VERIFICATION

Data Reduction

During flight test, three sets of data were recorded for analysis: laser, INS and DGPS. The Navigation algorithm was established to provide positional information. To validate the Navigation data, it was compared to a laser derived "Truth" data. Truth data is obtained via the Truth algorithm. The performance of the Navigation model was determined by calculating the deviations between the Navigation and Truth data.

The Truth algorithm merges laser and INS data to reduce noise and synchronizes laser data at 100 Hz to INS's 64 Hz update rate. The Navigation algorithm integrates the airborne system's INS and DGPS data. The data are presented to the algorithms in the RCS reference frame. A segment of a typical ground track and vertical descent profile is graphed in Figures 3.1 and 3.2. Note that the positions are plotted against a common Irig B time stamp The solid line represents raw laser data and the dashed line represents DGPS data.

While Figures 3.1 and 3.2 show the raw data, Figures 3.3 and 3.4 display the reduced data in the form of the Truth and Navigation data. The solid line represents Truth data and the dashed line is the Navigation data. Compared to the raw data, the reduced data correlates very well. In addition, a time-lag update correction has been factored into the Navigation algorithm. This is most apparent when comparing the peak time locations between the raw and reduced data. Note the reduction in spike amplitude in the reduced data plots. The following sections present the procedures required to reduce data such as that in Figures 3.1-3.2 to Figures 3.3-3.4. In addition, both source codes are available in Appendix A and B. Codes are written in Matlab script format [9]. Every effort has been made to keep the design of the algorithm simple, in addition to providing precise, high rate positional information.

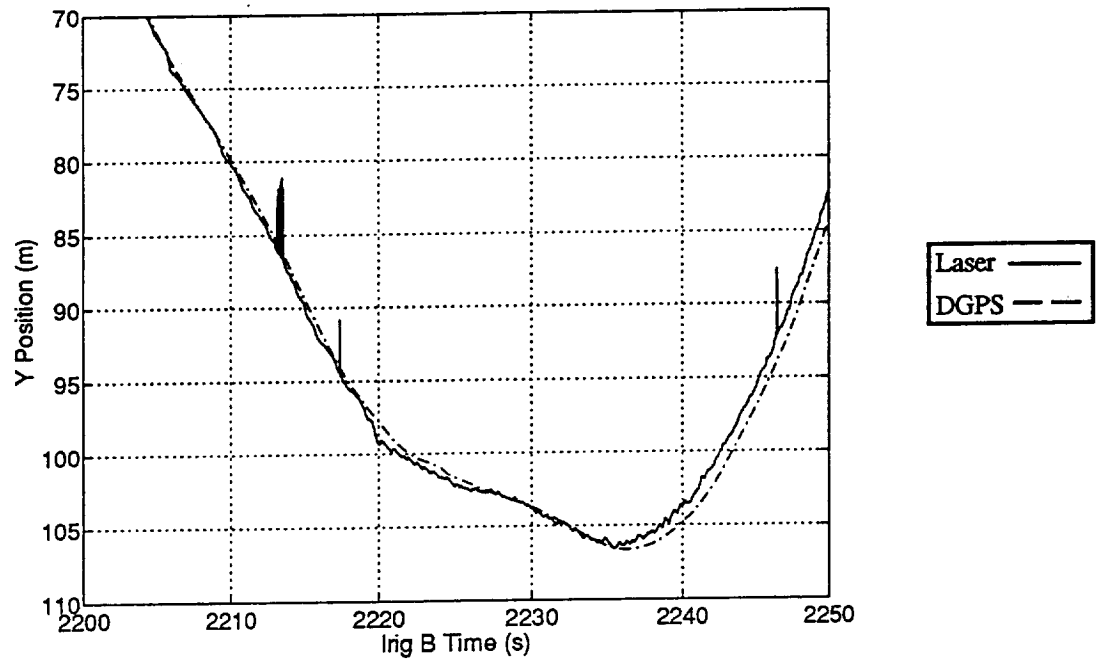


Figure 3.1 Sample Ground Track Plot of Raw Laser and DGPS Data

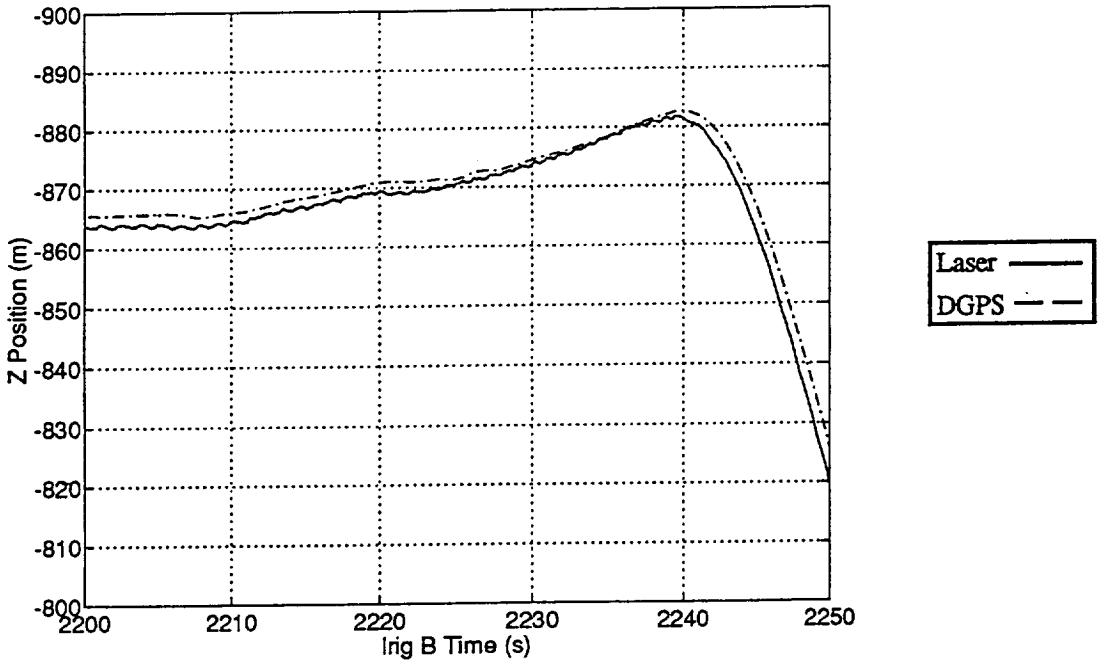


Figure 3.2 Sample Vertical Descent of Raw Laser and DGPS Data

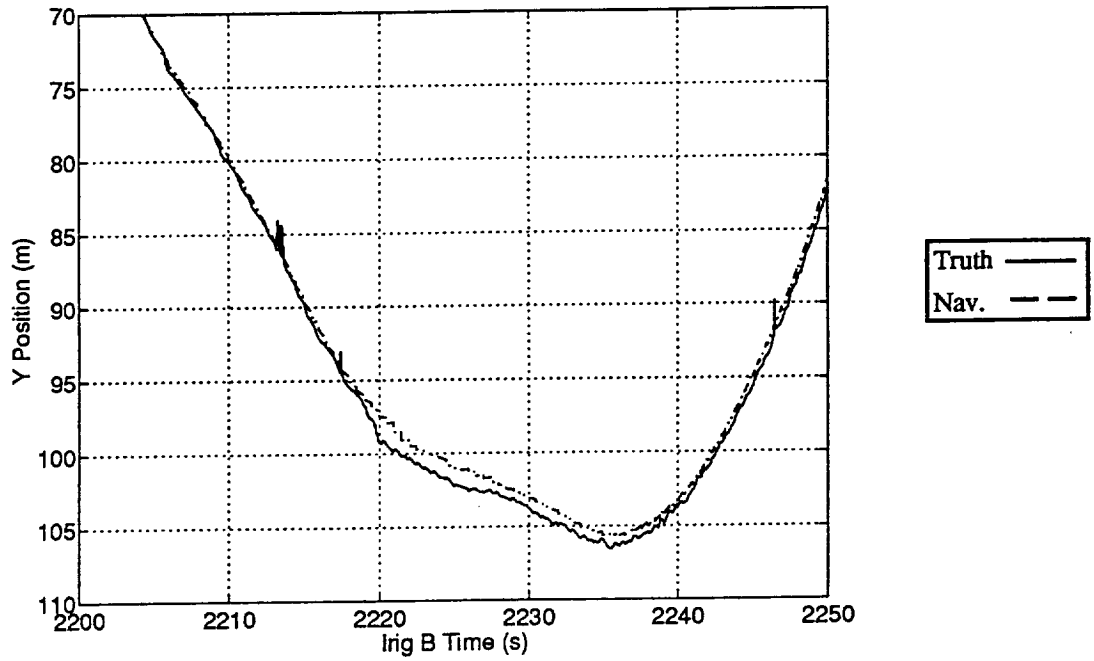


Figure 3.3 Sample Ground Track Plot of Truth and Navigation Data

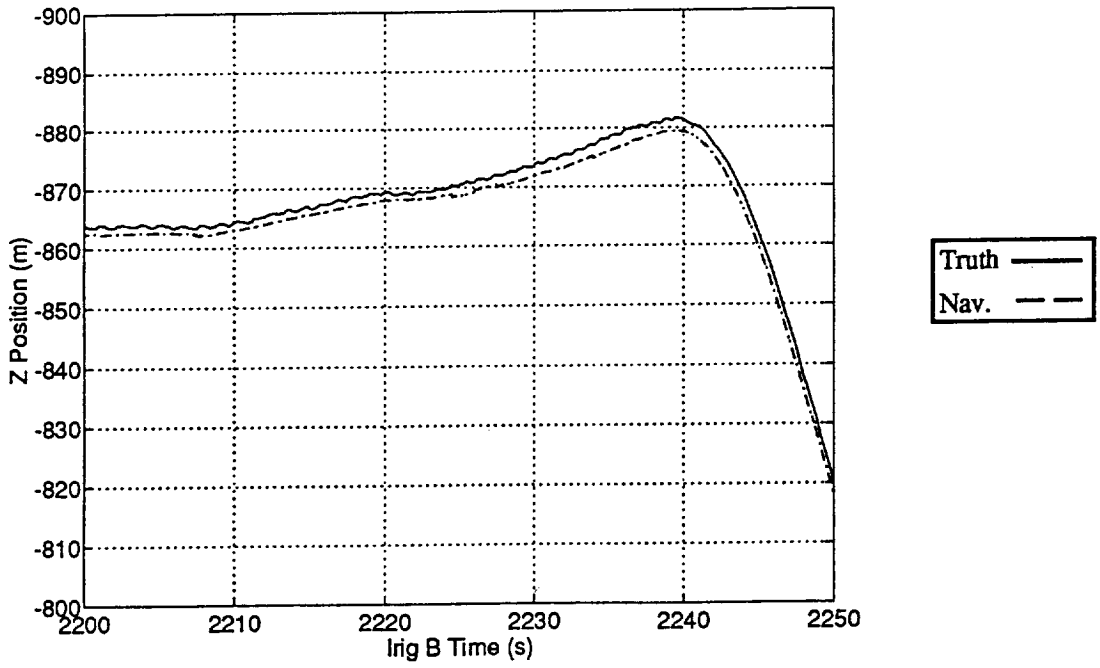


Figure 3.4 Sample Vertical Descent Plot of Truth and Navigation Data

Analysis of the Navigation Data

INS Data Reduction

A common time source, Irig B time, was used by both the airborne and ground-based systems. In addition, the INS data set required a considerable amount of matrix transformation before it could be used by the Kalman filter. The transformations were necessary to supply the Kalman filter with Irig B time (ms), yaw angle (degree), azimuth angle (degree), X velocity (m/s), Y velocity(m/s), and Z velocity (m/s) in the runway coordinate system. Converting INS velocities into the RCS reference frame required the transformations in Equation 3.1. Note that the INS velocity vector (Vins) for each axis is represented by a 3 by 1 matrix. The velocity data recorded by the INU is in the platform reference frame, which has the X-axis oriented North, Y-axis pointing West, both rotated by the wander angle, and the Z-axis pointing up. Recall that the RCS reference frame has the X-axis parallel to the runway, Y-axis pointing right and Z-axis pointing down.

$$V_{rcs} = \left[C_{pwu}^{rcs}\right]\left[C_{nwu}^{pwu}\right]\left[C_{ins}^{nwu}\right]\left[V_{ins}\right] \tag{3.1}$$

$$V_{ins} = \begin{bmatrix} \dot{X}_{ins} \\ \dot{Y}_{ins} \\ \dot{Z}_{ins} \end{bmatrix}$$

rcs = RCS frame, Parallel, East and Down into runway
pwu = Parallel, West and Up frame
nwu = North, West and Up frame
ins = Platform frame, North, West (rotated by wander angle)-and Up

In Equation 3.1, C_{ins}^{nwu} is the transformation matrix of the INU platform into the North, West and up reference frame. This transformation is shown in Equation 3.2, where α is defined as the difference between the platform azimuth angle and the yaw angle (in radians). The value of the α angle is calculated with every INS update.

$$C_{ins}^{nwu} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

α = platform azimuth angle - yaw angle

The C_{nwu}^{pwu}, matrix in Equation 3.3 transforms the helicopter to a system parallel to the runway. This time the transformation rotates about the wander angle, ø. The wander angle is defined as the angle between the runway X-axis and true North. The wander angle was recorded at 10.099 degree. Note that ø is analyzed in radians.

$$C_{nwu}^{pwu} = \begin{bmatrix} \cos(ø) & -\sin(ø) & 0 \\ \sin(ø) & \cos(ø) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

ø = 10.099 degree

The final transformation brings the INS velocities into the runway coordinate system. This simple transformation (Equation 3.4) rotates the Y-axis and Z-axis 180 degrees.

$$C_{pwu}^{rcs} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.4}$$

The order of these transformations are crucial in providing correct data for processing by the Kalman filter.

DGPS Data Reduction

DGPS data was recorded in the Earth-Centered Earth Fixed (ECEF) reference frame. This axis system has the Z-axis pointing from the center of the earth through the North Pole, the X-axis through the Greenwich prime meridian at the equator and the Y-axis

orthogonally rotated (see Figure 3.5). DGPS measurements include Irig B time (ms), X, Y, and Z positions (m) and differential X, Y, and Z velocities (m/s).

The first step transforms the WGS-84 geodetic coordinates of the Aim Point to the ECEF reference frame [4]. The graphical representation of the relationship between the WGS-84 ellipsoid and the ECEF reference frame is shown in Figure 3.5. The geodetic height, h represents the length of the ellipsoidal normal from the surface of the ellipsoid to the point P (say the Aim Point). The geodetic latitude, φ is the angle between the ellipsoidal normal and the equator. The geodetic longitude, λ defines the angle between two meridional planes oriented counter-clockwise from the ECEF's X axis.

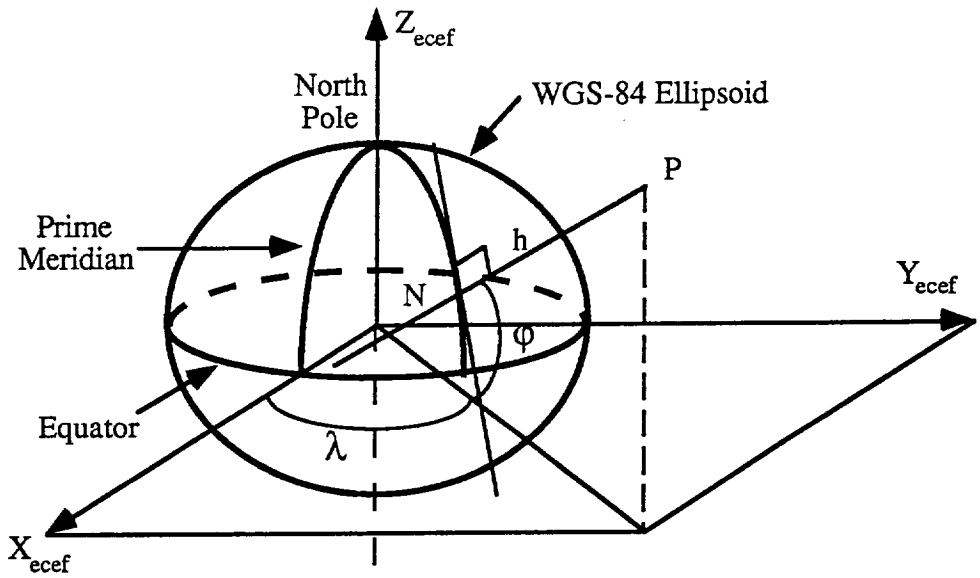


Figure 3.5 WGS-84 Ellipsoid and ECEF Reference Frame

The Aim Point can now be transform into the ECEF reference frame by the relationship shown in Equation 3.5. The radius of the earth is defined as N (Equation 3.6) and e is the eccentricity of the earth ellipsoid of evolution (Equation 3.7). In Equation 3.7, a and b represent the semi-major and semi-minor axis (in meters) of the earth ellipsoid of evolution respectively.

Note that the φ and λ are analyzed in radians. Following the ECEF reference convention, λ has a value of -121.1082725 degrees.

$$\begin{bmatrix} AP_{Xecef} \\ AP_{Yecef} \\ AP_{Zecef} \end{bmatrix} = \begin{bmatrix} (N+h)\cos(\varphi)\cos(\lambda) \\ (N+h)\cos(\varphi)\sin(\lambda) \\ (N(1-e^2)+h)\sin(\varphi) \end{bmatrix} \tag{3.5}$$

$$N = \frac{a}{\sqrt{1-e^2\sin^2(\varphi)}} \tag{3.6}$$

$$e = \sqrt{\frac{2(a-b)}{a} - \frac{(a-b)^2}{a^2}} \tag{3.7}$$

a = 6378137.0 m
b = 6356752.3141 m
φ = 37.41335361 degree
λ = -121.1082725 degree
h = 12.4 m

With the location of the Aim Point calculated in the ECEF frame, the distance between the airborne GPS receiver antenna and the Aim Point can be determined. Equation 3.8 displays this simple calculation.

$$\begin{bmatrix} \Delta X_{ecef} \\ \Delta Y_{ecef} \\ \Delta Z_{ecef} \end{bmatrix} = \begin{bmatrix} Antenna_{Xecef} \\ Antenna_{Yecef} \\ Antenna_{Zecef} \end{bmatrix} - \begin{bmatrix} AP_{Xecef} \\ AP_{Yecef} \\ AP_{Zecef} \end{bmatrix} \tag{3.8}$$

The next process takes the GPS receiver antenna position in the ECEF frame and transforms it into the RCS reference frame. The transformation is outlined in Equations 3.9-3.12.

$$\begin{bmatrix} \text{Antenna}_{Xrcs} \\ \text{Antenna}_{Yrcs} \\ \text{Antenna}_{Zrcs} \end{bmatrix} = \left[C_{ecef}^{rcs}\right] \begin{bmatrix} \Delta X_{ecef} \\ \Delta Y_{ecef} \\ \Delta Z_{ecef} \end{bmatrix} \tag{3.9}$$

The C_{ecef}^{rcs} matrix transforms ECEF data into RCS reference frame.

$$C_{ecef}^{rcs} = \left[C_{vcv}^{rcs}\right]\left[C_{ecef}^{vcv}\right] \tag{3.10}$$

The C_{ecef}^{vcv} matrix brings ECEF coordinates into the Vehicle-Carried Vertical (VCV) coordinate system. VCV has positive X oriented towards True North, positive Y in True East, and positive Z pointing down, normal to the runway. φ and λ were defined previously. VCV is transformed into the RCS system via the C_{vcv}^{rcs} matrix. The True Heading (H) of Runway 35 is 10.099 degrees.

$$C_{ecef}^{vcv} = \begin{bmatrix} -\sin(\varphi)\cos(\lambda) & -\sin(\varphi)\sin(\lambda) & \cos(\varphi) \\ -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\varphi)\cos(\lambda) & -\cos(\varphi)\sin(\lambda) & -\sin(\varphi) \end{bmatrix} \tag{3.11}$$

$$C_{vcv}^{rcs} = \begin{bmatrix} \cos(H) & \sin(H) & 0 \\ -\sin(H) & \cos(H) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

The DGPS positions and differentially derived velocities are transformed from the ECEF to the RCS reference frame via Equation 3.13.

$$X_{rcs} = \left[C_{ecef}^{rcs}\right]\left[X_{ecef}\right] \;,\quad \dot{X}_{rcs} = \left[C_{ecef}^{rcs}\right]\left[\dot{X}_{ecef}\right] \tag{3.13}$$

One last transformation is required before the data is to be processed by the Kalman filter. In the RCS reference frame, the helicopter's GPS antenna is translated to the INU location through Equations 3.14-3.15. The location of the INU was selected as the origin of the helicopter.

$$\begin{bmatrix} X_{gps} \\ Y_{gps} \\ Z_{gps} \end{bmatrix} = \begin{bmatrix} Antenna_{Xrcs} \\ Antenna_{Yrcs} \\ Antenna_{Zrcs} \end{bmatrix} - \begin{bmatrix} C_{antenna}^{inu} \end{bmatrix} \tag{3.14}$$

$$\begin{bmatrix} C_{antenna}^{inu} \end{bmatrix} = INU_{location} - GPS_{location} = \begin{bmatrix} -11.74115 \\ 0.8128 \\ -3.0861 \end{bmatrix} \tag{3.15}$$

Equation 3.14 produces the vital navigation information that the Kalman filter requires to predict the next positional estimate. Note that the units in Equation 3.15 are in meters. More details on component locations are available in Appendix C. This entire process from Equations 3.1-3.14 is repeated for each INS and DGPS update, up to 64 times a second.

Data Manipulation

After the appropriate transformations, the Navigation algorithm checks for and reject sporadic data. The data sets are then sent to the Kalman filter where INS and DGPS data are merged. The Kalman filter provides navigational position and velocity estimates, at a much higher rate.

Analysis of the Truth Data

INS Data Reduction

See "Analysis of the Navigation Data-INS Data Reduction" in the previous section.

Laser Data Reduction

Laser data is the simplest of the three data sets to analyze. Data collected include Irig B time (ms), X, Y, and Z positions (m) in the RCS frame. The laser reflector was transformed once, from the starboard side reflector location to the INU. Appendix C provides more detail and the location of each components. Although a laser reflector is situated on each side of the helicopter, the translation to the INU was only made for the

starboard reflector. Due to time constraint, analysis of the port reflector was not examined. The problem of reflector ambiguity was brought up by Kaufmann [4], which this algorithm partially addresses. Chapter 5 provides more detail and results.

Data Manipulation

The first procedure requires synchronization of the INS and laser data sets via the Irig B time stamp. When Irig B time matches laser data is interpolated to match INS data rate. This is required since the INS operates at 64 Hz while the laser operates at 100 Hz. The algorithm used a linear interpolation routine. The data sets then filters out and reject sporadic data. The INS and laser data sets are now suitable for input into the Kalman filter.

Data Verification and Selection

After the proper transformations into the RCS system, each data set is visually inspected for data consistency. The criteria for selecting the sample data sets includes: 1) a complete data set with minimal data drop-outs 2) a flight profile with small dynamic maneuvers and 3) a flight path representative of other data sets.

A complete data set is defined as a sample which contain INS, DGPS, and laser data, all synchronized to the common Irig B time. For this analysis, only small dynamic maneuvers are examined. Highly dynamic maneuvers are not analyzed since the investigation only concentrated on helicopter final approach maneuvers. Lastly, the sample data set selected for analysis was representative of the other data sets.

Although the algorithm is designed to handle sporadic data measurement, data set with excessive data drop-out were not analyzed. Figures 3.6-3.7 show a sample 3 degree laser data set that was not analyzed in its entirety due to excessive data drop-out. Because the drop-out occurred at the beginning of the approach and the rest of the data set met all the criteria stated above, this particular sample was analyzed. Again, the algorithm only analyze data between the IAF and AP locations.

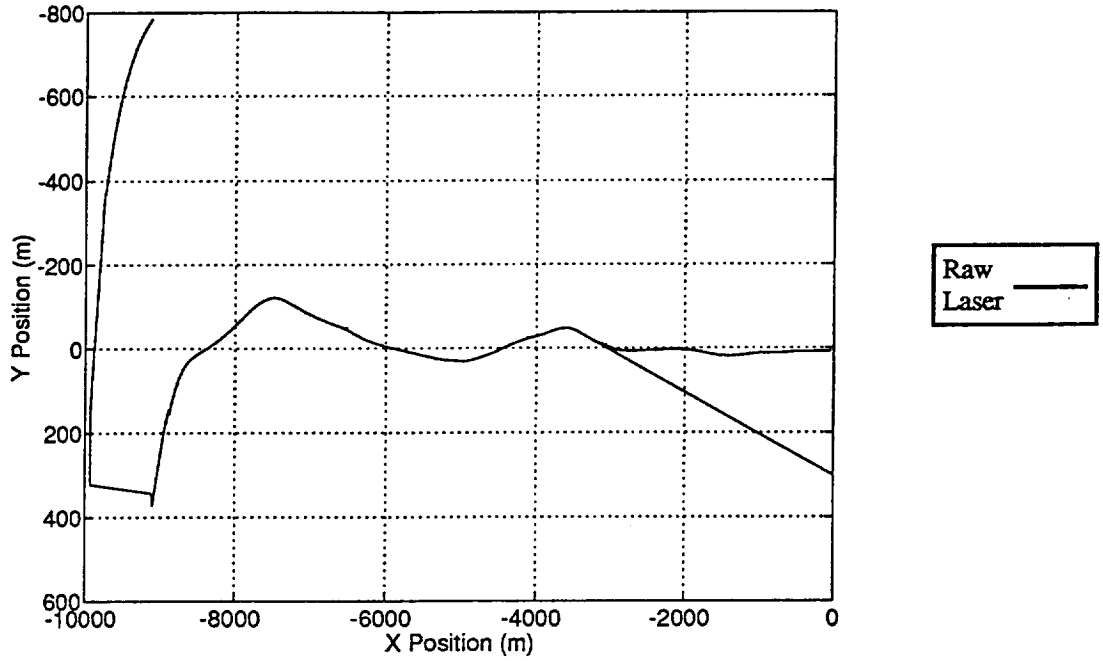


Figure 3.6 Sample Laser Ground Track Profile with Data Drop-out

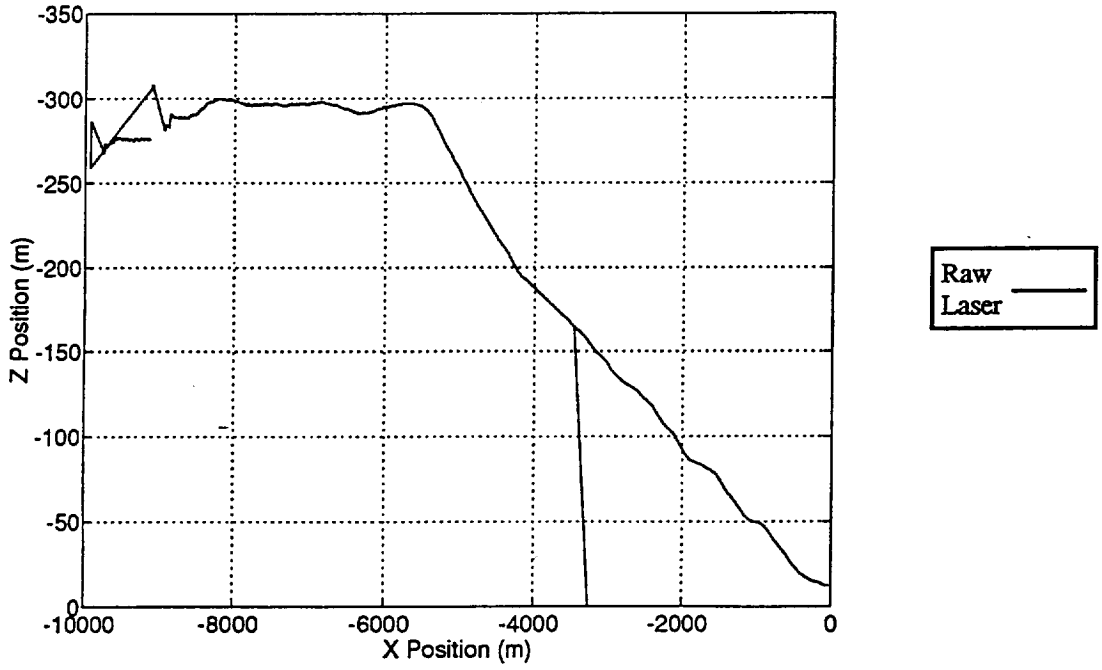


Figure 3.7 Sample Laser Vertical Descent Profile with Data Drop-out

Both figures show a substantial amount of laser data drop-out between -10000 and -9000 meters in the X position. The plot in Figure 3.6 should have made a smooth continuous loop joining the two end points (between 300-500 m in the Y axis) instead of the discontinuous jump. Figure 3.7 shows the corresponding vertical discontinuity as a jagged spike (between -250 and -310 m). The figures also illustrate sporadic data captured by the laser tracker. At about -3500 m (X axis, Figure 3.6), a major spike can be seen. The data spike deviates significantly from the zero Y position in the ground track profile.

The vertical spike is obvious, showing a rapid drop instead of a continuous descent profile. This spike was rejected as a sporadic data point. Figure 3.8 plots a blown up segment of Figure 3.6, showing the small dynamic maneuver experienced during this flight. Note the noise (small spikes) characteristics recorded by this laser system. This segment of the run met all three of the data selection criteria mentioned previously and was analyzed.

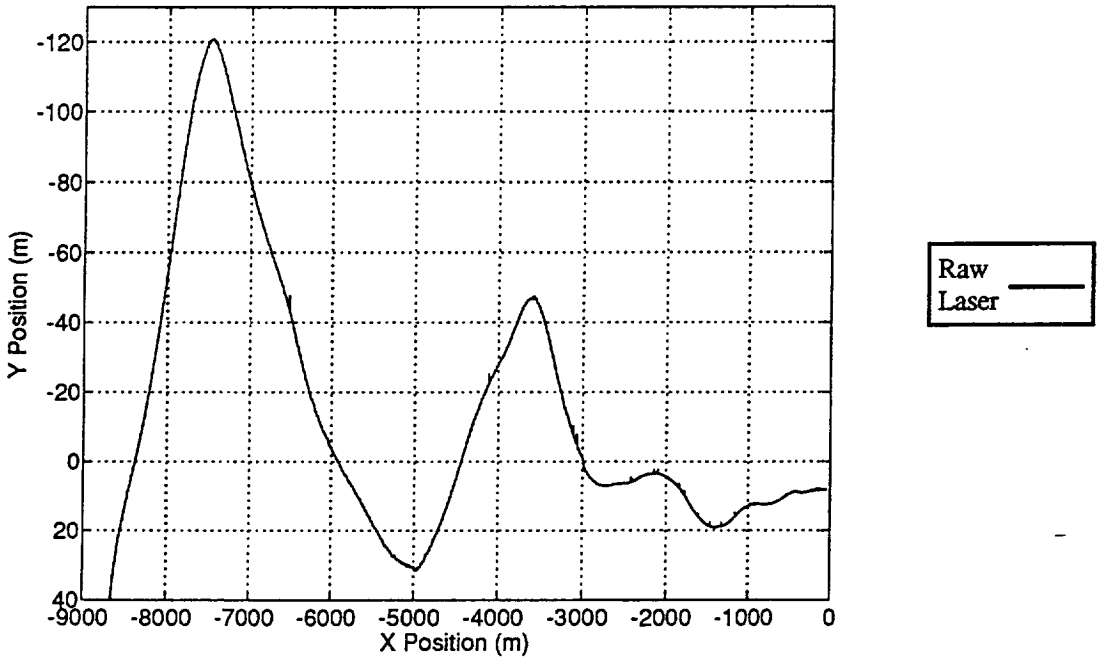


Figure 3.8 Sample Laser Ground Track Profile Showing Small Dynamic Maneuvers

CHAPTER 4

OVERVIEW AND DESIGN OF THE KALMAN FILTER

Kalman Filter Theory

The Kalman filter is one of many methods available for discrete time analysis and a standard technique used to process data with noise. It allows measurements to be processed in real-time. The filter can also integrate a redundancy of measurement data from a variety of sources. In addition, it carries an estimate of the measurements with every measurement update, allowing for the rejection of sporadic measurement data. The filter allows for the integration and update of non-synchronous measurements between the different sources.

The Kalman filter used in this thesis is similar in design and notation to that of Brown & Hwang [10]. The Kalman filter is comprised of two components; the State and Measurement models. They are defined respectively below in Equations 4.1 and 4.2.

$$\mathbf{x}_{k+1} = \Phi_k \mathbf{x}_k + \mathbf{w}_k \tag{4.1}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{4.2}$$

where:

- $\mathbf{x}_k$ = (n x 1) process state vector at time t_k .
- Φ_k = (n x n) matrix relating $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ in the absence of a forcing function. (If $\mathbf{x}_k$ is a sample of continuous process, Φ_k is the state transition matrix.)
- $\mathbf{w}_k$ = (n x 1) a vector assumed to be white (uncorrelated) sequence with know covariance structure.
- $\mathbf{z}_k$ = (m x 1) vector measurement at time t_k .
- $\mathbf{H}_k$ = (m x n) matrix giving ideal noiseless relation between the measurement and state vector at time t_k .
- $\mathbf{v}_k$ = (m x 1) measurement error, assumed to be white sequence with known covariance structure and uncorrelated with the $\mathbf{w}_k$ sequence.

The covariance matrices for $\mathbf{w}_k$ and $\mathbf{v}_k$ vectors, are given in Equations 4.3 - 4.5. $\mathbf{Q}_k$ is the covariance structure of $\mathbf{w}_k$ (state) and $\mathbf{R}_k$ is the covariance structure for $\mathbf{v}_k$ (measurement). The values for $\mathbf{Q}_k$ and $\mathbf{R}_k$ are usually supplied by the hardware manufacturer or are assumed. The subscript "k" represents current time.

$$E[\mathbf{w}_k \mathbf{w}_i^T] = \begin{cases} \mathbf{Q}_k , & i = k \\ 0 , & i \neq k \end{cases} \tag{4.3}$$

$$E[\mathbf{v}_k \mathbf{v}_i^T] = \begin{cases} \mathbf{R}_k , & i = k \\ 0 , & i \neq k \end{cases} \tag{4.4}$$

$$E[\mathbf{w}_k \mathbf{v}_i^T] = 0 , \quad \text{for all } k \text{ and } i \tag{4.5}$$

The Kalman filter has four basic steps (see below). The first step calculates the Kalman gain matrix, $\mathbf{K}_k$, the matrix that minimizes the mean square estimation error. In Step 2, the state measurement estimate ($\hat{\mathbf{x}}_k$) is updated with the calculated $\mathbf{K}_k$. The covariance matrix, ($\mathbf{P}_k$) associated with the optimal estimate can now be computed in Step 3. In Step 4, the state and covariance matrix are both projected ahead ($k+1$) and fed back into Step 1 with the next available update. A fifth step, Step M is a modified procedure incorporated in this algorithm to reject sporadic data.

Step 1: Calculate Kalman gain.

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1}$$

Step 2: Update state measurement estimate.

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-)$$

Step 3: Update error covariance.

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^-$$

Step 4: Project state and covariance ahead.

$$\hat{\mathbf{x}}_{k+1}^- = \Phi_k \hat{\mathbf{x}}_k$$

$$\mathbf{P}_{k+1}^- = \Phi_k \mathbf{P}_k \Phi_k^T + \mathbf{Q}_k$$

Step M: Erroneous positional data rejection.

$$\left| \mathbf{z}_k - \hat{\mathbf{x}}_k^- \right| > 30 \text{ m}$$

A figure borrowed from Brown & Hwang [10] and duplicated in Figure 4.1 shows how the Kalman filter loop is formed. Note that the Kalman loop is initialized with prior state and error covariance estimates.

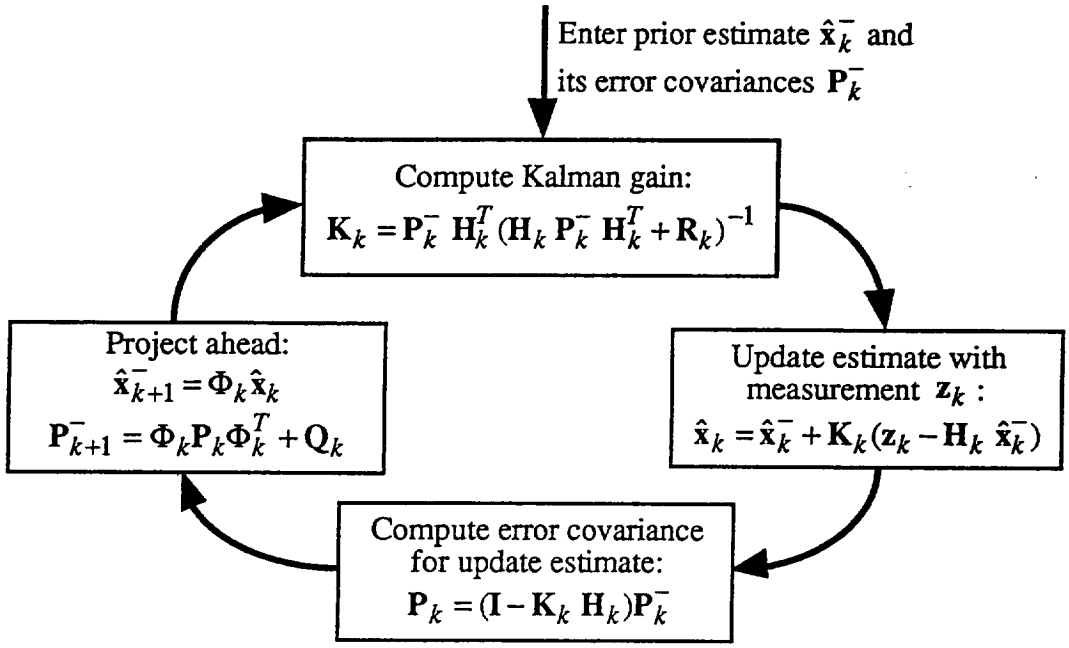


Figure 4.1 Kalman Filter Loop

While Figure 4.1 represents the basic Kalman filter loop, the dashed box in Figure 4.2 signifies the modification made to the basic Kalman filter. In the basic configuration, output from Step 1, (S1) goes directly into Step 2 (S2) for processing.

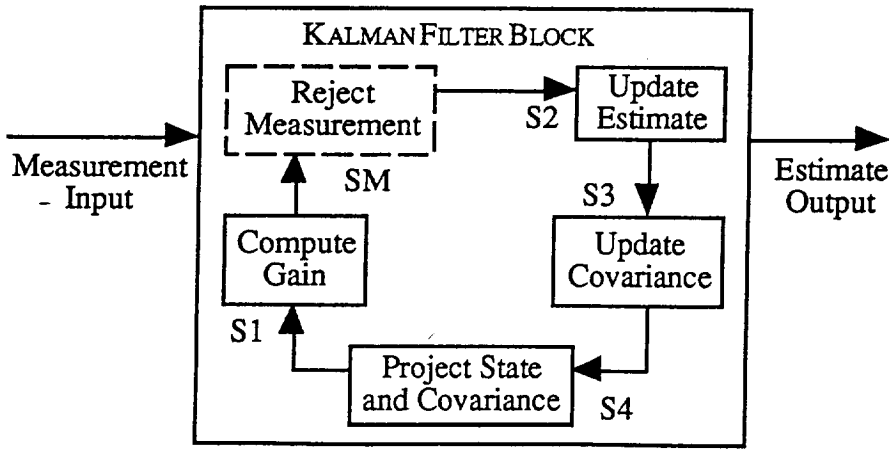


Figure 4.2 Modified Kalman Logical Loop

The criteria for positional measurement rejection is straight forward. If the difference between the measurement and estimated Kalman positional value is greater than a fixed value (30 meters, Step M), the measurement is rejected and the filter is updated with the estimated Kalman value. An incremental difference greater than 30 meters is sufficient to reject the data during an approach to landing maneuver. Highly dynamic maneuvers are not expected. Thus data not rejected via this criteria are processed by the filter. The 30 meter cut off value was heuristically derived. It is important not to make the rejection value too small, since this has the possibility of rejecting good but noisy data. This simple rejection criteria reduces computation time.

The INS velocity profile was assumed to vary slowly relative to the 64 Hz sample rate. Therefore, higher order dynamics are not necessary in the Kalman filter implementation. This reduces the number of states. The Kalman state equations were then decoupled into individual axis. By treating the axis independently, the algorithm was easier to code and modify. Execution time for each axis was reduced by solving a simple 3-by-3 matrix inversion. A graphical representation showing the integration of INS and DGPS information by the Kalman filter is depicted in Figure 4.3. The diagram represents data being processed for one axis. Note that position and velocity outputs from the Kalman filter are both at 64 Hz update rate.

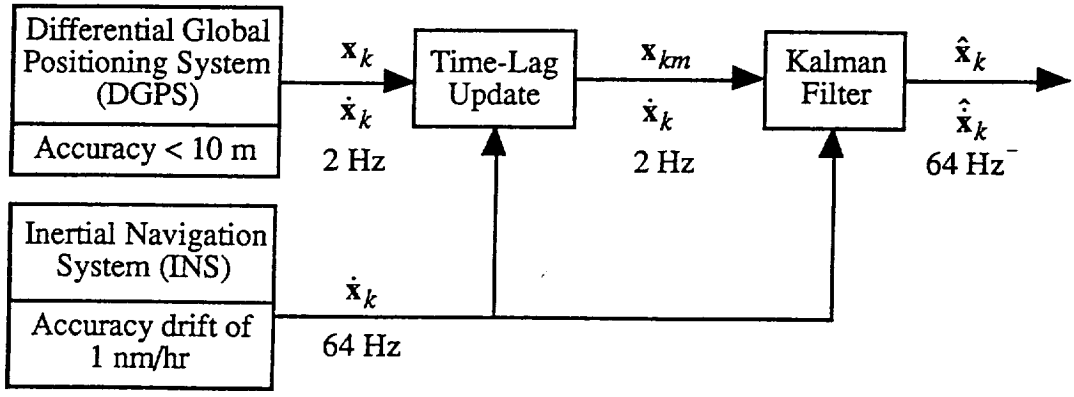


Figure 4.3 System Block Diagram of the Navigation Filter

The DGPS system is accurate to within 10 meters while the INS system has an accuracy drift of one nautical miles per hour. This DGPS unit supplies both position and derived velocity data at 2 Hz. INS velocity is supplied at 64 Hz. This particular DGPS unit had an inherent time-lag of 0.494 second which was incorporated in the algorithm to provide the Kalman filter with the most accurate measurement reading. Incorporation of the time-lag update (x_{km}) made a significant improvement in the performance of the algorithm.

The Navigation filter corrected this time-lag by updating the modified DGPS position with current DGPS position plus an averaged, integrated INS velocity calculated since the last update. The time-lag update formula is shown in Equation 4.6.

$$X_{km} = X_{gps} + \frac{\Sigma V_{ins}}{N} \Delta t_{gps} \tag{4.6}$$

$$\Delta t_{gps} = 0.494 \text{ seconds}$$

While Figure 4.3 pictured an example of data being processed for one axis, Figure 4.4 shows the integration of all three axes in the algorithm flow diagram. The data initialization and data rejection routine are also included in this algorithm flow diagram.

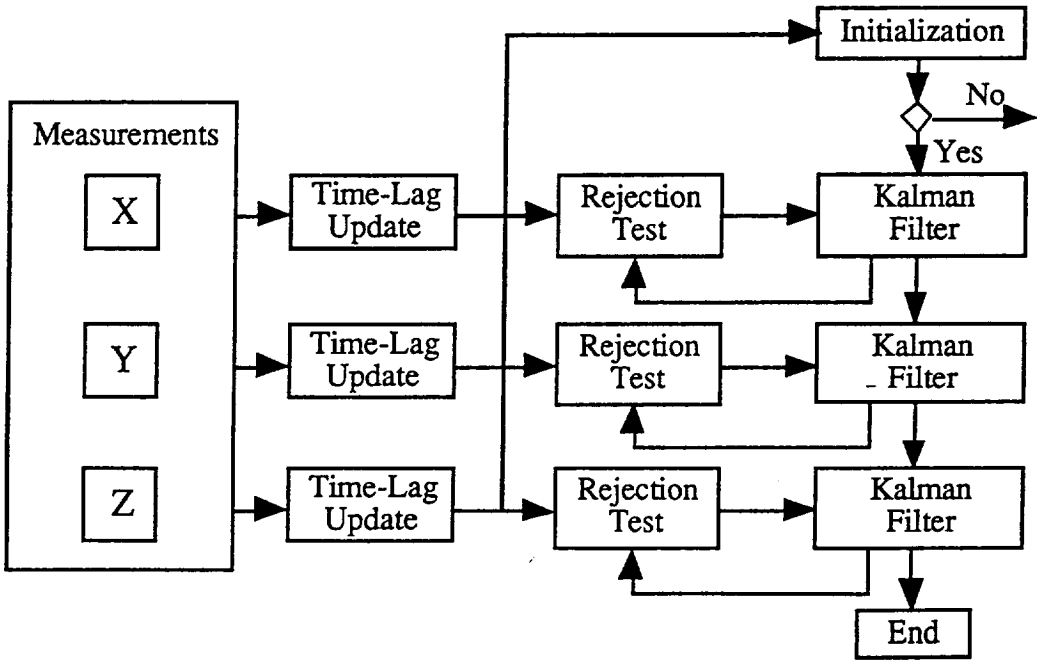


Figure 4.4 Algorithm Flow Diagram

Note that initialization occurred only once, at the beginning of the run. Data rejection testing begins with the next available measurement reading and before every measurement update.

Design of the Navigation Filter

The Navigation algorithm blends INS with DGPS measurements to produce navigational data. Simplicity in the design of the Navigation Kalman filter is a prime goal. This resulted in a nine state Kalman filter model, three states for each axis. By decoupling the equations (axis), the algorithm solves the three sets of filters sequentially. Input data include INS velocity which is derived from accelerometers, DGPS position and DGPS velocity derived from differential positions. Table 4.1 describes the state and measurement model used in the analysis of the Navigation filter. All measurements are in meters.

Table 4.1 The Nine State Navigation Filter

State Model	Measurement Model
$\mathbf{x}_{k+1} = \Phi_k \mathbf{x}_k + \mathbf{w}_k$	$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$
where $\mathbf{x}_k = \begin{bmatrix} \text{Position} \\ \text{Velocity} \\ \text{Velocity Bias} \end{bmatrix}$ $\text{Position} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$, $\text{Velocity} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix}$, $\text{Velocity Bias} = \begin{bmatrix} \dot{X}_{ins} - \dot{X}_{gps} \\ \dot{Y}_{ins} - \dot{Y}_{gps} \\ \dot{Z}_{ins} - \dot{Z}_{gps} \end{bmatrix}$	$\mathbf{z}_k = \begin{bmatrix} X_{\text{gps}} \\ \dot{X}_{\text{ins}} \\ \dot{X}_{\text{ins}} - \dot{X}_{\text{gps}} \end{bmatrix}$
$E(\mathbf{w}_k \mathbf{w}_k^T) = \begin{bmatrix} (4)^2 & 0 & 0 \\ 0 & (0.31)^2 & 0 \\ 0 & 0 & (1.52)^2 \end{bmatrix}$	$E(\mathbf{v}_k \mathbf{v}_k^T) = \begin{bmatrix} (0.31)^2 & 0 & 0 \\ 0 & (0.30)^2 & 0 \\ 0 & 0 & (0.91)^2 \end{bmatrix}$
$\Phi_k = \begin{bmatrix} 1 & \Delta t & \Delta t \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\Delta t = 1/64$ sec	$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

The state vector, $\mathbf{x}_k$ includes the position, velocity, and velocity bias states. The incorporation of the velocity bias state was an attempt to reduce the velocity error due to drift and to control the performance of the barometric altimeter damped vertical axis. The Φ_k matrix shows that position is updated by both the INS velocity and the velocity bias states. Note that Δt is 64th of a second, the INS data rate. As each DGPS and INS data are captured, the states are computed and projected ahead via the Φ_k matrix. Values for the covariance matrices $E(\mathbf{w}_k\mathbf{w}^T_k)$ and $E(\mathbf{v}_k\mathbf{v}^T_k)$ are estimated from empirical data or provided by manufacturer specifications. The $\mathbf{z}_k$ vector captures measurement data. For simplicity, the $\mathbf{H}$ matrix is represented by the identity matrix.

Design of the Truth Filter

The design of the Truth filter is very similar to that of the Navigation filter. The Truth filter integrates 64th Hz INS and 100 Hz laser data. The Truth filter synchronizes laser and INS measurement data and smoothes out laser data. The Truth filter is simpler in design than the Navigation filter. Laser position and INS velocity make up the state vector. Once again the axes are analyzed independently and are then sequentially processed. Descriptions for the six state Truth filter are presented in Table 4.2. Here, the Φ_k matrix updates position with only INS velocity.

Table 4.2 The Six State Truth Filter

State Model	Measurement Model
$\mathbf{x}_{k+1} = \Phi_k\mathbf{x}_k + \mathbf{w}_k$	$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$
$\mathbf{x}_k = \begin{bmatrix}\text{Position}\\ \text{Velocity}\end{bmatrix}$	$\mathbf{z}_k = \begin{bmatrix}x_{laser}\\ \dot{x}_{ins}\end{bmatrix}$
$E(\mathbf{w}_k\mathbf{w}_k^T) = \begin{bmatrix}(0.15)^2 & 0\\ 0 & (0.31)^2\end{bmatrix}$	$E(\mathbf{v}_k\mathbf{v}_k^T) = \begin{bmatrix}(0.30)^2 & 0\\ 0 & (0.31)^2\end{bmatrix}$
$\Phi_k = \begin{bmatrix}1 & \Delta t\\ 0 & 1\end{bmatrix}$ $\Delta t = 1/64$ sec	$\mathbf{H}_k = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$

CHAPTER 5

NAVIGATION ALGORITHM RESULTS

The Navigation Kalman filter integration of DGPS and INS data was conducted for the sample 3, 6, and 9 degree flights. The positional output of the navigation filter was compared with the laser tracker derived Truth data. The result of the sample 6 degree flight profile is plotted in Figures 5.1 and 5.2. These figures show a portion of the level and descent segments of flight. The ground track and vertical descent figures are representative of the other 3 and 9 degree approaches. The high correlation between the longitudinal and lateral positions characterizes the ground track profile. Truth data is represented by a solid line while Navigation data is represented by a dashed line.

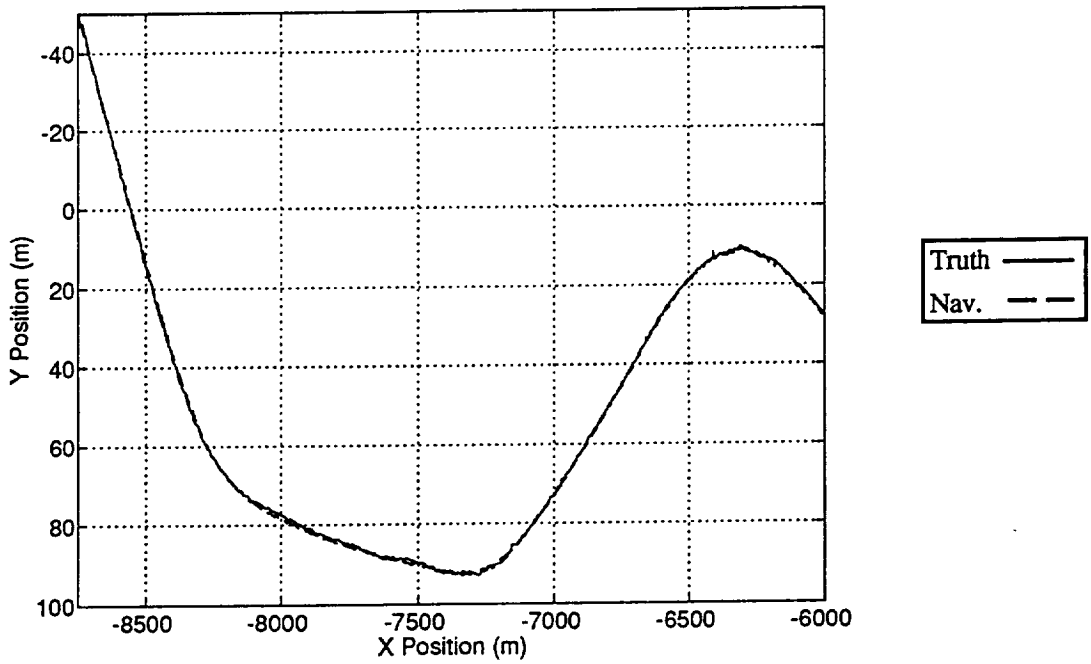


Figure 5.1 Representative Navigation and Truth Ground Track Profile

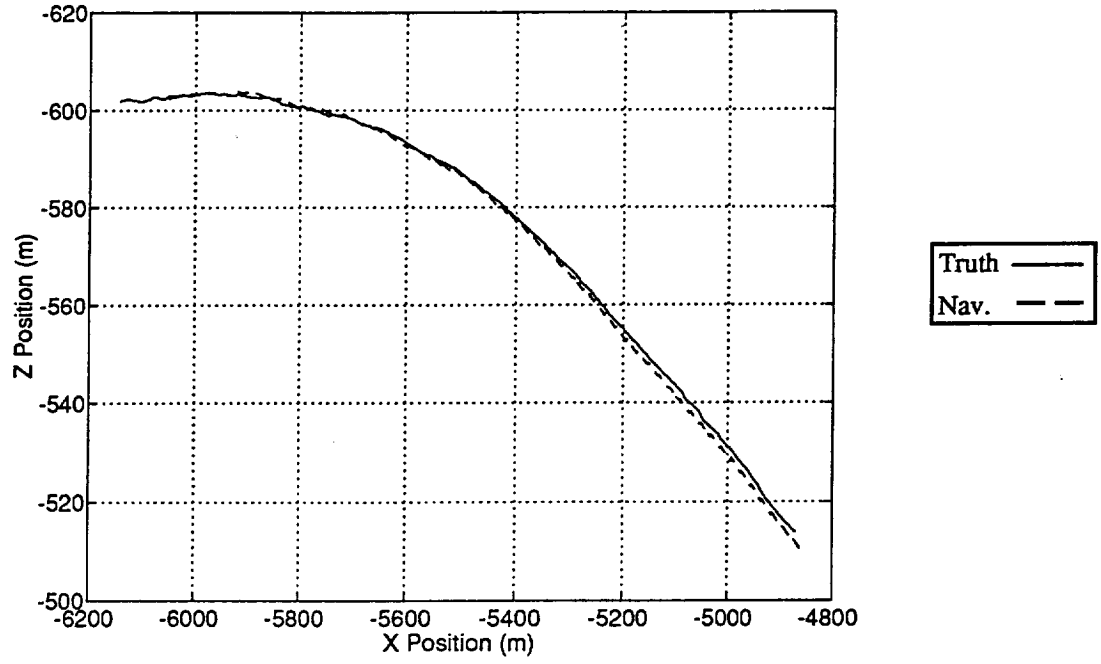


Figure 5.2 Representative Navigation and Truth Vertical Descent Profile

A characteristic of the vertical descent profile is the small bias error between the Navigation and Truth data. The vertical bias error is attributed to the general characteristic of the GPS system where the vertical axis is the least accurate [7].

The 3 degree positional histories of the difference (Delta) between the Navigation and Truth filters are shown in Figures 5.3-5.5. The figures points out a common trend in all three axes. At the initialization of the filters, errors greater than 4 m were calculated. During the initialization period, the filters were trying to converge the errors by producing a more accurate state estimator. As the system gradually converged, the errors stabilized to ± 3 meters of its mean value. On average, the system stabilized after 30-40 seconds. The error reduction in two sigma standard deviation is significant after convergence. This trend is also consistent with the 6 and 9 degree error differences.

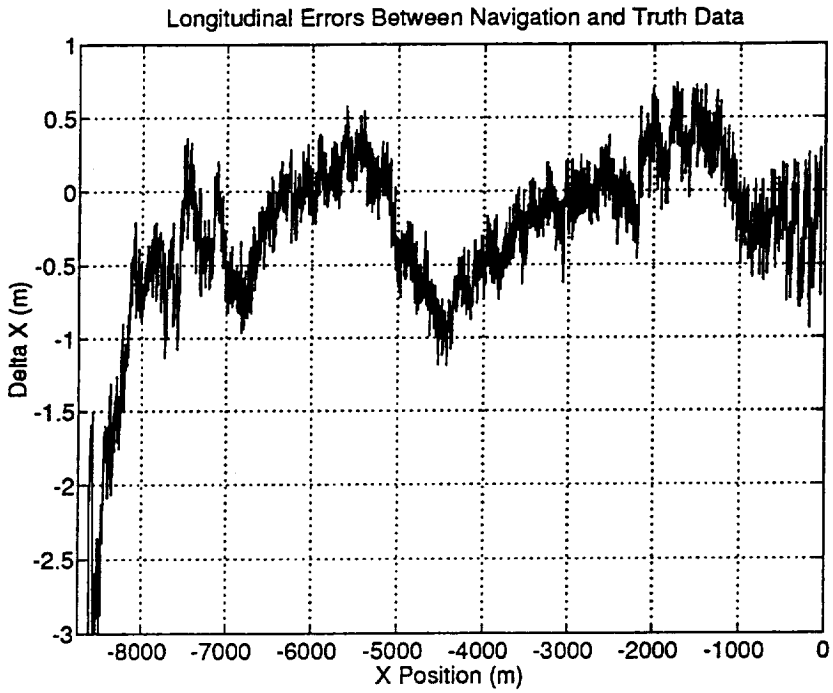


Figure 5.3 3° Longitudinal Errors

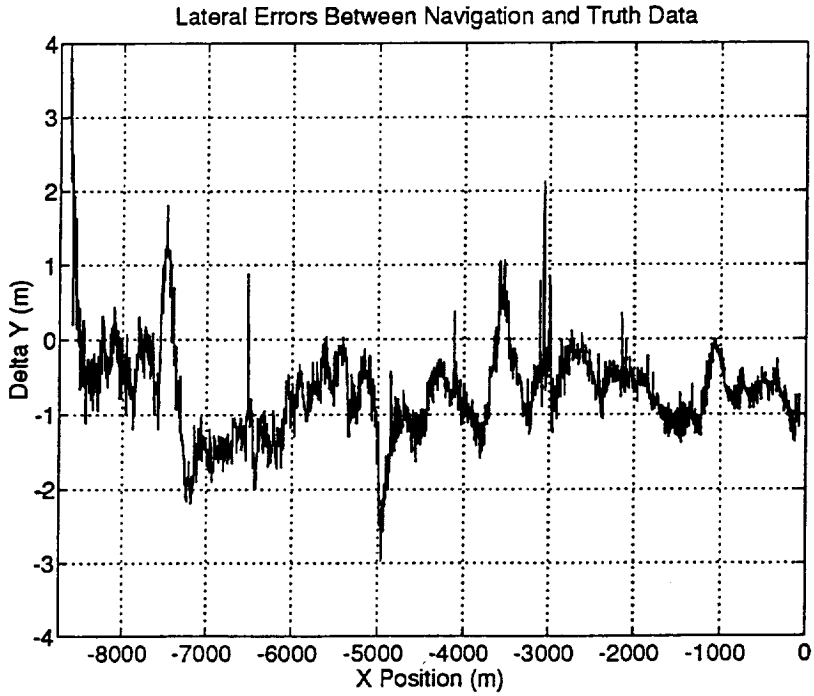


Figure 5.4 3° Lateral Errors

The navigation solution shows a higher variance at X positions less than -1000m, for the longitudinal and vertical errors. Within this range, the helicopter is arresting its vertical rate of descent and leveling off near the Aim Point. The filter tries to incorporate this dynamic maneuver into its prediction of the next state estimate. The slight increase in error is expected since the state model did not incorporate an acceleration component. This is most apparent in the vertical axis of Figure 5.5.

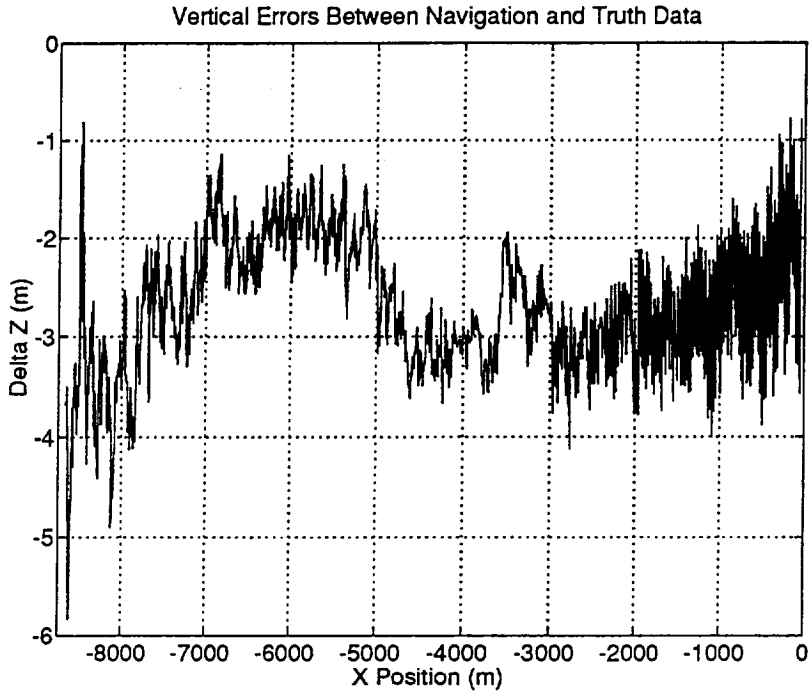


Figure 5.5 3° Vertical Errors

The lateral errors of Figure 5.4 converge from the initial spike at -8500 meter, like the longitudinal and vertical results. However, the figure shows a distinct spike occurring at -7500 meters with a shift in mean error of approximately one meter. At the corresponding X position in Figure 5.6, the aircraft is performing a level turn in the lateral plane. This maneuver can cause a possible ambiguity in laser reflector readings. Two more spikes occurring at -5000 and -3500 meters (in Figure 5.4) also correlate to the other level maneuvers of Figure 5.6.

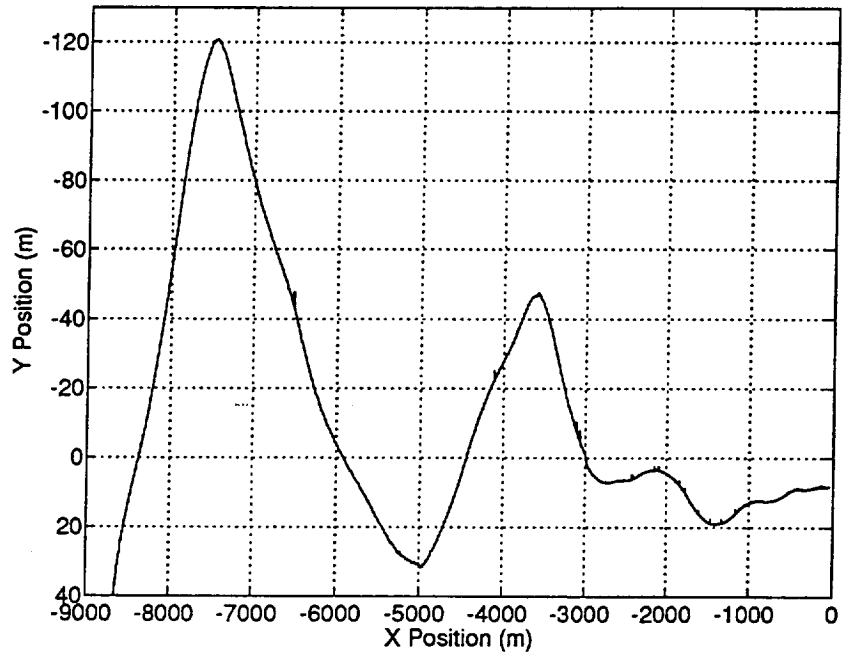


Figure 5.6 3° Ground Track Profile - Raw Laser Data

The statistical results for the sample 3, 6, and 9 degree flights are summarized in Figure 5.7. Figure 5.7 shows the mean and 2σ standard deviation of the differences between the Navigation and Truth data.

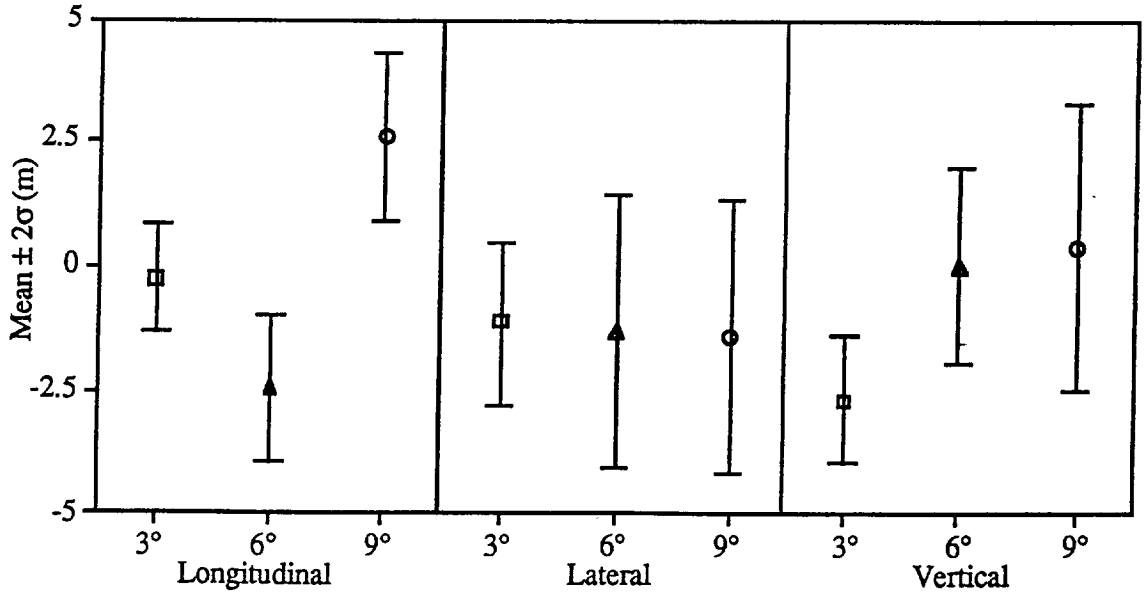


Figure 5.7 Statistical Results Showing Mean ± 2σ for 3°, 6° and 9° Flights

The statistical results are a culmination of both the level and dynamic descent portions of flight. In Figure 5.7, the 3 degree results show the smallest standard deviation spread. The 2σ spread widened slightly with steeper glideslope angle. The increase in spread can be attributed to the fact that during the 6 and 9 degree approaches, the aircraft was performing a higher dynamic maneuver. When comparing errors among the level portion of flight, the 2σ spread of the steeper approaches closely resembles that of the 3 degree profile. Overall, the results are well within RASCAL's 10 meter absolute positional requirement.

Table 5.1 compares the sample 3, 6, and 9 degree approaches with the corresponding time history flights from Kaufmann [4]. At first glance, the numbers indicate only a small difference between the two analytical methods. However, a direct comparison between the two results is difficult to make since the methods are fundamentally very different. But, the proximity of the results indicates that the Kalman filter can produce results within the same order of magnitude as the time history method.

Table 5.1 Navigation Filter and Time History Results - 3, 6, and 9 Degree Samples

Glideslope Angle / Flight Numbers	Axis	Navigation Filter Method*		Time History Method**	
		Mean (m)	2σ (m)	Mean (m)	2σ (m)
3 Degree	X	-0.27	± 1.08	-1.09	±1.71
3042-015 †	Y	-0.70	± 1.16	0.26	± 3.25
3092-314 ††	Z	-2.68	± 1.28	-2.80	± 2.25
6 Degree	X	-2.42	± 1.46	-4.98	± 1.83
3092-04 †	Y	-1.31	± 2.72	-0.47	± 1.06
3092-604 ††	Z	0.00	± 1.97	0.23	± 1.35
9 Degree	X	2.59	± 1.68	0.22	± 2.23
3092-018 †	Y	-1.42	± 2.75	0.13	± 1.36
3092-917 ††	Z	0.40	± 2.88	0.13	±1.51

† Current flight reference
†† Kaufmann's cross reference flight number
* Real-time analysis
** Post flight analysis

The fundamental differences between the two analytical methods are: real-time capability, time-lag update, and INS data update. Recall that the Navigation algorithm analyzes data in real-time and includes an embedded, time-lag advancement scheme. In contrast, the time history method advances the DGPS positions to match the laser positions during post flight analysis. Afterwards, the Irig B time-lag factor is calculated. Also, time history analysis uses no INS information. Without INS data, the time history method can not be implemented on a real-time dynamic airborne system. However, the time history results do show the best positional accuracy achievable with this DGPS system. These differences make a direct comparison between the two methods difficult.

CHAPTER 6

SUMMARY AND RECOMMENDATIONS

Summary

A real-time, high-rate precision Navigation algorithm has been developed and analyzed. The algorithm was designed to integrate time-lagged DGPS position and velocity data with high-rate INS velocity and attitude information via the Kalman filter. Result, the Navigation algorithm met and exceeded RASCAL's 10 meter absolute positional accuracy and 20 Hz update requirements. The algorithm demonstrated absolute precision navigation performance within 4.5 meters in all three axes and produced positional solution at a 64 Hz update rate.

The solutions surpassed RASCAL's positional requirement by 50% and the update rate by 200%. A relatively simple nine state Kalman filter model accomplished all this. Furthermore, the Kalman state matrix did not include an acceleration measurement model. The algorithm also accounted for errors caused by ambiguous and sporadic data readings. Additional number of states could be use to improve filter performance but doing so would compromise the simplicity of the algorithm design.

These results were achieved by strictly adhering to the data selection criteria. This data criteria allowed for testing of small dynamic maneuvers only. The logical extension would be to test the algorithm with more dynamic maneuvers. However, high dynamic maneuvers should be avoided since the algorithm was designed to provide navigation data during approach to landing only. This flight profile does not expect to experience such demanding maneuvers.

Recommendations

The algorithm used thus far has not been optimized for positional accuracy. Recall that the state matrix used only DGPS position, INS velocity, and velocity bias states. To increase positional accuracy, the velocity bias state may be replaced with an acceleration state or a better method may be employed on the velocity bias state itself. To check the robustness of the filter, less stringent data set should be tested. Different amount of data drop-outs should suffice. In addition, attention to detail during the data collection is crucial. This is especially true with Irig B time synchronization between the different data sets. Recalibration of the INU before each flight is also crucial in providing consistent data.

REFERENCES

1. van de Leijgraaf, R., Breeman, J., Moek, G., and van Leeuwen, S.S., "*A Position Reference System for the Fokker 70*," National Aerospace Laboratory, Amsterdam, The Netherlands.

2. McNally, B.D., Warner, D.N. Jr., Hegarty, D.M., Schultz, T.A., and Bronson, R., "*Flight Evaluation of Precision Code Differential GPS for Terminal Area Positioning*," Institute of Navigation Satellite Division's 4th International Technical Meeting (ION GPS-91), September 11-13, 1991.

3. McNally, B.D., Paielli, R.A., Bach, R.E., and Warner, D.N. Jr., "*Flight Evaluation of Differential GPS Aided Inertial Navigation Systems*," AGARD Guidance and Control Panel Specialist Meeting on Integrated and Multi-Function Navigation, Ottawa, Ontario, Canada, May 14-15, 1992.

4. Kaufmann, David N., "*Helicopter Approach Capability Using the Differential Global Positioning System*," NASA CR 177618, Moffett Field, NASA Ames Research Center, 1993

5. Edwards, F.G. and Hegarty, D.M., "*Flight Test Evaluation of Civil Helicopter Terminal Approach Operations Using Differential GPS*," AIAA 89-3635, AIAA Guidance, Navigation and Control Conference, Boston, MA, August 1989.

6. Edwards, F.G., and Loomis, P.V.W., "*Civil Helicopter Flight Operations Using Differential GPS*," Navigation: Journal of the Institute of Navigation, Vol. 32, No. 3, Fall 1985.

7. Edwards, F.G., Paielli, R.A. and Hegarty, D.M., "*Helicopter Terminal Approach Using Differential GPS with Vertical-Axis Enhancement*," Satellite Division Meeting of the Institute of Navigation, Colorado Springs, CO September 1987

8. Jacobson, R.A., Doane, D.H., Eshow, M.M., Aiken, E.W., and Hindson, W.S., "*An Integrated Rotorcraft Avionics / Controls Architecture to Support Advanced Controls and Low-Altitude Guidance Flight Research*," NASA TM-103983, Moffett Field, NASA Ames Research Center, October, 1992.

9. *MATLAB® High-Performance Numeric Computation and Visualization Software*, The MathWorks, Inc., Version 4.0, Natick, MA, 1992

10. Brown, R.G., and Hwang, P.Y.C. "*Introduction to Random Signals and Applied Kalman Filtering*," New York, John Wiley & Son, 1983.

APPENDIX A

NAVIGATION ALGORITHM WRITTEN IN MATLAB CODE

The following flights met all data set requirements and were analyzed by this algorithm:

3° : FSN-01901

6° : FSN-3092-011

9° : FSN-01904

```
% ##########################################################################
% #                                                                        #
% #   PROJECT: NAVIGATION KALMAN FILTER                                    #
% #            INTEGRATION OF DGPS & INS INFORMATION                       #
% #                                                                        #
% #                                                                        #
% #   WRITTEN BY: TY HOANG                                                 #
% #               AERONAUTICAL ENGINEERING DEPARTMENT                      #
% #               CAL POLY STATE UNIVERSITY - SAN LUIS OBISPO              #
% #               FEBRUARY 18 1994                                         #
% #                                                                        #
% #   In fulfillment of a Master's Thesis at CAL POLY                      #
% #                                                                        #
% #   CAL POLY ADVISOR:  DR. DANIEL J. BIEZAD,                             #
% #                      AERONAUTICAL ENGINEERING DEPARTMENT               #
% #   NASA AMES ADVISOR: MR. HARRY N. SWENSON,                             #
% #                      NAVIGATION AND CONTROLS BRANCH                    #
% #                                                                        #
% ##########################################################################
% ________________________________________________________________________
% Objective: To implement a Kalman filter integrating GPS and INS data, to
%   produce positional information. Output has GPS accuracy with high rate
%   of update, 64 Hertz.
%   INS data received at 64Hz and GPS at 2Hz

 file_KF_Navigation = ('KFNavigation.m');

% Assumptions:
%    Guessed Initial Value of: Q & R covariances
%    Initialized Values: x_prior, P_prior, tau, del_time
%    Constant Calculated Matrix: phi, H
%    Constant velocity during delta_time segment
%    First data point is not a wild point
% NOTE: insdata is in English Units
%       gpsdata is in SI Units

tic  % Engage time counter
clc  % clear command window
clear_on = input('Clear memory before starting script? << YES >> or [< NO >]
',','s');
 if
clear_on=='y'|clear_on=='Y'|clear_on=='yes'|clear_on=='YES'|clear_on=='Yes'
   clear
 end
```

```
 echo off
echo_on = input('Turn echo on?  << YES >> or [< NO >] ','s');
 if isempty(echo_on)==1|echo_on=='n'|echo_on=='N'|echo_on=='No'|echo_on=='NO'
   echo off
 else
   echo on
 end
% ____________________________________________________________________
% ###################### INITIALIZING CONSTANTS ########################
% Input Parameters
 gps_rate = 2 ;   fprintf('GPS data rate at  2 Hz.\n')  % In Hertz;
 ins_rate = 64;   fprintf('INS data rate at 64 Hz.\n\n')
 half = ins_rate/gps_rate;

% Setting Constants' initial values: Length(ft), Time(ms)
 del_time = 1/ins_rate;
 gps_time = 0.494;

 ft2m = 0.3048;          % Conversion from feet to meters
 in2ft = 1/12;           % Conversion from inches to feet

 cg_location   =[359.50; 0.00; 258.75]*in2ft*ft2m;  % CG depends on config.
 gps_location  =[761.00; 0.00; 334.00]*in2ft*ft2m;  % GPS antenna
 inu_location  =[298.75; 32.0; 212.50]*in2ft*ft2m;  % Actual INU location
 laser_location=[251.00; 56.0; 206.50]*in2ft*ft2m;  % Right Reflector

 % Coordinate GPS from tail boom to INU location;
 gps_inu = inu_location - gps_location;  % In feet & good sign convention
 % gps_inu = [298.75-761.00; 32.0-0.00; 212.50-334.00]*in2ft*ft2m;
 % gps_inu = [ - ; + ; - ];  % For proper sign convention when subtracted
% ____________________________________________________________________
% ###################### INITIALIZING INPUT FILES ######################
% Default raw data file formatted columns:
% Irig B Time, Roll, Pitch, Yaw, Platform Azimuth Angle, Vx, Vy, Vz.
% Or modified data file:
% Irig B Time, Vx, Vy, Vz

 fprintf('\nINS raw data file MUST contain ALL the following data \n')
 fprintf('(In column format)\n')
 fprintf('[Irig_B, Yaw, Planform Azimuth Angle, Vx_ins, Vy_ins, Vz_ins]\n\n')
 fprintf('NOTE: Columns do not have to be in this specific order\n\n')

ins_data_file = input('Enter INS data file: ','s');
while isempty(ins_data_file)==1
  ins_data_file = input('Enter INS data file: ','s');
end

 fprintf('\nLoading INS data.....\n\n\n')
 eval(['load ', ins_data_file]);
 eval(['INS = ', ins_data_file, ';']);
 [rins,cins] = size(INS);  % sizing row by column data

 fprintf('INS data is in Geodetic Coordinate System \n')
 fprintf('    (Longitude, Latitude, and Altitude)\n')
 fprintf('Time: milliseconds, Angle: degree, Velocity: feet/second  \n')
 fprintf('\nDEFAULT loads the following column format : \n')
```

```
 fprintf('[Irig_B, Yaw, Pitch, Roll, Azimuth Angle, Vx_ins, Vy_ins,
Vz_ins]\n')
 fprintf('NOTE: Current code uses only the following information \n')
 fprintf('      [Irig_B, Yaw, Azimuth Angle, Vx_ins, Vy_ins, Vz_ins] \n\n')
IC = input('Use DEFAULT [< YES >] or << NO >> : ','s');

 fprintf('\nConverting INS data into SI Units.....\n')

if IC=='n'|IC=='no'|IC=='N'|IC=='NO'
      IBT = input('Enter Column Number for "INS Irig B time"  :');
      IYW = input('Enter Column Number for "INS Yaw Angle"  :');
      IAA = input('Enter Column Number for "Azimuth Angle"  :');
      IVX = input('Enter Column Number for "INS X Velocity"  :');
      IVY = input('Enter Column Number for "INS Y Velocity"  :');
      IVZ = input('Enter Column Number for "INS Z Velocity"  :');
      raw_ins(:,1) = INS(:,IBT);
      raw_ins(:,2) = INS(:,IYW);
      raw_ins(:,3) = INS(:,IAA);
      raw_ins(:,4) = INS(:,IVX);
      raw_ins(:,5) = INS(:,IVY);
      raw_ins(:,6) = INS(:,IVZ);
end  % End IF IC Loop

 raw_ins = zeros(rins,6);

 if IC=='y'|IC=='Y'|IC=='yes'|IC=='YES'|isempty(IC)==1
   raw_ins(:,1) = INS(:,1);           % Irig
   raw_ins(:,2) = INS(:,2);           % Yaw did not use Pitch and Roll
   raw_ins(:,3) = INS(:,5);           % Azimuth Angle
   raw_ins(:,4:6) = INS(:,6:8);       % Vxins, Vyins, Vzins
 end  % End IF IC Loop

gps_data_file = input('Enter GPS data file: ','s');
while isempty(gps_data_file)==1
  gps_data_file = input('Enter GPS data file: ','s');
end

 fprintf('\nLoading GPS data.....\n\n')
 eval(['load ', gps_data_file]);
 eval(['GPS = ', gps_data_file, ';']);
 [rgps,cgps] = size(GPS);  % sizing row by column data

 fprintf('GPS data is in Earth Center Earth Fixed Coordinate System\n')
 fprintf('    (Xposition, Yposition, and Zposition)\n')
 fprintf('Time: milliseconds, Position: meters, Velocity: meters/second\n')
 fprintf('\nDEFAULT loads the following column format : \n')
 fprintf('[Irig_B, gps_time, Age, Xgps, Ygps,  Zgps, Vxgps, Vygps, Vzgps,
PDOP)\n')
 fprintf('NOTE: Current code uses only the following information \n')
 fprintf('      [Irig_B, Xgps, Ygps, Zgps, Vxgps, Vygps, Vzgps]\n\n')
GC = input('Use DEFAULT [< YES >] or << NO >> : ','s');

if GC=='n'|GC=='no'|GC=='N'|GC=='NO'
  GB = input('Enter Column Number for "GPS Irig B time"  :');
  GX = input('Enter Column Number for "GPS X Position"  :');
  GY = input('Enter Column Number for "GPS Y Position"  :');
  GZ = input('Enter Column Number for "GPS Z Position"  :');
```

```
   GU = input('Enter Column Number for "GPS X Velocity"  :');
   GV = input('Enter Column Number for "GPS Y Velocity"  :');
   GW = input('Enter Column Number for "GPS Z Velocity"  :');
   gps_raw(:,1) = (GPS(:,GB));
   gps_raw(:,2) = (GPS(:,GX));
   gps_raw(:,3) = (GPS(:,GY));
   gps_raw(:,4) = (GPS(:,GZ));
   gps_raw(:,5) = (GPS(:,GU));
   gps_raw(:,6) = (GPS(:,GV));
   gps_raw(:,7) = (GPS(:,GW));
end

 if GC=='y'|GC=='yes'|GC=='Y'|GC=='YES'|isempty(GC)==1
   gps_raw(:,1) = (GPS(:,1));          % Irig B did not use GPS time and Age
   gps_raw(:,2:7) = (GPS(:,4:9));
   % Did not use PDOP
 end

 [rgps_raw,cgps_raw] = size(gps_raw);

printme=input('Save Kalman, gps, covariance, or Bias files? [< YES >] or << NO
>> ','s');

 if printme=='y'|printme=='Y'|printme=='YES'|printme=='yes'|printme=='Yes'|
isempty(printme)==1
  savekal = input('Save Kalman output to file? [< YES >] or << NO >> ','s');
   if savekal=='y'|savekal=='Y'|savekal=='yes'|savekal=='YES'|savekal=='Yes'|
isempty(printme)==1
    kalout = input('Enter name for Kalman filter output file: ','s');
    while isempty(kalout)==1
      kalout = input('Enter name for Kalman filter output file: ','s');
    end
     KALout = 1;
   else
     KALout = 0;
   end

  savegps = input('Save GPS output to file? [< YES >] or << NO >> ','s');
   if
savegps=='y'|savegps=='Y'|savegps=='yes'|savegps=='YES'|savegps=='Yes'|isempty
(savegps)==1
    gpsout = input('Enter name for transformed GPS file: ','s');
    while isempty(gpsout)==1
      gpsout = input('Enter name for transformed GPS file: ','s');
    end
     GPSout = 1;
   else
     GPSout = 0;
   end

  savecov = input('Save Covariances to file? [< YES >] or << NO >> ','s');
   if
savecov=='y'|savecov=='Y'|savecov=='yes'|savecov=='YES'|savecov=='Yes'|isempty
(savecov)==1
    covout = input('Enter name for Covariance output file: ','s');
    while isempty(covout)==1
      covout = input('Enter name for Covariance output file: ','s');
```

```
    end
     COVout = 1;
   else
     COVout = 0;
   end

  savebias = input('Save Velocity bias to file? << YES >> or [< NO >] ','s');
   if savebias=='n'|savebias=='N'|savebias=='no'|savebias=='NO'|
isempty(savebias)==1
     BIASout = 0;
   else
     BIASout = 1;
     biasout = input('Enter name for Bias output file: ','s');
     while isempty(biasout)==1
       biasout = input('Enter name for Bias output file: ','s');
     end
   end  % End IF savebias Loop
 end  % End IF printme Loop

 if printme=='n'|printme=='N'|printme=='no'|printme=='NO'|printme=='No'
   fprintf('\nGgeneric output files have been created: kal_out and gps_out\n')
   eval(['!rm cov_out kal_out gps_out'])
   kalout = 'kal_out';
   gpsout = 'gps_out';
   covout = 'cov_out';
   KALout = 1;
   GPSout = 1;
   COVout = 1;
   printme = 'y';
 end
% ____________________________________________________________________
% ################### COORDINATE TRANSFORMATIONS  ########################
% Coordinate Transform of INS from VCV (Geodetic) to RCS
% Variable convention:
% C_nwu_ins = transformation matrix from ins to nwu system
% Conversion from V_ins to V_rcs:
%   V_rcs = C_rcs_pwu * C_pwu_nwu * C_nwu_ins * V_ins;
%   rcs = runway coordinate system (parallel runway, right, down)
%   pwu = Parallel runway, West(left), Up system
%   nwu = North, West, Up system
%   ins = inertial navigation system
%
   wander_angle = 10.099;          % Degrees, wander angle
   phi = wander_angle/180.0*pi;  % Angle between True North and X_rcs(rad)

% Read in ins data:
% First Transformation:  From ins to nwu: C_nwu_ins * V_ins = V_nwu
% Second Transformation: From nwu to pwu: C_pwu_nwu * V_nwu = V_pwu
% Third Transformation:  From pwu to rcs: C_rcs_pwu * V_pwu = V_rcs
% Translate than Rotate Coordinates
% Final Translation is from the GPS to INU location of aircraft.
% gps_cg = gps_location - cg_location

  prme = input('SAVE Coordinate Transform of Vins to Vrcs ? << YES >> or [< NO
>]  ','s');
   if isempty(prme) == 1
     prme == 'no';
```

```
    end
    if prme=='y'|prme=='Y'|prme=='YES'|prme=='yes'|prme=='Yes'
      inout=input('Enter name for transformed Vins-Vrcs output file: ','s');
      while isempty(inout)==1
        inout=input('Enter name for transformed Vins-Vrcs output file: ','s');
      end
    end

    % Preallocate of memory location to increase processing speed
    % Zeros(row,columns); 6 because of six column output.
    insdata = zeros(rins,6);

    C_pwu_nwu = [cos(phi),-sin(phi),0; sin(phi),cos(phi),0; 0,0,1];
    C_rcs_pwu = [1,0,0; 0,-1,0; 0,0,-1];
    fprintf('\nPerforming Coordinate Transform of INS from Local Level to RCS
system\n')

    insdata(:,1) = raw_ins(:,1);  % Needs no transformation for time
    for i = 1:rins
      % alpha = azimuth angle - true heading: (in radians)
      alpha = raw_ins(i,3) - raw_ins(i,2);
      C_nwu_ins=[cos(alpha),-sin(alpha),0; sin(alpha),cos(alpha),0; 0,0,1];

V_rcs=C_rcs_pwu*C_pwu_nwu*C_nwu_ins*[raw_ins(i,4);raw_ins(i,5);raw_ins(i,6)];
      insdata(i,2:4) = V_rcs' .* ft2m;
      if prme=='y'|prme=='Y'|prme=='YES'|prme=='yes'|prme=='Yes'
        fprintf(inout,'%8.0f   ',insdata(i,1) )
        fprintf(inout,'%10.5f  ',insdata(i,2:3) )
        fprintf(inout,'%10.5f\n',insdata(i,4) )
      end  % End IF prme Loop
    end  % End For i Loop

% Coordinate Transform of GPS from ECEF to RCS
% Variable convention:
% C_r_e = transformation matrix from ecef to rcs system
% C_r_e = C_r_v * C_v_e
%     r = rcs, runway coordinates
%     e = earth center earth fixed
%     v = vehicle carried vertical (true north)
%
 a = 6378137.0;          % Semi-major axis of earth ellipsoid (6378137 m)
 b = 6356752.3141;       % Semi-minor axis of earth ellipsoid (m)
 h = 12.4;               % Geodetic height of Runway 35 Aim Point (m)
 e = sqrt((2*(a-b)/a)-((a-b)^2/(a^2)));  % Eccentricity of earth ellipsoid
 SI = 37.41335361/180.0*pi;  % Geodetic latitude of Runway Aim Point (rad)
 LM = -121.1082725/180.0*pi;  % Geodetic longitude of Runway Aim Point (rad)
 N = a/(sqrt(1-(e*e*sin(SI)*sin(SI))));  % Radius of curvature of ellipsoid

% AP_ecef is the relationship between geodetic & ECEF at Crows Landing, now in
SI units
 AP_ecef=[(N+h)*cos(SI)*cos(LM); (N+h)*cos(SI)*sin(LM); (N*(1-
e*e)+h)*sin(SI)];

 C_v_e=[-sin(SI)*cos(LM),-sin(SI)*sin(LM),cos(SI);-sin(LM),cos(LM),0;-
cos(SI)*cos(LM),-cos(SI)*sin(LM),-sin(SI)];
 C_r_v = [cos(phi), sin(phi), 0; -sin(phi), cos(phi), 0; 0, 0, 1];
 C_r_e = C_r_v * C_v_e;
```

```
fprintf('\nPerforming Coordinate Transform of GPS from VCV to RCS system\n')

gpsdata = zeros(rgps,7);  % 7 for 7 columns output style.
gpsdata(:,1) = gps_raw(:,1);  % Needs no transformation for time

for j = 1:rgps
  del_wgs = [gps_raw(j,2);gps_raw(j,3);gps_raw(j,4)]-AP_ecef;
  gps_rcs = C_r_e * del_wgs;          % Now gpsdata is in rcs frame
  gpsV_rcs = C_r_e * [gps_raw(j,5); gps_raw(j,6); gps_raw(j,7)];
  gpsdata(j,2:4) = gps_rcs';          % GPS positions
  gpsdata(j,5:7) = gpsV_rcs(1:3,1)';  % Vxrcs, Vyrcs, Vzrcs

  if GPSout == 1
    fprintf(gpsout,'%8.0f   ',gpsdata(j,1) )    % Irig B
    fprintf(gpsout,'%10.5f  ',gpsdata(j,2:6) )  % Xgps, Ygps, Zgps, Vx, Vy
    fprintf(gpsout,'%10.5f\n',gpsdata(j,7) )    % Vz
  end  % End IF GPSout Loop
end  % End FOR j Loop

% ____________________________________________________________________
% Transferring GPS at tail to INU location before entering Kalman filter
fprintf('And from GPS antenna to INU location.\n')
for b = 1:rgps
  gpsdata(b,2:4) = gpsdata(b,2:4) - gps_inu';
end

% ################## INITIALIZING KALMAN VARIABLES ######################
fprintf('\nInitalizing Kalman Variables now.....\n')
% Initializing Matrices              % format [11 12 13; 21 22 23; ...]
g = 10.0;
err = [g*3.0; g*3.0; g*3.0];         % Wild data, prcnt change from previous
fprintf('\nDATA Filter now at +/-%6.3f meters.\n',err)
Q = [(4.0)^2, 0, 0; 0, (1*ft2m)^2, 0; 0, 0, (5*ft2m)^2];
% Init Measure Cov, Q(1,1) is already in meters.
H = [1, 0, 0; 0, 1, 0; 0, 0, 1]; % H = Ideal noiseless matrix between z & x
R = [(1.0*ft2m)^2, 0, 0; 0, (3*ft2m)^2, 0; 0, 0, (3*ft2m)^2];  %
% Initial: State Covariance
IM = [1, 0, 0; 0, 1, 0; 0, 0, 1];  % Identity Matrix
Kx = [1, 0, 0; 0, 1, 0; 0, 0, 1];  % Initial Guess: Kalman Gain Value
Ky = [1, 0, 0; 0, 1, 0; 0, 0, 1];
Kz = [1, 0, 0; 0, 1, 0; 0, 0, 1];
Px_prior = [1, 1, 1; 1, 1, 1; 1, 1, 1];  % Prior estimate of Xerror P
Py_prior = [1, 1, 1; 1, 1, 1; 1, 1, 1];
Pz_prior = [1, 1, 1; 1, 1, 1; 1, 1, 1];
phi_Matrix = [1, del_time, del_time; 0, 1, 0; 0, 0, 1];

% Search for a common Irig B time before starting Kalman Filter Routine
ins_start = 2;  % Start INS at 2nd row of data
gps_start = 1;

for start = 1:rgps
if insdata(start,1)==gpsdata(gps_start,1)|insdata(start,1)
<gpsdata(gps_start,1)
  ins_start = start;
  break;
else
```

```
   gps_start = gps_start + 1;
 end  % End IF insdata... Loop
 end  % End FOR start Loop
 if start == rgps
   fprintf('\n\n !!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!\n')
   fprintf('\nINS and GPS Irig B times are not within the same time frame\n')
   fprintf('\n********************* Goshawk Terminated ***************\n')
   break; end;
 end

 % Prior estimate of X state = [Xtrue_position; Vtrue_velocity; V_bias]
  x_prior = [gpsdata(gps_start,2)+insdata(ins_start,2)*gps_time;
insdata(ins_start,2); (insdata(ins_start,2)-gpsdata(gps_start,5))];
  y_prior = [gpsdata(gps_start,3)+insdata(ins_start,3)*gps_time;
insdata(ins_start,3); (insdata(ins_start,3)-gpsdata(gps_start,6))];
  z_prior = [gpsdata(gps_start,4)+insdata(ins_start,4)*gps_time;
insdata(ins_start,4); (insdata(ins_start,4)-gpsdata(gps_start,7))];
%
% ##################### ENTERING KALMAN FILTER LOOP ######################
fprintf('\nEntering Kalman Filter Loop.....\n')
% Equations working with
% State:              X(k+1) = Phi(k) * X(k) + W(k)
% Measurement:          Z(k) = H(k) * X(k) + V(k)
% Where:                X(k) = [True_position; True_velocity]
%                       W(k) = [Qgps 0; 0 Qinu]
%                       Z(k) = [Position_gps; Velocity_inu]
%                       V(k) = [Rgps; Rinu]
%
% Kalman Gain:             K = P * H' * (H * P * H' + R)^(-1)
% Update Estimate:   X_upest = X_prior + K * (Z - H * X_prior)
% Error Covariance:  P_upest = (IM - K * H) * P_prior;
% Project Ahead:     X_prior = phi_Matrix * X_upest;
%                    P_prior = phi_Matrix * P_upest * phi_Matrix' + Q;
%
 m = 0;
 gps_x_total = 0;
 gps_y_total = 0;
 gps_z_total = 0;
 qc = gps_start;          % Counters for GPS data advancement
 gps_X_total = 0;
 gps_Y_total = 0;
 gps_Z_total = 0;         % Total # GPS ignored
 half_time = 1;
 Vx_sum = 0;
 Vy_sum = 0;
 Vz_sum = 0;              % Running total of gps-ins lag time
 X_average = 0;
 Y_average = 0;
 Z_average = 0;
 X_count = 1;
 Y_count = 1;
 Z_count = 1;
 X_skip_data = 2;
 Y_skip_data = 2;
 Z_skip_data = 2;
 Pins(1,1) = gpsdata(gps_start,2);   % Initial INS position
 Pins(2,1) = gpsdata(gps_start,3);
```

```
 Pins(3,1) = gpsdata(gps_start,4);
%
%_____________________ Entering Kalman Loop _____________________________
 K_kalm = zeros(0.8*rins,13);  % Output has 13 columns
 for p = ins_start:rins
   m = m + 1;
  % STEP ONE:  Compute Kalman Gain
   Kx = Px_prior * H' * (H * Px_prior * H' + R)^(-1);
   Ky = Py_prior * H' * (H * Py_prior * H' + R)^(-1);
   Kz = Pz_prior * H' * (H * Pz_prior * H' + R)^(-1);

  % Test to ensure not using wild data in next z measurement
   if half_time == half & qc < rgps
     qc = qc + 1;                          % Read next gps data or incoming data

    if abs(gpsdata(qc,2)+(Vx_sum/X_count+gpsdata(qc,5))/2*gps_time-
x_prior(1,1)) > err(1,1)
      fprintf('p = %8.0f, Irig = %8.0f :',p,insdata(p,1))
      fprintf('X Diff = %9.4f\n',abs(gpsdata(qc,2)+X_average-x_prior(1,1)))
      X_skip_data = 1;                     % Update ins measurements only
      gps_x_total = gps_x_total + 1;       % Keep track of total points deleted
    else
      X_skip_data = 2;                     % Status flag, good measurements
    end

    if abs(gpsdata(qc,3)+(Vy_sum/Y_count+gpsdata(qc,6))/2*gps_time-
y_prior(1,1)) > err(2,1)
      fprintf('p = %8.0f, Irig = %8.0f :',p,insdata(p,1))
      fprintf('Y Diff = %9.4f\n',abs(gpsdata(qc,3)+Y_average-y_prior(1,1)))
      Y_skip_data = 1;                     % Update ins measurements only
      gps_y_total = gps_y_total + 1;       % Keep track of total points deleted
    else
      Y_skip_data = 2;                     % Status flag, good measurements
    end

    if abs(gpsdata(qc,4)+(Vz_sum/Z_count+gpsdata(qc,7))/2*gps_time-
z_prior(1,1)) > err(3,1)
      fprintf('p = %8.0f, Irig = %8.0f :',p,insdata(p,1))
      fprintf('Z Diff = %9.4f\n',abs(gpsdata(qc,4)+Z_average-z_prior(1,1)))
      Z_skip_data = 1;                     % Update ins measurements only
      gps_z_total = gps_z_total + 1;       % Keep track of total points deleted
    else
      Z_skip_data = 2;                     % Status flag, good measurements
    end

    half_time = 1;                         % Reset ins/gps time sync
  else  % Else half_time Loop

    X_skip_data = 1;                       % Update with INS Position estimate
    Y_skip_data = 1;
    Z_skip_data = 1;
    half_time = half_time + 1;             % Incremental sync time counter
  end  % End IF half_time Loop

 % STEP TWO:  Update estimate with measurement z
 % Equation:  #_upest = #_prior + K# * (z# - H * #_prior)
```

```
    if X_skip_data == 2     % Update X and use good measurement data
      X_average = ( Vx_sum/X_count + gpsdata(qc,5) )/2 * gps_time;
      zx = [gpsdata(qc,2)+X_average; insdata(p,2); (insdata(p,2)-
 gpsdata(qc,5))];
      X_count = 0;
      Vx_sum = 0;
    else
      zx = [x_prior(1,1); insdata(p,2); x_prior(3,1)];
      X_count = X_count + 1;
      Vx_sum = Vx_sum + insdata(p,2);
    end
    x_upest = x_prior +  Kx * (zx - H * x_prior);

    if Y_skip_data == 2     % Update Y and use good measurement data
      Y_average = ( Vy_sum/Y_count + gpsdata(qc,6) )/2 * gps_time;
      zy = [gpsdata(qc,3)+Y_average; insdata(p,3); (insdata(p,3)-
 gpsdata(qc,6))];
      Y_count = 0;
      Vy_sum = 0;
    else
      zy = [y_prior(1,1); insdata(p,3); y_prior(3,1)];
      Y_count = Y_count + 1;
      Vy_sum = Vy_sum + insdata(p,3);
    end
    y_upest = y_prior +  Ky * (zy - H * y_prior);

    if Z_skip_data == 2     % Update Z and use good measurement data
      Z_average = ( Vz_sum/Z_count + gpsdata(qc,7) )/2 * gps_time;
      zz = [gpsdata(qc,4)+Z_average; insdata(p,4); (insdata(p,4)-
 gpsdata(qc,7))];
      Z_count = 0;
      Vz_sum = 0;
    else
      zz = [z_prior(1,1); insdata(p,4); z_prior(3,1)];
      Z_count = Z_count + 1;
      Vz_sum = Vz_sum + insdata(p,4);
    end
    z_upest = z_prior +  Kz * (zz - H * z_prior);

    K_kalm(1)     = insdata(p,1);  % Irig B
    K_kalm(2:4)   = x_upest';  % X_update, Vx_update, Vx_bias_update
    K_kalm(5:7)   = y_upest';  % Y_update, Vy_update, Vy_bias_update
    K_kalm(8:10)  = z_upest';  % Z_update, Vz_update, Vz_bias_update
    K_kalm(11:13) = Pins';
    Pins(1,1) = Pins(1,1) + insdata(p,2) * del_time;
    Pins(2,1) = Pins(2,1) + insdata(p,3) * del_time;
    Pins(3,1) = Pins(3,1) + insdata(p,4) * del_time;

  % STEP THREE:  Compute Error Covariance for Updated Estimate
    Px_upest = (IM - Kx * H) * Px_prior;
    Py_upest = (IM - Ky * H) * Py_prior;
    Pz_upest = (IM - Kz * H) * Pz_prior;

    if
printme=='y'|printme=='Y'|printme=='YES'|printme=='yes'|printme=='Yes'|isempty
(printme)==1
      if KALout == 1
```

```
        fprintf(kalout,'%-7.0f  ',K_kalm(1) )     % INS Irig B
        % X_ins, X_kalman, Y_ins, Y_kalman, Z_ins
        fprintf(kalout,'%-10.4f  ',K_kalm(11),K_kalm(2),K_kalm(12),K_kalm(5),
K_kalm(13))
        fprintf(kalout,'%-10.4f\n',K_kalm(8))  %Z_kalman
      end

      if COVout == 1
        fprintf(covout,'%-8.0f  ', K_kalm(1) ) % Irig B
        % Px_upest, Pvx_upest,  Py_upest
        fprintf(covout,'%-6.3f  ',Px_upest(1,1),Px_upest(2,2),Py_upest(1,1))
        % Pvy_upest, Pz_upest
        fprintf(covout,'%-6.3f  ',Py_upest(2,2),Pz_upest(1,1))
        fprintf(covout,'%-6.3f\n',Pz_upest(2,2))  % Pvz_upest
      end

      if BIASout == 1
        fprintf(biasout,' %-8.0f  ', K_kalm(1) ) % Irig B
        fprintf(biasout,' %-10.4f  ', zx(3,1), zy(3,1) )
        % Vx_bias_update, Vy_bias_update
        fprintf(biasout,' %-10.4f\n', zz(3,1) )     % Vz_bias_update
      end
    end  % End IF printme Loop

   % STEP FOUR:  Project Ahead
    x_prior = phi_Matrix * x_upest;
    y_prior = phi_Matrix * y_upest;
    z_prior = phi_Matrix * z_upest;
    Px_prior = phi_Matrix * Px_upest * phi_Matrix' + Q;
    Py_prior = phi_Matrix * Py_upest * phi_Matrix' + Q;
    Pz_prior = phi_Matrix * Pz_upest * phi_Matrix' + Q;
  end  % End For p Loop

  fprintf('\n')
  fprintf('Total number of GPS X data omitted:  %-5.0f points\n', gps_x_total)
  fprintf('Total number of GPS Y data omitted:  %-5.0f points\n', gps_y_total)
  fprintf('Total number of GPS Z data omitted:  %-5.0f points\n', gps_z_total)

echo off
toc  % Print time counter
end  % End of script
% ____________________________________________________________
% ############################## END OF SCRIPT ##############################
```

APPENDIX B

TRUTH ALGORITHM WRITTEN IN MATLAB CODE

The following flights met all data set requirements and were analyzed by this algorithm:

3° : FSN-01901

6° : FSN-3092-011

9° : FSN-01904

```
% ##########################################################################
% #                                                                        #
% #   PROJECT: TRUTH KALMAN FILTER                                         #
% #            INTEGRATION OF DGPS & LASER INFORMATION                     #
% #                                                                        #
% #                                                                        #
% #   WRITTEN BY: TY HOANG                                                 #
% #               AERONAUTICAL ENGINEERING DEPARTMENT                      #
% #               CAL POLY STATE UNIVERSITY - SAN LUIS OBISPO              #
% #               FEBRUARY 18 1994                                         #
% #                                                                        #
% #   In fulfillment of a Master's Thesis at CAL POLY                      #
% #                                                                        #
% #   CAL POLY ADVISOR:  DR. DANIEL J. BIEZAD,                             #
% #                      AERONAUTICAL ENGINEERING DEPARTMENT               #
% #   NASA AMES ADVISOR: MR. HARRY N. SWENSON,                             #
% #                      NAVIGATION AND CONTROLS BRANCH                    #
% #                                                                        #
% ##########################################################################
% ____________________________________________________________________________
% Objective: To implement a Kalman filter routine on laser and ins data, to
%   produce positional information. Output used as truth data and compared
%   to the positional information from the Navigation Kalman Filter.
%   INS data received at 64Hz and LASER at 100 Hz

 file_KF_Truth = ('KFTruth.m');

% Assumptions:
%    Guessed Initial Value of: Q & R covariances
%    Initialized Values: x_prior, P_prior, tau, del_time
%    Constant Calculated Matrix: phi, H
%    Constant velocity during flight and data gathering
%    First data point is not a wild point
% NOTE: insdata is in English Units
%       laser data is in SI Units

tic  % Engage time counter
clc  % clear command window
clear_on = input('Clear memory before starting script? << YES >> or [< NO >]
', 's');
 if
clear_on=='y'|clear_on=='Y'|clear_on=='yes'|clear_on=='YES'|clear_on=='Yes'
   clear
 end
```

```
 echo off
echo_on = input('Turn echo on?  << YES >> or [< NO >] ','s');
 if isempty(echo_on)==1|echo_on=='n'|echo_on=='N'|echo_on=='No'|echo_on=='NO'
   echo off
 else
   echo on
 end
% ____________________________________________________________________
% ######################## INITIALIZING CONSTANTS ########################
% Input Parameters
 laser_rate = 2 ;   % 2 Hertz to simulate gps update of 2 Hertz;
fprintf('LASER data rate of 100 Hz is captured at 2 Hz for INS update.\n')
 ins_rate = 64;
fprintf('INS data rate at 64 Hz.\n\n')
 half = ins_rate/laser_rate;

% Setting Constants' initial values: Length(ft), Time(ms)
 del_time = 1/ins_rate;

 ft2m = 0.3048;            % Conversion from feet to meters
 in2ft = 1/12;             % Conversion from inches to feet
 cg_location    = [359.50; 0.00; 258.75]*in2ft*ft2m;  % cg depends on config
 gps_location   = [761.00; 0.00; 334.00]*in2ft*ft2m;  % GPS antenna
 inu_location   = [298.75; 32.0; 212.50]*in2ft*ft2m;  % Actual INU location
 laser_location = [251.00; 56.0; 206.50]*in2ft*ft2m;  % Right Reflector

 % Coordinate Transform of LASER from right reflector to INU location
 laser_inu = inu_location - laser_location;
 % laser_inu = [298.75-251.00; 32.0-56.0; 212.50-206.50]*in2ft*ft2m;  % RHS
 % laser_inu = [ + ; - ; + ];  % For proper sign convention when subtracted.
% ____________________________________________________________________
% ######################## INITIALIZING INPUT FILES ######################
% Default raw data file formatted columns:
% Irig B Time, Roll, Pitch, Yaw, Platform Azimuth Angle, Vx, Vy, Vz.
% Or modified data file:
% Irig B Time, Vx, Vy, Vz

 fprintf('\nINS raw data file MUST contain ALL the following data \n')
 fprintf('(In column format)\n')
 fprintf('[Irig_B, Yaw, Planform Azimuth Angle, Vx_ins, Vy_ins, Vz_ins]\n\n')
 fprintf('NOTE: Columns do not have to be in this specific order\n\n')

ins_data_file = input('Enter INS data file: ','s');
while isempty(ins_data_file)==1
  ins_data_file = input('Enter INS data file: ','s');
end

 fprintf('\nLoading INS data.....\n\n\n')
 eval(['load ', ins_data_file]);
 eval(['INS = ', ins_data_file, ';']);
 [rins,cins] = size(INS);  % sizing row by column data

 fprintf('INS data is in Geodetic Coordinate System \n')
 fprintf('...(Longitude, Latitude, and Altitude)\n')
 fprintf('Time: milliseconds, Angle: degree, Velocity: feet/second  \n')
 fprintf('\nDEFAULT loads the following column format : \n')
```

```
 fprintf('[Irig_B, Yaw, Pitch, Roll, Azimuth Angle, Vx_ins, Vy_ins,
Vz_ins]\n')
 fprintf('NOTE: Current code uses the following information \n')
 fprintf('      [Irig_B, Yaw, Azimuth Angle, Vx_ins, Vy_ins, Vz_ins] \n\n')
IC = input('Use DEFAULT [< YES >] or << NO >> ','s');
 if isempty(IC)==1
   IC = 'y';
 end

 fprintf('\nConverting INS data into SI Units.....\n')

if IC=='n'|IC=='no'|IC=='N'|IC=='NO'
  IBT = input('Enter Column Number for "INS Irig B time"  :');
  IYW = input('Enter Column Number for "INS Yaw Angle"  :');
  IAA = input('Enter Column Number for "Azimuth Angle"  :');
  IVX = input('Enter Column Number for "INS X Velocity"  :');
  IVY = input('Enter Column Number for "INS Y Velocity"  :');
  IVZ = input('Enter Column Number for "INS Z Velocity"  :');
  raw_ins(:,1) = INS(:,IBT);
  raw_ins(:,2) = INS(:,IYW);
  raw_ins(:,3) = INS(:,IAA);
  raw_ins(:,4) = INS(:,IVX);
  raw_ins(:,5) = INS(:,IVY);
  raw_ins(:,6) = INS(:,IVZ);
end  % End IF IC Loop

 raw_ins = zeros(rins,6);

 if IC=='y'
   raw_ins(:,1) = INS(:,1);            % Irig
   raw_ins(:,2) = INS(:,2);            % Yaw
   % raw_ins(:,#) = INS(:,3:4);        % Pitch, Roll : not used
   raw_ins(:,3) = INS(:,5);            % Azimuth Angle
   raw_ins(:,4:6) = INS(:,6:8);        % Vxins, Vyins, Vzins
 end  % End IF IC Loop

laser_data_file = input('Enter LASER data file: ','s');
while isempty(laser_data_file)==1
  laser_data_file = input('Enter LASER data file: ','s');
end

 fprintf('\nLoading LASER data.....\n\n')
 eval(['load ', laser_data_file]);
 eval(['LASER = ', laser_data_file, ';']);
 [rlaser,claser] = size(LASER);  % sizing row by column data

 fprintf('LASER data is in Runway Coordinate System\n')
 fprintf('...(Xposition, Yposition, and Zposition)\n')
 fprintf('Time: milliseconds, Position: meters, Velocity: meters/second\n')
 fprintf('\nDEFAULT loads the following column format : \n')
 fprintf('[Irig_B, Xlaser, Ylaser]\ n')
 fprintf('NOTE: Current code uses the following information \n')
 fprintf('      [Irig_B, Xlaser, Ylaser, Zlaser]\n\n')
LC = input('Use DEFAULT [< YES >] or << NO >>  ','s');
if isempty(LC)==1
  LC = 'y';
end
```

```
 laserdata = zeros(rlaser,4);
if LC=='n'|LC=='no'|LC=='N'|LC=='NO'
  LB = input('Enter Column Number for "LASER Irig B time"  :');
  LX = input('Enter Column Number for "LASER X Position"  :');
  LY = input('Enter Column Number for "LASER Y Position"  :');
  LZ = input('Enter Column Number for "LASER Z Position"  :');
  laserdata(:,1) = (LASER(:,LB));  % Laser data in SI Units
  laserdata(:,2) = (LASER(:,LX));
  laserdata(:,3) = (LASER(:,LY));
  laserdata(:,4) = (LASER(:,LZ));
end

 if LC=='y'|LC=='yes'|LC=='Y'|LC=='YES'|isempty(LC)==1
   laserdata(:,1) = (LASER(:,1));          % Irig B
   laserdata(:,2:4) = (LASER(:,2:4));  % Convert to English Units
 end

% Making sure data are within range before further processing
 if raw_ins(rins,1) < laserdata(1,1) | laserdata(rlaser,1) < raw_ins(1,1)
   fprintf('\n************************ WARNING ************************\n')
   fprintf('Time stamps between the two file are out of range.\n')
   fprintf('Program TERMINATED, please try again...\n\n')
   break;
 end

printme=input('Save Truth Laser/Kalman file? [< YES >] or << NO >> ','s');
if isempty(printme) == 1
  printme = 'y';
end
 if printme=='y'|printme=='Y'|printme=='yes'|printme=='YES'|printme=='Yes'
  truth_X = input('Enter name of X axis Truth Laser/Kalman output file:
','s');
  while isempty(truth_X)==1
    truth_X=input('Enter name of X axis Truth Laser/Kalman output file:
','s');
  end
  truth_Y = input('Enter name of Y axis Truth Laser/Kalman output file:
','s');
  while isempty(truth_Y)==1
    truth_Y=input('Enter name of Y axis Truth Laser/Kalman output file:
','s');
  end
  truth_Z = input('Enter name of Z axis Truth Laser/Kalman output file:
','s');
  while isempty(truth_Z)==1
    truth_Z=input('Enter name of Z axis Truth Laser/Kalman output file:
','s');
  end
 end  % End IF printme=='Yes' Loop
 if printme=='n'|printme=='N'|printme=='no'|printme=='NO'|printme=='No'
   fprintf('\nGeneric output files have been created: truthX_out, truthY_out,
and truthZ_out.\n\n')
   eval(['!rm truthX_out truthY_out truthZ_out'])
   truth_X = 'truthX_out';
   truth_Y = 'truthY_out';
   truth_Z = 'truthZ_out';
```

```
    printme = 'y';
  end

 save_sync=input('Save synchronized INS/LASER data to file? << YES >> or [< NO
>] ','s');
  if isempty(save_sync) == 1
    save_sync = 'non';
  end

  if save_sync=='y'|save_sync=='Y'|save_sync=='yes'|save_sync=='YES'
    save_sync = 'yes';
   sync_out = input('Enter name of synchronized INS/LASER output file: ','s');
    while isempty(sync_out)==1
     sync_out = input('Enter name of synchronized INS/LASER output file:
','s');
    end
  end
% ____________________________________________________________________
% ################### COORDINATE TRANSFORMATIONS  ######################
% Coordinate Transform of INS from VCV (Geodetic) to RCS
% Variable convention:
% C_nwu_ins = transformation matrix from ins to nwu system
% Conversion from V_ins to V_rcs:
%   V_rcs = C_rcs_pwu * C_pwu_nwu * C_nwu_ins * V_ins;
%   rcs = runway coordinate system (parallel runway, right, down)
%   pwu = Parallel runway, West(left), Up system
%   nwu = North, West, Up system; also Local Level System
%   ins = inertial navigation system, geodesic (long.,lat.,alt)
%
   wander_angle = 10.099;         % Degrees, wander angle
   phi = wander_angle/180.0*pi;  % Angle between True North and X_rcs(rad)

% Read in ins data:
% First Transformation:  From ins to nwu: C_nwu_ins * V_ins = V_nwu
% Second Transformation: From nwu to pwu: C_pwu_nwu * V_nwu = V_pwu
% Third Transformation:  From ned to rcs: C_rcs_pwu * V_pwu = V_rcs
% Translate than Rotate Coordinates
% Final Translation is from the GPS to INU location of aircraft.

   % Preallocate of memory location to increase processing speed
   insdata = zeros(0.8*rins,4);
   C_pwu_nwu = [cos(phi),-sin(phi),0; sin(phi),cos(phi),0; 0,0,1];
   C_rcs_pwu = [1,0,0; 0,-1,0; 0,0,-1];
   fprintf('\nPerforming Coordinate Transform of INS from Local Level to RCS
system\n')

   for i = 1:rins
     insdata(i,1) = raw_ins(i,1);  % Needs no transformation for time
     % alpha = azimuth angle - true heading: (in radians)
     alpha = raw_ins(i,3) - raw_ins(i,2);
     C_nwu_ins=[cos(alpha),-sin(alpha),0; sin(alpha),cos(alpha),0; 0,0,1];

V_rcs=C_rcs_pwu*C_pwu_nwu*C_nwu_ins*[raw_ins(i,4);raw_ins(i,5);raw_ins(i,6)];
     insdata(i,2:4) = V_rcs' * ft2m;
   end  % End For i Loop

% Synchronizing INS and LASER data and transforming LASER to INU location
```

```
fprintf('\nSynchronizing INS and LASER data.....\n')

syncdata = zeros(0.8*rlaser,7);
entry = 2;
sync_row = 0;

if insdata(2,1) < laserdata(2,1)
  while insdata(entry,1) < laserdata(2,1)
    entry = entry + 1;
  end
  if insdata(entry,1) > laserdata(2,1)
    entry = entry - 1;  % To always make laser the larger time stamp
  end
end

 for lsrrow = 2:rlaser
   % If laser and ins time matches, just print to file
   while laserdata(lsrrow,1) == insdata(entry,1) & entry < rins
     sync_row = sync_row + 1;
     syncdata(sync_row,1:4) = insdata(entry,1:4);
     syncdata(sync_row,5:7) = laserdata(lsrrow,2:4);
     entry = entry + 1;
   end  % End WHILE laserdata... = ... Loop

   % Interpolate Irig B time between ins and laser if times dont match
   % Only laser time will be interpolated
   % Calling interpolation function ('inter8.m' is a 'function' file)
   % function = inter8( TL(i), TL(f), TI(n), X(i), X(f))
   % TL=laser time, TI=ins time, i=initial, f=final, n=now, X=Position
   while laserdata(lsrrow,1) > insdata(entry,1) & entry < rins
     sync_row = sync_row + 1;

     NOW(1,1) = feval('inter8', laserdata(lsrrow-1,1), laserdata(lsrrow,1),
insdata(entry,1), laserdata(lsrrow-1,2), laserdata(lsrrow,2));
     NOW(2,1) = feval('inter8', laserdata(lsrrow-1,1), laserdata(lsrrow,1),
insdata(entry,1), laserdata(lsrrow-1,3), laserdata(lsrrow,3));
     NOW(3,1) = feval('inter8', laserdata(lsrrow-1,1), laserdata(lsrrow,1),
insdata(entry,1), laserdata(lsrrow-1,4), laserdata(lsrrow,4));

     syncdata(sync_row,1:4) = insdata(entry,1:4);
     syncdata(sync_row,5:7) = NOW';
     entry = entry + 1;
   end  % End WHILE laserdata... > ... Loop
   if sync_row <= rins
     rhos = sync_row;
   end
end  % End FOR lsrrow Loop

% sizing synchronized row by column data
[rsync,csync] = size(syncdata);
% ____________________________________________________________
% Transferring RHS laser to INU location before entering Kalman filter
 for b = 1:rsync
   syncdata(b,5:7) = syncdata(b,5:7) - laser_inu';
 end

% Equations working with
```

```
% State: X(k+1) = Phi(k) * X(k) + W(k)
% Measurement: Z(k) = H(k) * X(k) + V(k)      Where:
% X(k) = [Xtrue_position; Vtrue_velocity]
% W(k) = [Qlaser 0; 0 Qinu]
% Z(k) = [Xlaser; Vinu]
% V(k) = [Rlaser; Rinu]

% %%%%%%%%%%%%%%%%%%%%% Initializing Constants %%%%%%%%%%%%%%%%%%%%%
% Setting initial values: Length in feet, Time in seconds
g = 10.0; % Set as a criteria for wild data filter
err = [g*3.0; g*3.0;g*3.0];  % Wild data clearance, 3.5 for Xtra clearance
lerr = [g*3.0; g*3.0; g*3.0];  % Laser wild data clearance in feet
ins_rate = 64;  % In Hertz
del_time = 1/ins_rate;
inscount = 0;  % Total number of data points skipped
las_x_total = 0;
las_y_total = 0;
las_z_total = 0;
X_skip_data = 0;  % 0 is good initial data, 1 is bad initial data
Y_skip_data = 0;
Z_skip_data = 0;
track_x_las = 0;
track_y_las = 0;
track_z_las = 0;

fprintf('\nFilter Bandwidth now at +/-%6.3f meters.\n',err)

% %%%%%%%%%%%%%%%%%%% Initializing Kalman Constants %%%%%%%%%%%%%%%%%%%
% Initializing Matrices
phi_Matrix = [1  del_time; 0 1];
Q = [(0.5*ft2m)^2 0; 0 (1*ft2m)^2];  % Initial Guess: Measurement Covariance
H = [1 0; 0 1];  % Ideal noiseless matrix between z & x
R = [(1*ft2m)^2, 0; 0, (0.3)^2];  % 1^2 Initial Guess: State Covariance
IM =[1 0 ; 0 1];  % Identity Matrix
Kx = [1 0; 0 1];   % Initial Guess: Kalman Gain Value
Ky = [1 0; 0 1];   % Initial Guess: Kalman Gain Value
Kz = [1 0; 0 1];   % Initial Guess: Kalman Gain Value
Px_prior = [1 1; 1 1];  % Prior estimate of Xerror covariance
Py_prior = [1 1; 1 1];  % Prior estimate of Xerror covariance
Pz_prior = [1 1; 1 1];  % Prior estimate of Xerror covariance

x_prior = [syncdata(1,5); syncdata(1,2)];  % Set equal to laser data
y_prior = [syncdata(1,6); syncdata(1,3)];
z_prior = [syncdata(1,7); syncdata(1,4)];

Pins(1,1) = syncdata(1,5);  % Set equal to laser data
Pins(2,1) = syncdata(1,6);
Pins(3,1) = syncdata(1,7);

xn = 0;
yn = 0;
zn = 0;

xk = 0;
yk = 0;
zk = 0;
% _____________________ Entering Kalman Loop _____________________
```

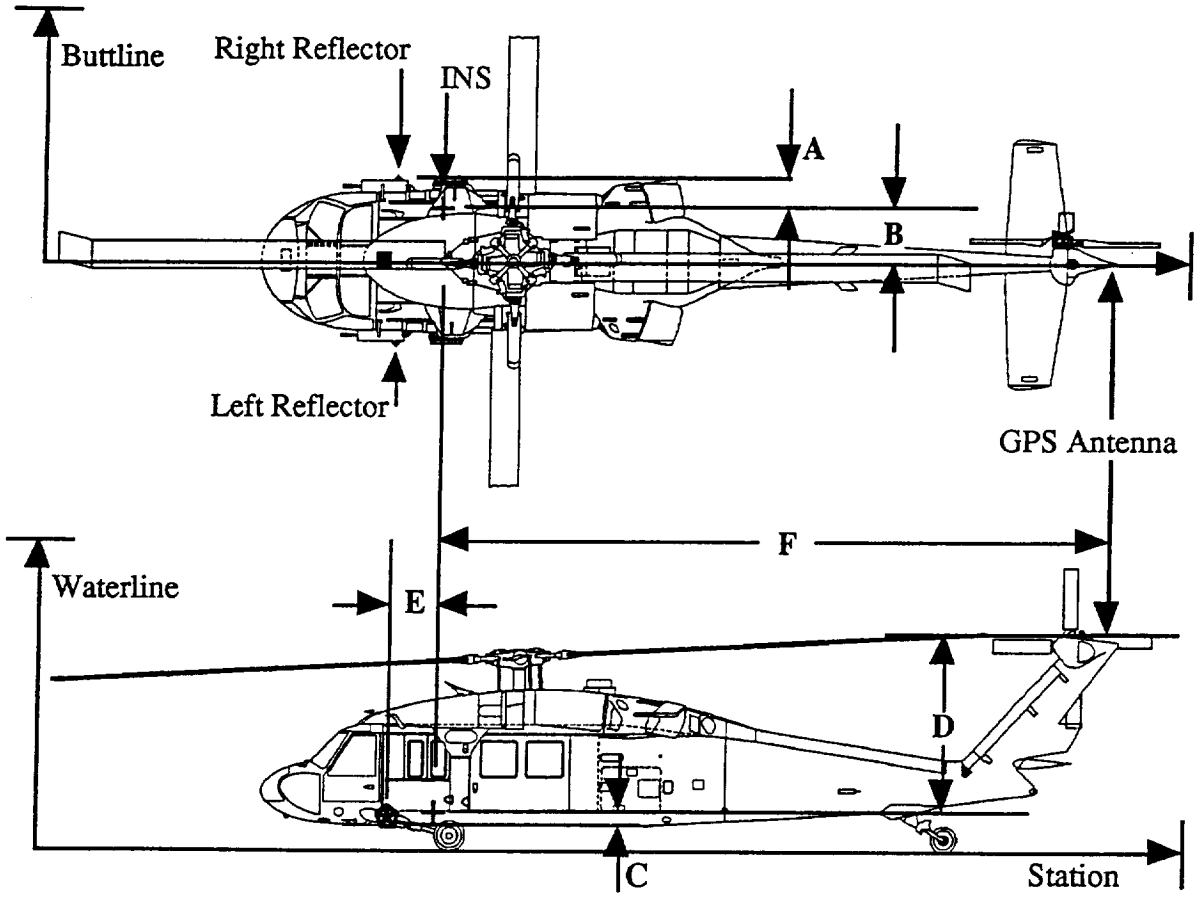
```
% NOTE: output file format
% [Irig B,  Vyinsrcs, Vyupes, Yinsrcs, Yupest, Ylaser, Px, Pv]

kaloutX = zeros(.8*rsync,8);
kaloutY = zeros(.8*rsync,8);
kaloutZ = zeros(.8*rsync,8);

fprintf('\nEntering Kalman Filter Loop\n')

for m = 1:rhos
% STEP ONE:  Compute Kalman Gain
  Kx = Px_prior * H' * (H * Px_prior * H' + R)^(-1);
  Ky = Py_prior * H' * (H * Py_prior * H' + R)^(-1);
  Kz = Pz_prior * H' * (H * Pz_prior * H' + R)^(-1);

  xt = syncdata(m,5);
  xw = x_prior(1,1);
  if abs(xt-xw) > err(1,1)
    X_skip_data = 1;                    % Update ins measurements only
    las_x_total = las_x_total + 1;     % Keep track of total points deleted
  else
    X_skip_data = 0;                    % Status flag, good measurements
  end

  yt = syncdata(m,6);
  yw = y_prior(1,1);
  if abs(yt-yw) > err(2,1)
    Y_skip_data = 1;                    % Update ins measurements only
    las_y_total = las_y_total + 1;     % Keep track of total points deleted
  else
    Y_skip_data = 0;                    % Status flag, good measurements
  end

  zt = syncdata(m,7);
  zw = z_prior(1,1);
  if abs(zt-zw) > err(3,1)
    Z_skip_data = 1;            % Update ins measurements only
    las_z_total = las_z_total + 1;     % Keep track of total points deleted
  else
    Z_skip_data = 0;                    % Status flag, good measurements
  end

  if m == 1
    X_skip_data = 0;  % To ensure 1st data is read as good measurement
    Y_skip_data = 0;
    Z_skip_data = 0;
  end

% Determine Wild Laser Data for Output to file, this is for output purposes
%  not used by code for Kalman analysis
% If wild data point is bad print last "good" data point.
  if m > 1 & m < rhos  -1
    if abs(syncdata(m,5)-syncdata(m-1,5)) > lerr(1,1)
      xn = xn + 1;
    else
      xn = 0;
    end
```

```
    xk = m - xn;

    if abs(syncdata(m,6)-syncdata(m-1,6)) > lerr(2,1)
      yn = yn + 1;
    else
      yn = 0;
    end
    yk = m - yn;

    if abs(syncdata(m,7)-syncdata(m-1,7)) > lerr(3,1)
      zn = zn + 1;
    else
      zn = 0;
    end
    zk = m - zn;

  else
    xk = m;
    yk = m;
    zk = m;
  end

% STEP TWO:  Update estimate with measurement z
  if X_skip_data == 0  % Update and use good measurement data
    zx = [syncdata(m,5); syncdata(m,2)];  % X axis
  else
    zx = [x_prior(1,1); syncdata(m,2)];
  end
  x_upest = x_prior +  Kx * (zx - H * x_prior);

  if Y_skip_data == 0  % Update and use good measurement data
    zy = [syncdata(m,6); syncdata(m,3)];  % Y axis
  else
    zy = [y_prior(1,1); syncdata(m,3)];
  end
  y_upest = y_prior +  Ky * (zy - H * y_prior);

  if Z_skip_data == 0  % Update and use good measurement data
    zz = [syncdata(m,7); syncdata(m,4)];  % Z axis
  else
    zz = [z_prior(1,1); syncdata(m,4)];
  end
  z_upest = z_prior +  Kz * (zz - H * z_prior);

% STEP THREE:  Compute Error Covariance for Updated Estimate
  Px_upest = (IM - Kx * H) * Px_prior;
  Py_upest = (IM - Ky * H) * Py_prior;
  Pz_upest = (IM - Kz * H) * Pz_prior;

% Output in following column format:
% [Irig, Vxins, Vkalman, Xinsrcs, Xkalman, Xlaser, Px, PVx]
  kaloutX(m,1) = syncdata(m,1);
  kaloutX(m,2:5) = [syncdata(m,2), x_upest(2,1), Pins(1,1), x_upest(1,1)];
  kaloutX(m,6:8) = [syncdata(xk,5), Px_upest(1,1), Px_upest(2,2)];
  Pins(1,1) = Pins(1,1) + syncdata(m,2) * del_time;

  kaloutY(m,1) = syncdata(m,1);
```

```
  kaloutY(m,2:5) = [syncdata(m,3), y_upest(2,1), Pins(2,1), y_upest(1,1)];
  kaloutY(m,6:8) = [syncdata(yk,6), Py_upest(1,1), Py_upest(2,2)];
  Pins(2,1) = Pins(2,1) + syncdata(m,3) * del_time;

  kaloutZ(m,1) = syncdata(m,1);
  kaloutZ(m,2:5) = [syncdata(m,4), z_upest(2,1), Pins(3,1), z_upest(1,1)];
  kaloutZ(m,6:8) = [syncdata(zk,7), Pz_upest(1,1), Pz_upest(2,2)];
  Pins(3.1) = Pins(3,1) + syncdata(m,4) * del_time;

% STEP FOUR:  Project Ahead
  x_prior  = phi_Matrix * x_upest;
  y_prior  = phi_Matrix * y_upest;
  z_prior  = phi_Matrix * z_upest;
  Px_prior = phi_Matrix * Px_upest * phi_Matrix' +  Q;
  Py_prior = phi_Matrix * Py_upest * phi_Matrix' +  Q;
  Pz_prior = phi_Matrix * Pz_upest * phi_Matrix' +  Q;

end  % End IF m Loop

fprintf('\nNumber of X LASER points thrown out:  %6g\n',las_x_total)
fprintf('\nNumber of Y LASER points thrown out:  %6g\n',las_y_total)
fprintf('\nNumber of Z LASER points thrown out:  %6g\n',las_z_total)

toc

% [Irig, Vxins, Vkalman, Xinsrcs, Xkalman, Xlaser, Px, PVx]
 if printme == 'y'|printme=='YES'|printme=='yes'|printme=='Y'
 [rkalout,ckalout] = size(kaloutX);

   for v = 1:rkalout
     fprintf(truth_X,'%8.0f  ',kaloutX(v,1))
     fprintf(truth_X,'%10.5f  ',kaloutX(v,2:7))
     fprintf(truth_X,'%10.5f\n',kaloutX(v,8))

     fprintf(truth_Y,'%8.0f  ',kaloutY(v,1))
     fprintf(truth_Y,'%10.5f  ',kaloutY(v,2:7))
     fprintf(truth_Y,'%10.5f\n',kaloutY(v,8))

     fprintf(truth_Z,'%8.0f  ',kaloutZ(v,1))
     fprintf(truth_Z,'%10.5f  ',kaloutZ(v,2:7))
     fprintf(truth_Z,'%10.5f\n',kaloutZ(v,8))
   end
 end  % End IF printme Loop

 if save_sync == 'yes'
   for j = 1:rsync
     fprintf(sync_out,'%8.0f  ',syncdata(j,1))
     fprintf(sync_out,'%10.5f  ',syncdata(j,2:6))
     fprintf(sync_out,'%10.5f\n',syncdata(j,7))
   end  % End FOR Loop
 end  % End IF Loop
 toc
end
```

APPENDIX C

LOCATIONS OF NAVIGATION COMPONENTS



Laser to INU	GPS to INU	Station, m (in)	Buttline, m (in)	Waterline, m (in)
A		--	0.6096 (24.0)	--
	B	--	0.8128 (32.0)	--
C		--	--	0.1524 (6.0)
	D	--	--	3.0861 (121.5)
E		1.2129 (47.75)	--	--
	F	11.7412 (462.25)	--	--