ANTENNA BORESIGHT CALIBRATION

USING OPTIMAL ESTIMATION TECHNIQUES

by

JOHN GORDON CHRISTIANSON

B.A.Sc., The University of British Columbia, 1975

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Electrical Engineering)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 1993

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date September 28, 1993

DE-6 (2/88)

ABSTRACT

As part of the Canadian Department of National Defence Spotlight SAR program, the Defence Research Establishment Ottawa (DREO) has been developing a Synthetic Aperture Radar Motion Compensation System (SARMCS). The SARMCS processes accelerometer and gyro measurements from an inertial measurement unit (IMU) to estimate and compensate for radar antenna motion. Uncompensated azimuth misalignment of the antenna boresight with respect to the IMU is a significant contributor to motion compensation error. Currently, an awkward to perform, laser based antenna boresight calibration procedure is employed.

This thesis investigates an alternative approach in which the antenna azimuth misalignment is estimated in-flight by means of a Kalman filter. The filter integrates data from an inertial navigation system (INS), a Global Positioning System receiver (GPS) and the IMU to maintain position, velocity and orientation information. During calibration, measurements of the antenna azimuth angle to a fixed radar target are provided by a Target Acquisition Unit (TAU). A measurement of antenna azimuth misalignment is obtained by comparing the TAU-determined antenna azimuth with a computed value based on the IMU position and heading and the known target location. Several of these measurements are processed by the Kalman filter to obtain an accurate estimate of the antenna azimuth misalignment.

The current SARMCS Kalman filter integrates INS, GPS and IMU data for navigation and motion compensation purposes. Formulation of an antenna boresight calibration Kalman filter is discussed with the objective of augmenting the SARMCS filter for antenna calibration. Much of the investigation involves simulations to determine a flight trajectory and aircraft/target

geometry that will allow the filter to accurately estimate antenna misalignment in a minimum amount of time. A computer simulation software package, developed to support this effort, is described and the analysis methods and results of the simulations are presented. It is shown that two horizontal manoeuvres a few minutes apart followed by a period of TAU measurements while flying directly toward the target is effective. A performance evaluation using synthesized sensor data provided by DREO suggests that calibration to an accuracy of better than 4 arc minutes rms within a 20 minute period is feasible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS AND ABBREVIATIONS

DREO	Defence Research Establishment Ottawa
GPS	Global Positioning System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
LOS	Line-of-sight
SAR	Synthetic Aperture Radar
SARMCS	Synthetic Aperture Radar Motion Compensation System
TAU	Target Acquisition Unit
WGS 84	World Geodetic System 1984

NOMENCLATURE

α	wander angle
φ	geodetic latitude
λ	longitude
h	altitude
θ	aircraft pitch
ϕ	aircraft roll
ψ	aircraft heading
ξ	azimuth (platform heading)
β	bearing
ε	antenna boresight azimuth misalignment
ϑ	antenna boresight azimuth
x, y, z	coordinate system axis designations
a	ellipsoid semimajor axis
b	ellipsoid semiminor axis
f	ellipsoid flattening
e	ellipsoid first eccentricity
ω_{ie}	earth rotation rate
GM	earth gravitational constant
γ	theoretical gravity at surface of ellipsoid
γ_e	theoretical gravity at equator
γ_p	theoretical gravity at poles

M	radius of curvature in plane of meridian
N	radius of curvature in plane of prime vertical
R_x, R_y, T	general radii of curvature terms
g	gravity
Δg	gravity anomaly
η	small angle vertical deflection of gravity vector
$\mathbf{R}$	position vector (also measurement noise covariance matrix)
$\mathbf{V}$	velocity vector
$\mathbf{f}$	specific force vector
$\mathbf{g}$	gravity vector
ω_{ab}	vector angular velocity of b frame with respect to a frame
$\mathbf{C}_a^b$	direction cosine matrix which transforms a vector from a coordinates to b coordinates
$\mathbf{\Omega}_{ab}$	skew symmetric matrix representation of $[\omega_{ab}\times]$ operator
$\delta\mathbf{R}$	position error vector
$\delta\mathbf{V}$	velocity error vector
ϕ	small angle platform frame misalignment vector
$\delta\theta$	small angle computer frame misalignment vector
$\delta\mathbf{f}$	accelerometer specific force measurement error vector
ϵ	gyro angular velocity measurement error vector
$\mathbf{x}$	state vector
$\mathbf{F}$	continuous time system dynamics matrix
$\mathbf{w}$	process noise vector
$\mathbf{Q}$	process noise spectral density matrix
$\mathbf{x}_k$	discrete time state vector
$\mathbf{\Phi}_k$	discrete time state transition matrix
$\mathbf{w}_k$	discrete time process noise vector
$\mathbf{Q}_k$	process noise covariance matrix

$\mathbf{z}_k$	measurement vector
$\mathbf{H}_k$	measurement matrix
$\mathbf{v}_k$	measurement noise vector
$\mathbf{R}_k$	measurement noise covariance matrix (also position vector)

ACKNOWLEDGEMENT

The idea of antenna boresight calibration using optimal estimation techniques was suggested to me as a thesis topic by Mr. D.J. DiFilippo of the Defence Research Establishment Ottawa (DREO). The work was supported by DSS contract number W7714-2-9649/01-ST.

I wish to thank Dr. M.R. Ito, my supervisor at the University of British Columbia, for his support and especially for his efforts in obtaining and administering the contract with DREO. I also thank my external supervisor Mr. D.J. DiFilippo for his guidance and for preparing the synthesized sensor data used in the antenna boresight calibration performance evaluation.

SECTION ONE

INTRODUCTION

The Canadian Department of National Defence is developing a high resolution airborne spotlight synthetic aperture radar (SAR) (Haslam, Vant and DiFilippo 1988). As part of the Spotlight SAR program, the Defence Research Establishment Ottawa (DREO) has been involved in the development of a SAR Motion Compensation System (SARMCS) (Hepburn et al. 1984; DiFilippo, Haslam and Widnall 1988). This system compensates SAR signal returns for the effects of aircraft motion during the imaging interval. The SARMCS design incorporates a small strapdown inertial measurement unit (IMU), mounted directly on the radar antenna structure, to measure antenna motion. Raw accelerometer and gyro measurements are processed to estimate the motion of the antenna phase centre along the radar line-of-sight (LOS). Currently, an experimental version of the SARMCS is installed on board the National Aeronautical Establishment Convair 580 research aircraft. Figure 1-1 shows the locations of the motion compensation sensors on the aircraft. Since it is intended that the new antenna boresight calibration technique described herein will ultimately be incorporated into the SARMCS as a special operating mode, these sensors will also be used for the antenna boresight calibration.

Figure 1-2 depicts the SAR antenna and the strapdown IMU. In the figure, the point O represents the centre of the IMU accelerometer triad and the directions x, y and z are the accelerometer input axes. Antenna azimuth is defined as the angle ϑ between the input axis of the forward pointing accelerometer and the direction of the antenna boresight. Measurements of this angle are provided by an azimuth encoder on the antenna structure. In general, there is a fixed error component associated with the measurements due to mechanical misalignment of

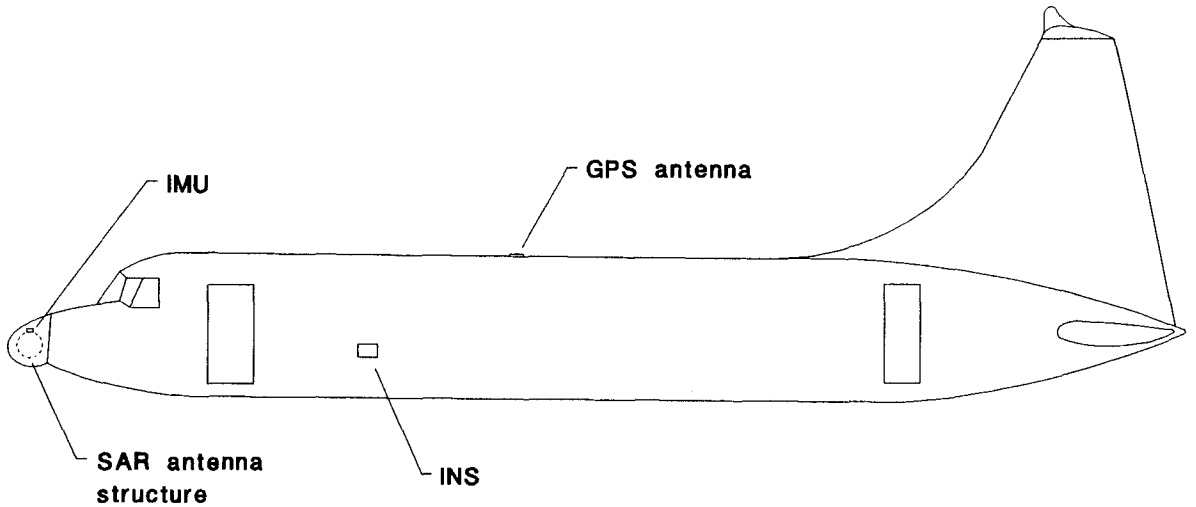


Figure 1-1 Location of Motion Compensation Sensors on Convair 580

the antenna boresight with respect to the strapdown IMU mount. Antenna boresight calibration refers to the determination of this azimuth misalignment.

An analysis of the error in the phase centre LOS displacement (DiFilippo, Haslam and Widnall 1988) indicates the importance of accurately resolving the phase centre acceleration, as measured by the strapdown IMU accelerometers, along the radar LOS. Uncompensated azimuth misalignment of the antenna boresight with respect to the IMU is a significant contributor to the error. The SARMCS error budget specifies that the antenna azimuth be calibrated to an accuracy of 5 arc minutes rms or better. To achieve this level of accuracy, the fixed mechanical misalignment must be estimated and used to correct the azimuth encoder measurements.

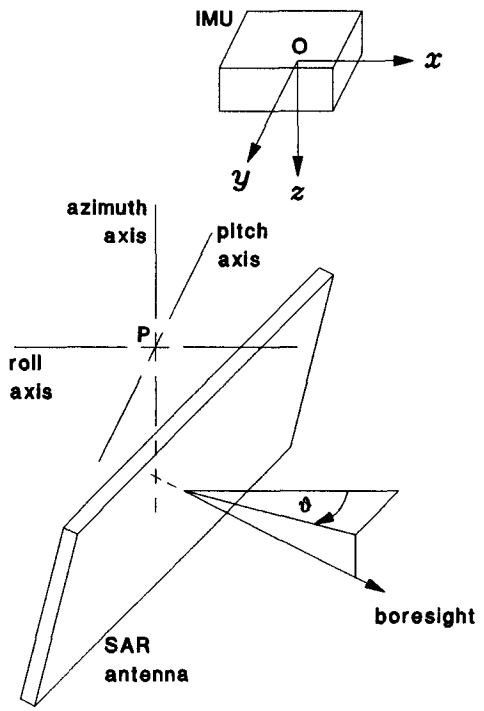


Figure 1-2 SAR Antenna and Strapdown IMU

Currently, a laser based antenna boresight calibration procedure is employed. While the procedure provides the required accuracy, it is awkward to perform. Firstly, it requires removal of the SAR antenna structure from the aircraft and attachment of a small laser in place of the IMU. Secondly, the entire structure must be mounted atop a 20 metre tower and operated to lock the antenna onto a radar transponder target located a distance of 5.5 kilometres away. Finally, the azimuth misalignment is estimated by comparing azimuth encoder measurements with measurements obtained by laser sighting. Although this calibration need only be performed infrequently, it is still too cumbersome to be an operational procedure. This thesis investigates an alternative approach in which the antenna azimuth misalignment is estimated in-flight by means of a Kalman filter.

1.1 PROPOSED CALIBRATION TECHNIQUE

The proposed calibration technique uses a Kalman filter to estimate the antenna azimuth misalignment. Figure 1-3 is a plan view of the azimuth angle relationships relevant to this discussion.

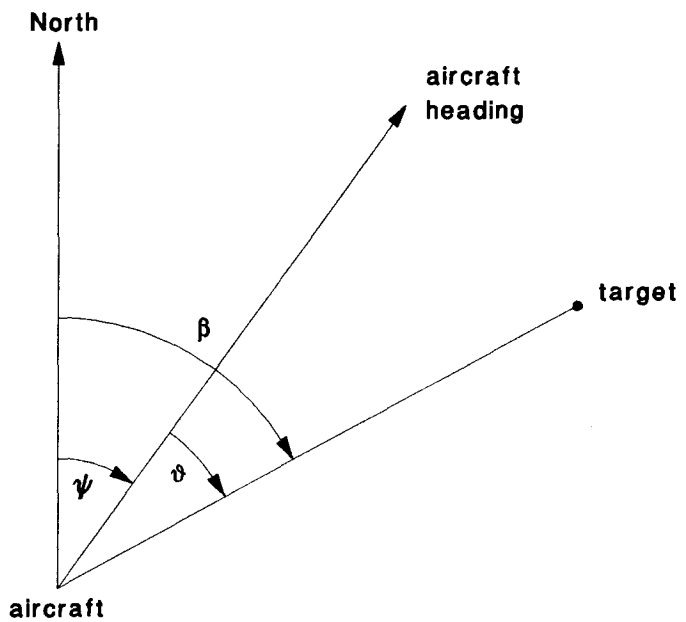


Figure 1-3 Plan View of Azimuth Angle Relationships

While in flight, the aircraft position and heading are computed by processing IMU measurements in a strapdown inertial navigation algorithm. Inertial navigation errors are controlled by processing measurements based on comparing Global Positioning System (GPS) (Milliken and Zoller 1978; Zachmann 1988) and inertial information in the Kalman filter. During calibration, a radar transponder, situated at a well surveyed fixed location on the ground, is used to generate point target returns. The radar incorporates a Target Acquisition Unit (TAU) which can detect

the point target, by processing radar returns, and measure the corresponding antenna azimuth angle using the azimuth encoder. The TAU provides a measurement of the azimuth angle ϑ to the target and at the same time, a value for the angle is computed as

$$\vartheta = \beta - \psi \tag{1-1}$$

where β is the bearing of the target from north, computed from the strapdown navigator indicated position and the surveyed location of the target, and ψ is the strapdown navigator indicated heading. A measurement of the antenna azimuth error is now obtained as

$$\varepsilon = \vartheta_{TAU} - \vartheta_C \tag{1-2}$$

where ϑ_{TAU} is the TAU-determined antenna azimuth and ϑ_C is the computed value based on the strapdown IMU inertial navigation quantities and the known target location. By processing a number of these measurements in the Kalman filter, an accurate estimate of the antenna azimuth misalignment is achieved.

A further consideration is the possibility of losing the GPS information for short periods (for example, the GPS antenna may be in the shadow of the fuselage during a turn). The high drift characteristics of the small IMU gyros make this of particular concern. In order to ensure a robust system, the proposed calibration Kalman filter includes data from an inertial navigation system (INS) with low drift gyros to maintain stability during those short periods of GPS unavailability.

1.2 SCOPE OF THIS THESIS

The proposed antenna boresight calibration technique offers the potential of a simple procedure for accurate, in-flight, real-time calibration. This thesis investigates the proposed technique to determine if the anticipated benefits can be realized. The objectives of the investigation are to:

- Formulate an antenna boresight calibration Kalman filter to augment the current SARMCS Kalman filter.

- Determine an effective flight profile and aircraft/target geometry for antenna boresight calibration.

- Evaluate the resulting antenna calibration performance in terms of misalignment estimation accuracy and required calibration time.

The first task in the investigation is a review the navigation concepts applicable to bearing calculations and inertial navigation on the earth. Results important to the present work are described in Section 2.

Section 3 describes the formulation of the Kalman filter. This task begins with an examination of the current SARMCS Kalman filter. It is shown that the SARMCS filter can be augmented for antenna calibration with a random constant state representing antenna azimuth misalignment, and a new antenna azimuth matching measurement. A linearized antenna azimuth matching measurement model is derived.

Much of the investigation involves simulations to determine the flight profile and aircraft/target geometry that will allow the Kalman filter to estimate the antenna boresight azimuth misalignment to the required accuracy in a minimum amount of time. A computer simulation software package, developed to support this effort, is described in Section 4. Major features of the package include: generation of arbitrary test flight trajectories specified by simple command file input; synthesis of sensor data corresponding to the flight trajectory with major sensor error sources included; processing of sensor data using the inertial navigation algorithm described in Section 2 and the Kalman filter formulation described in Section 3; evaluation of the results by graphing and comparison with true trajectory and true sensor error data.

Section 5 describes the test methods and presents the results of the flight profile and aircraft/target geometry simulations. It is shown that two simple turning manoeuvres a few minutes apart followed by a period of TAU measurements while flying directly toward the target is effective in allowing the Kalman filter to estimate the antenna azimuth misalignment.

Evaluation of the Kalman filter performance is described in Section 6. This task involves the processing of simulated sensor data provided by DREO that is generated using their SARMCS synthesis package. Results suggest that estimation of the antenna azimuth misalignment to an accuracy of better than 4 arc minutes rms within a 20 minute period is feasible.

Conclusions and recommendations regarding the Kalman filter design, the test methods, the flight profile and aircraft/target geometry, and the calibration performance are summarized in Section 7.

SECTION TWO

REVIEW OF NAVIGATION CONCEPTS

Aircraft navigation and, in particular, inertial navigation, has a dominating influence on the formulation of the antenna boresight calibration Kalman filter. A review of some fundamental concepts and specific results, which will be used in subsequent sections, is presented here.

2.1 FRAMES OF REFERENCE AND COORDINATE SYSTEMS

Several coordinate systems are commonly encountered in navigation. The coordinate systems used in this thesis and described in the following paragraphs are all right-handed, orthonormal systems and are chosen to be compatible with the SARMCS.

2.1.1 Earth-Centred Inertial Frame

Newton's laws of motion hold true only in an inertial frame. A coordinate system that has its origin at the mass centre of the earth and that is non-rotating relative to the "fixed" stars, can be considered to be an inertial frame for measurements made in the vicinity of the earth. The IMU provides measurements of angular velocity and specific force with respect to this frame.

2.1.2 Geocentric Earth-Fixed Frame

The geocentric earth-fixed frame has its origin at the earth's centre and its basis vectors fixed to the earth. The x axis is along the rotational axis of the earth and points towards the north pole, the y axis is in the equatorial plane at -90° longitude and the z axis is in the equatorial plane at 0° longitude (Greenwich Meridian). The earth-fixed frame rotates with respect to the inertial frame at the constant sidereal rate.

2.1.3 Geographic North-East-Down Frame

The geographic north-east-down frame is a locally level frame with its axes pointing true north, east and down. The x axis points north, the y axis points east and the z axis points down. Although this frame is intuitively appealing, it is usually not used as an inertial navigation computational coordinate system due to singularities at the poles. The closer the aircraft passes to the pole, the faster the geographic frame must rotate about the vertical axis to keep the x axis pointing north, going to infinite angular velocity in the limit.

2.1.4 Wander Azimuth Navigation Frame

The wander azimuth frame solves the high latitude problem of the geographic frame. It is used as the computational coordinate system for inertial navigation. Like the geographic frame, the wander azimuth frame is locally level. Unlike the geographic frame, however, the vertical component of its angular velocity relative to the earth-fixed frame is, by definition, identically equal to zero. The direction of the horizontal axes varies (wanders) depending on the path taken by the aircraft. Poor (1989) has given a description of the wander azimuth frame in particularly understandable terms. The frame has its x axis displaced from true north through the wander angle, its z axis pointing along the local vertical in the upward direction and its y axis completing the right-handed system. The Euler angles relating the wander azimuth frame to the earth-fixed frame are the longitude λ, the geodetic latitude φ, and the wander angle α. Figure 2-1 illustrates the relationship between the earth-fixed frame and the wander azimuth frame.

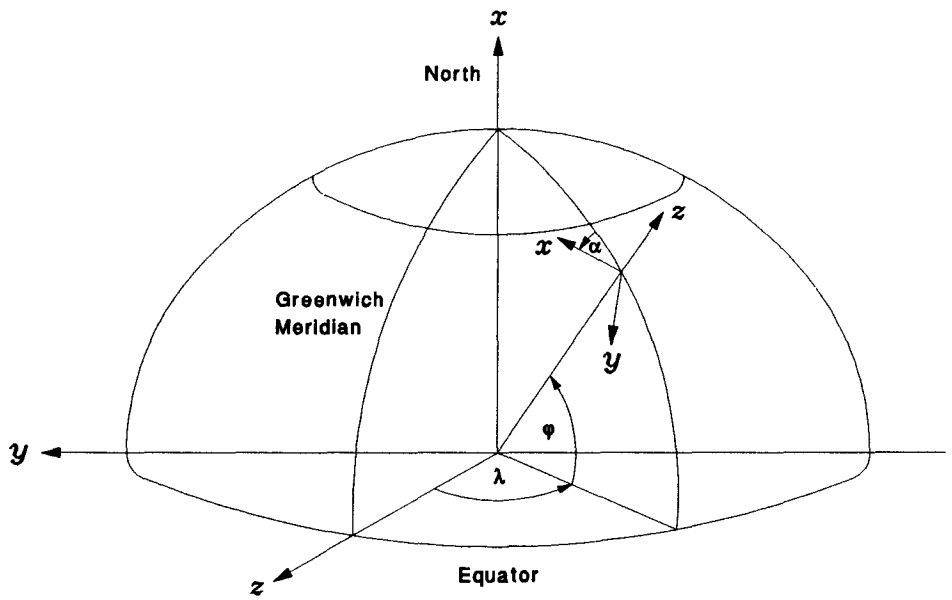


Figure 2-1 Relationship Between Earth-Fixed Frame and Wander Azimuth Frame

2.1.5 Vehicle Body Frame

The vehicle body frame is fixed to the aircraft body. The x axis points in the forward direction along the longitudinal (roll) axis, the y axis points to the right of the aircraft along the lateral (pitch) axis and the z axis points downward along the normal (yaw) axis. For this work, it is convenient to define any point fixed to the aircraft body in terms of its displacement vector (lever arm) from the INS. The vehicle body frame is, therefore, defined to have its origin at the centre of the INS accelerometer/gyro instrument cluster. Lever arms from the INS to the GPS antenna and from the INS to the intersection of the antenna roll and pitch axes (point P in Figure 1-2) are defined in this frame.

2.1.6 Stabilized Antenna Ring Gear Frame

The stabilized antenna ring gear frame has its origin at the intersection of the antenna roll and pitch axes (point P in Figure 1-2). The x axis points in the forward direction along the antenna roll axis, the y axis is to the right along the antenna pitch axis and the z axis is downward along the antenna azimuth axis. Since the antenna ring gear is roll and pitch stabilized, the ring gear frame approximates a locally level frame. The origin of the antenna ring gear frame is the origin for the bearing calculations performed during antenna boresight calibration.

2.1.7 IMU Body Frame

IMU accelerometer and gyro measurements are given in the IMU body frame. This frame has its origin at the centre of the IMU accelerometer/gyro instrument cluster and its basis vectors aligned with the instrument input axes (see Figure 1-2). The x axis points forward, the y axis points to the right, and the z axis points downward. The orientation of the IMU body frame is fixed with respect to the antenna ring gear frame. A fixed lever arm (point P to point O in Figure 1-2) locates the IMU relative to the antenna roll/pitch intersection.

2.2 THE EARTH MODEL

Computations involving positions on the earth (for example, bearing calculations and inertial navigation) require a mathematical model of the shape of the earth. In addition, when measurements from accelerometers and gyros are to be used, a model describing the earth's gravity and a knowledge of the earth's rotational rate are required. Detailed descriptions of the models and their derivations are given in Ewing and Mitchell (1970), Heiskanen and Moritz (1967), Dragomir et al. (1982) and others.

2.2.1 Shape

Roughly speaking, the earth is spherical. However, its departure from a true spherical shape is too great for this model to be used for navigation calculations. Surface irregularities are easily handled by adopting mean sea level as the altitude reference. The surface defined by mean sea level is an equipotential surface known as the geoid. It is everywhere normal to the direction of gravity but, due to mass concentration variations within the earth, the shape of the geoid can only be approximated. Because the earth rotates, it assumes a shape that is bulging at the equator and flattened at the poles. A figure which adequately approximates the shape of the geoid is the ellipsoid of revolution. It is produced by rotating an ellipse about its minor axis, with the major axis generating the equatorial plane. Figure 2-2 illustrates the relationship between the physical surface, the geoid and the ellipsoid of revolution. The vertical separation of the geoid from the ellipsoid is known as the geoidal undulation and the angle between the normal to the ellipsoid and the normal to the geoid (the direction of gravity) is called the deflection of the vertical.

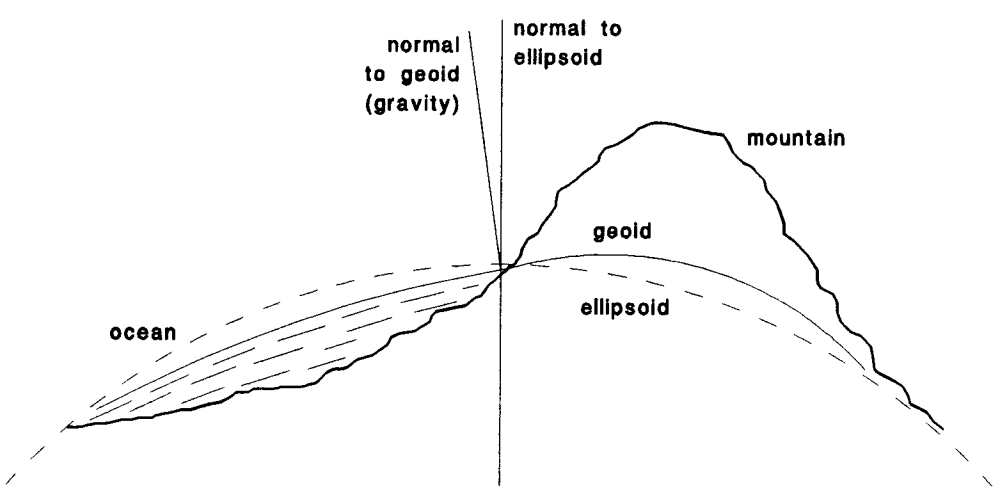


Figure 2-2 Relationship Between Physical Surface, Geoid and Ellipsoid

Figure 2-3 shows the ellipse with a major axis of length $2a$ and a minor axis of length $2b$. The equation of the ellipse is

$$\frac{x^2}{b^2}+\frac{y^2}{a^2}=1 \tag{2-1}$$

The flattening of the ellipse, known as f, is defined by

$$f=\frac{a-b}{a} \tag{2-2}$$

Very often, the two parameters a and f are used to define the ellipsoid.

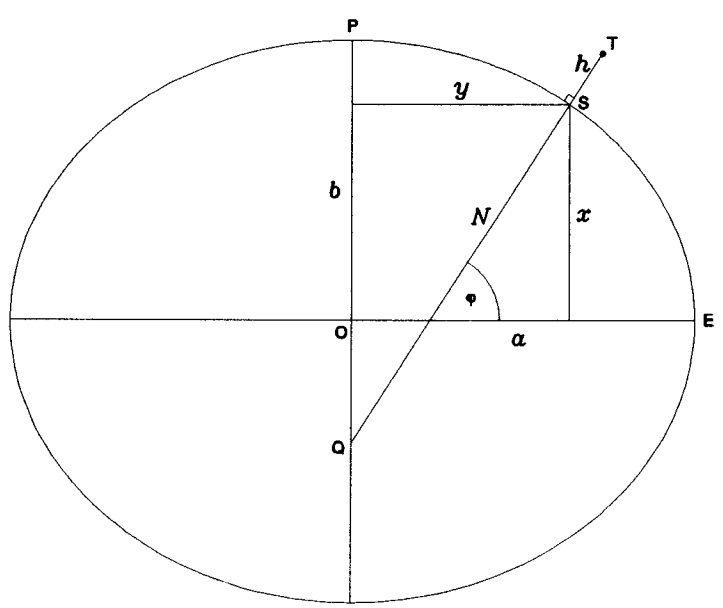


Figure 2-3 The Ellipse

A parameter of the ellipse often used in computations is the square of the first eccentricity, designated e^2, which is defined as

$$e^2 = \frac{a^2-b^2}{a^2} = 2f-f^2 \tag{2-3}$$

Additional properties of the ellipsoid which are needed for navigation calculations are the radius of curvature in the plane of the meridian, designated M, and the radius of curvature in the plane of the prime vertical, designated N. At any point on the ellipse (for example, point S in Figure 2-3), the radius of curvature of the ellipse in the plane of the meridian is given by

$$M = \frac{a(1-e^2)}{(1-e^2\sin^2\varphi)^{3/2}} \tag{2-4}$$

where φ, the geodetic latitude, is the angle between the equatorial plane and the normal to the ellipsoid at the point. The radius of curvature in the plane of the prime vertical is the radius of curvature through any point on the surface of the ellipsoid (for example, point S in Figure 2-3) in a plane at right angles to the plane of the meridian. The distance QS in Figure 2-3 is identical to the radius of curvature in the prime vertical through point S. The value of N is given by

$$N = \frac{a}{(1-e^2\sin^2\varphi)^{1/2}} \tag{2-5}$$

The position of a point (for example, point T in Figure 2-3) relative to the centre of the ellipsoid may be expressed in Cartesian coordinates (x, y, z) or in curvilinear coordinates (geodetic latitude φ, longitude λ and ellipsoidal height h). Ellipsoidal height (or altitude) is the distance above the ellipsoid measured along the normal to the ellipsoid at the point. The transformation from curvilinear geodetic coordinates to Cartesian coordinates is given by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ((1-e^2)N+h)\sin\varphi \\ -(N+h)\cos\varphi\sin\lambda \\ (N+h)\cos\varphi\cos\lambda \end{bmatrix} \tag{2-6}$$

The transformation from Cartesian coordinates to curvilinear geodetic coordinates is most often performed iteratively. Equations (2-6) can be rearranged to obtain

$$\tan\lambda = \frac{-y}{z} \tag{2-7}$$

$$\tan\varphi = \frac{N+h}{(1-e^2)N+h}\ \frac{x}{(y^2+z^2)^{1/2}} \tag{2-8}$$

$$h = \frac{x}{\sin\varphi} - (1-e^2)N = \frac{(y^2+z^2)^{1/2}}{\cos\varphi} - N \tag{2-9}$$

λ is obtained directly from (2-7). To solve for latitude and altitude, first obtain an initial estimate for the latitude from (2-8) by setting e^2 to zero. Next, calculate N using (2-5). Now,

compute h using (2-9) and φ using (2-8). Finally, iterate the calculation of N, h and φ until the required accuracy is achieved (typically four to six iterations).

2.2.2 Rotation

The rate of the earth's rotation with respect to the inertial frame is designated ω_{ie}. Conceptualization of the rotation rate is aided by imagining a line joining the earth and the sun. The period of rotation of the earth with respect to the earth-sun line is one day and the earth-sun line experiences one revolution in approximately 365.25 days. The result is a constant angular velocity of the earth (called the sidereal rate) of about $(365.25+1)/(365.25\times 24)$ cycles per hour.

2.2.3 Gravity

The force acting on a body at rest on the earth's surface is the resultant of the gravitational force and the centrifugal force of the earth's rotation. The total force is called gravity. Theoretical gravity at the surface of the ellipsoid is

$$\gamma = \frac{a\gamma_e \cos^2\varphi + b\gamma_p \sin^2\varphi}{(a^2\cos^2\varphi + b^2\sin^2\varphi)^{1/2}} \tag{2-10}$$

which represents the closed formula due to Somigliana (Heiskanen and Moritz 1967). In this equation, γ_e is the theoretical gravity at the equator and γ_p is the theoretical gravity at the poles. The formula may be expressed as a function of the eccentricity as

$$\gamma = \frac{\gamma_e(1 + k\sin^2\varphi)}{(1 - e^2\sin^2\varphi)^{1/2}} \tag{2-11}$$

where k is the constant given by

$$k=\frac{b\gamma_p}{a\gamma_e}-1$$

For a small height h above the ellipse, normal gravity can be expanded into a power series in h. To second order, the resulting expression for gravity is

$$g=\gamma[1-\frac{2}{a}(1+f+m-2f\sin^2\varphi)h+\frac{3}{a^2}h^2] \tag{2-12}$$

where m is the constant given by

$$m=\frac{\omega_{ie}^2 a^2 b}{GM}$$

and GM is the earth's gravitational constant.

2.2.4 World Geodetic System

GPS-determined position coordinates refer to the World Geodetic System 1984 earth-centred earth-fixed geodetic datum (Defense Mapping Agency 1987). WGS 84 specifies a set of defining parameters for the ellipsoid from which all of the constants referenced in subsections 2.2.1 through 2.2.3 may be derived. Table 2-1 lists the required constants.

Table 2-1 WGS 84 Constants

Constant	Notation	Value
Semimajor Axis	a	6.378137×10^6 m
Flattening	f	$3.35281066474 \times 10^{-3}$
First Eccentricity Squared	e^2	$6.69437999013 \times 10^{-3}$
Angular Velocity	ω_{ie}	7.292115×10^{-5} rad/s
Gravitational Constant	GM	3.986005×10^{14} m^3/s^2
Normal Gravity at Equator	γ_e	9.7803267714 m/s^2
Gravity Formula Constant	k	$1.93185138693 \times 10^{-3}$

2.3 BEARING CALCULATION

Bearing is the direction to the target measured relative to north and about the vertical at the aircraft's position. Figure 2-4 represents a horizontal plane viewed from an aircraft directly above point A. The plane contains point T which is the position of the target. Note that although the plane is horizontal with respect to the normal to the ellipsoid at A, in general it is not tangent to the ellipsoid since it is defined to contain point T. In the figure, x is the length of line segment BT which is perpendicular to the meridian plane containing A and extends to point T, and y is the length of line segment AB which is perpendicular to the vertical containing A and extends to join the first line segment. Bearing is then given by

$$\beta = \arctan\left[\frac{x}{y}\right] \tag{2-13}$$

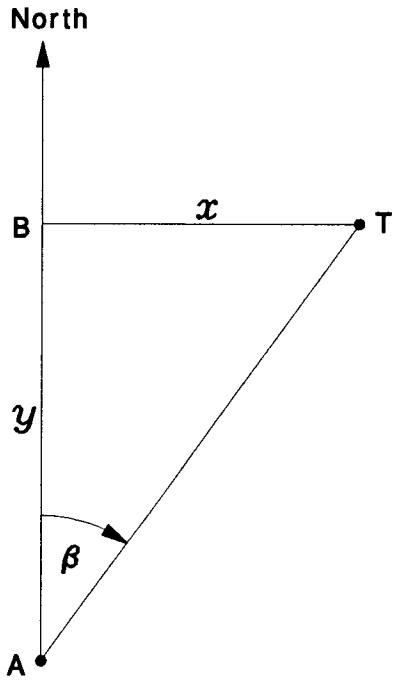


Figure 2-4 Bearing Plane Viewed from an Aircraft Above Point A

Figure 2-5 shows the meridian plane containing the target point T. In the figure, QP is the projection of QT onto a plane parallel to the equator. The length of QP is

$$|\mathrm{QP}| = (N_T + h_T)\cos\varphi_T$$

where N_T is the prime vertical at the target location, h_T is the height of the target and φ_T is the geodetic latitude of the target.

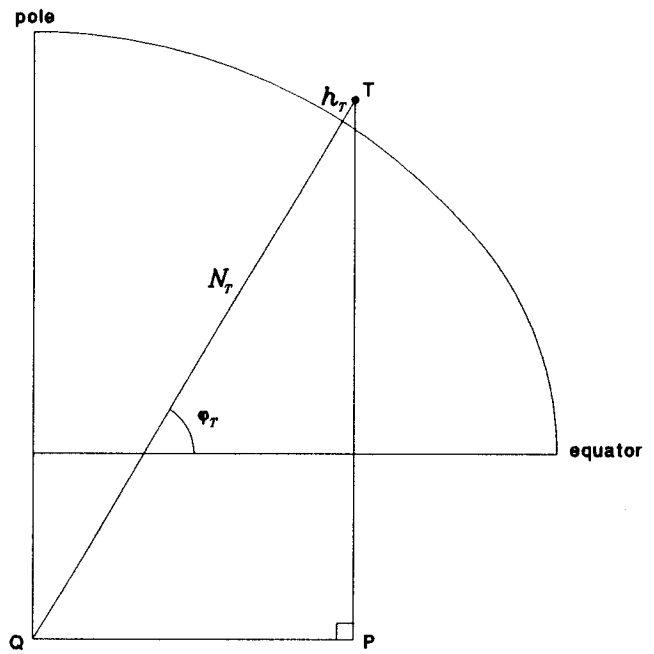


Figure 2-5 Target Meridian Plane

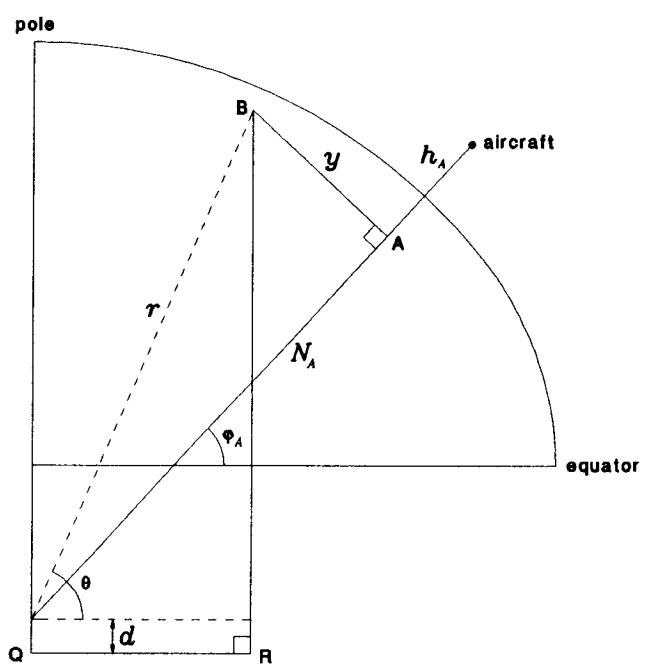


Figure 2-6 Aircraft Meridian Plane

Figure 2-6 shows the meridian plane containing the aircraft and the points A and B. The projection of QP onto this plane is QR which has length

$$|\mathrm{QR}| = (N_T + h_T)\cos\varphi_T \cos(\lambda_T - \lambda_A)$$

where λ_T is the longitude of the target and λ_A is the longitude of the aircraft.

The result of these projections is a right triangle parallel to the equatorial plane with hypotenuse QT and sides QR and RP. But, RP is parallel to BT and, therefore, must have length x so that

$$x = (N_T + h_T)\cos\varphi_T \sin(\lambda_T - \lambda_A) \tag{2-14}$$

To find an expression for y, define the angle θ and the distance r as shown in Figure 2-6. Now y is given by

$$y = r\sin(\theta - \varphi_A) = r(\sin\theta\cos\varphi_A - \cos\theta\sin\varphi_A)$$

where φ_A is the geodetic latitude of the aircraft. The length of BR in Figure 2-6 is equal to the length of TP in Figure 2-5 so that

$$|\mathrm{BR}| = (N_T + h_T)\sin\varphi_T$$

and

$$\sin\theta = \frac{(N_T+h_T)\sin\varphi_T - d}{r}$$

$$\cos\theta = \frac{(N_T+h_T)\cos\varphi_T\cos(\lambda_T-\lambda_A)}{r}$$

where d accounts for the fact that, in general, the prime vertical at the target location and the prime vertical at the aircraft location do not intersect the earth's axis at the same point. Knowing the earth-fixed frame x coordinates of the aircraft and target from equation (2-6) and adjusting for the x displacements to the axis intersection points given by $(N_A+h_A)\sin\varphi_A$ and $(N_T+h_T)\sin\varphi_T$, an expression for the difference d is obtained as

$$d = e^2(N_T\sin\varphi_T - N_A\sin\varphi_A) \tag{2-15}$$

Now y can be written

$$y = [(N_T+h_T)\sin\varphi_T - d]\cos\varphi_A - (N_T+h_T)\cos\varphi_T\sin\varphi_A\cos(\lambda_T-\lambda_A)$$

which, after some algebraic manipulation, becomes

$$y = (N_T+h_T)\left[\sin(\varphi_T-\varphi_A) + \cos\varphi_T\sin\varphi_A(1-\cos(\lambda_T-\lambda_A)) - \frac{d\cos\varphi_A}{N_T+h_T}\right] \tag{2-16}$$

Combining (2-13), (2-14) and (2-16) the expression for bearing becomes

$$\beta = \arctan\left[\frac{\cos\varphi_T \sin\Delta\lambda}{\sin\Delta\varphi + \cos\varphi_T \sin\varphi_A (1-\cos\Delta\lambda) - \dfrac{d\cos\varphi_A}{N_T + h_T}}\right] \tag{2-17}$$

where d is given by (2-15) and $\Delta\varphi$ and $\Delta\lambda$ are defined as

$$\Delta\varphi = \varphi_T - \varphi_A$$

$$\Delta\lambda = \lambda_T - \lambda_A$$

2.4 INERTIAL NAVIGATION

A strapdown IMU provides three-dimensional measurements of specific force (nongravitational acceleration) and angular velocity with respect to an inertial frame. Inertial navigation systems solve Newton's force equations using the accelerometer measurements coordinatized in a frame whose orientation is known by processing the gyro measurements.

2.4.1 Navigation in Wander Azimuth Coordinates

Terrestrial navigation equations in wander azimuth coordinates are derived in many references (Britting 1971; Farrell 1976). The equations are

$$\dot{\mathbf{V}} = \mathbf{f} + \mathbf{g} - (2\omega_{ie} + \omega_{ew}) \times \mathbf{V} \tag{2-18}$$

$$\dot{\mathbf{R}} = \mathbf{V} - \omega_{ew} \times \mathbf{R} \tag{2-19}$$

where

$\mathbf{R}$ = position with respect to earth centre

$\mathbf{V}$ = velocity with respect to earth

$\mathbf{f}$ = specific force

$\mathbf{g}$ = gravity

ω_{ie} = angular velocity of earth-fixed frame with respect to inertial frame

ω_{ew} = angular velocity of wander azimuth frame with respect to earth-fixed frame

and all vectors are expressed in wander azimuth coordinates. The cross product terms are corrections required because the wander azimuth frame and the earth-fixed frame are rotating with respect to the inertial frame.

Typically, (2-19) is not solved explicitly for position. Instead, the angular velocity ω_{ew} is integrated to obtain the geodetic latitude, longitude and wander angle and (2-19) is used only for the vertical position. If the orientation of the wander azimuth frame with respect to the earth-fixed frame is expressed in terms of the direction cosine matrix $\mathbf{C}_w^e$, which transforms a vector in wander azimuth coordinates to the same vector in earth-fixed coordinates, then the horizontal position equation becomes

$$\dot{\mathbf{C}}_w^e = \mathbf{C}_w^e \mathbf{\Omega}_{ew} \tag{2-20}$$

where $\mathbf{\Omega}_{ew}$ is the skew symmetric matrix

$$\mathbf{\Omega}_{ew}=[\omega_{ew}\times]=\begin{bmatrix} 0 & -\omega_{ewz} & \omega_{ewy} \\ \omega_{ewz} & 0 & -\omega_{ewx} \\ -\omega_{ewy} & \omega_{ewx} & 0 \end{bmatrix} \tag{2-21}$$

with ω_{ewx}, ω_{ewy} and ω_{ewz} being the components of $\boldsymbol{\omega}_{ew}$.

$\mathbf{C}_w^e$ can be expressed in terms of the Euler angles φ, λ and α as

$$\mathbf{C}_w^e=\begin{bmatrix} \cos\varphi\cos\alpha & -\cos\varphi\sin\alpha & \sin\varphi \\ \cos\lambda\sin\alpha+\sin\varphi\sin\lambda\cos\alpha & \cos\lambda\cos\alpha-\sin\varphi\sin\lambda\sin\alpha & -\cos\varphi\sin\lambda \\ \sin\lambda\sin\alpha-\sin\varphi\cos\lambda\cos\alpha & \sin\lambda\cos\alpha+\sin\varphi\cos\lambda\sin\alpha & \cos\varphi\cos\lambda \end{bmatrix} \tag{2-22}$$

and φ, λ and α can be obtained from the elements of $\mathbf{C}_w^e$ using the equations

$$\tan\varphi=\frac{C_{13}}{(C_{11}^2+C_{12}^2)^{1/2}} \tag{2-23}$$

$$\tan\lambda=\frac{-C_{23}}{C_{33}} \tag{2-24}$$

$$\tan\alpha=\frac{-C_{12}}{C_{11}} \tag{2-25}$$

To solve the navigation equations, it remains to express the angular velocities ω_{ie} and ω_{ew} in terms of the current position and velocity. ω_{ie} is obtained by transforming its earth-fixed coordinate representation

$$\omega_{ie}^{e} = \begin{bmatrix} \omega_{ie} \\ 0 \\ 0 \end{bmatrix}$$

to wander azimuth coordinates using the direction cosine matrix $\mathbf{C}_{w}^{e}$

$$\omega_{ie} = \left[\mathbf{C}_{w}^{e}\right]^{T} \omega_{ie}^{e} = \begin{bmatrix} \omega_{ie}\cos\varphi\cos\alpha \\ -\omega_{ie}\cos\varphi\sin\alpha \\ \omega_{ie}\sin\varphi \end{bmatrix} \tag{2-26}$$

ω_{ew} is computed from knowledge of the velocity and the radii of curvature. When $\alpha=0$, the wander azimuth x and y axes correspond to the north and west directions and ω_{ew} is given by the expression

$$\omega_{ew}\big|_{\alpha=0} = \begin{bmatrix} \dfrac{-V_y}{N+h} \\ \dfrac{V_x}{M+h} \\ 0 \end{bmatrix}$$

For arbitrary α, the expression for ω_{ew} becomes

$$\omega_{ew} = \begin{bmatrix} \dfrac{-V_x \sin\alpha - V_y \cos\alpha}{N+h}\cos\alpha + \dfrac{V_x \cos\alpha - V_y \sin\alpha}{M+h}\sin\alpha \\ \dfrac{-V_x \sin\alpha - V_y \cos\alpha}{N+h}\sin\alpha + \dfrac{V_x \cos\alpha - V_y \sin\alpha}{M+h}\cos\alpha \\ 0 \end{bmatrix} \tag{2-27}$$

Rearranging (2-27) and defining the inverse radii of curvature terms

$$\frac{1}{R_x} = \frac{(M+h)\sin^2\alpha + (N+h)\cos^2\alpha}{(M+h)(N+h)} \tag{2-28}$$

$$\frac{1}{R_y} = \frac{(M+h)\cos^2\alpha + (N+h)\sin^2\alpha}{(M+h)(N+h)} \tag{2-29}$$

$$\frac{1}{T} = \frac{(N+h)-(M+h)}{(M+h)(N+h)}\sin\alpha\cos\alpha \tag{2-30}$$

permits ω_{ew} to be written as

$$\omega_{ew} = \begin{bmatrix} -\dfrac{V_y}{R_y} + \dfrac{V_x}{T} \\ \dfrac{V_x}{R_x} - \dfrac{V_y}{T} \\ 0 \end{bmatrix} \tag{2-31}$$

Note that the italicized symbols R_x and R_y used for the earth radii should not be confused with the components of the position vector **R** denoted by R_x and R_y.

2.4.2 Attitude Computation

An inertial navigation system maintains position and velocity information by solving equations (2-18) through (2-31) given initial values of position and velocity and continuing measurements of specific force expressed in wander azimuth coordinates. Since a strapdown IMU provides specific force measurements $\mathbf{f}^b$ in IMU body coordinates, it is necessary to transform the measurements to wander azimuth coordinates before they can be used in the equations. The direction cosine matrix $\mathbf{C}_b^w$, which transforms from IMU body coordinates to wander azimuth coordinates, is maintained by processing the IMU gyro measurements. The strapdown gyros measure the angular velocity of the IMU body frame with respect to the inertial frame and provide measurements ω_{ib}^b in IMU body coordinates. ω_{ie} and ω_{ew}, after transformation to IMU body coordinates, are subtracted from ω_{ib}^b to obtain the angular velocity of the IMU body frame with respect to the wander azimuth frame

$$\omega_{wb}^b = \omega_{ib}^b - \left[\mathbf{C}_b^w\right]^T (\omega_{ie} + \omega_{ew}) \tag{2-32}$$

which is then integrated (typically using a quaternion based algorithm) to maintain $\mathbf{C}_b^w$. $\mathbf{C}_b^w$ can be expressed in terms of the roll angle ϕ, the pitch angle θ, and the azimuth angle (platform heading) ξ as

$$\mathbf{C}_b^w = \begin{bmatrix} \cos\theta\cos\xi & -\cos\phi\sin\xi + \sin\phi\sin\theta\cos\xi & \sin\phi\sin\xi + \cos\phi\sin\theta\cos\xi \\ -\cos\theta\sin\xi & -\cos\phi\cos\xi - \sin\phi\sin\theta\sin\xi & \sin\phi\cos\xi - \cos\phi\sin\theta\sin\xi \\ \sin\theta & -\sin\phi\cos\theta & -\cos\phi\cos\theta \end{bmatrix} \tag{2-33}$$

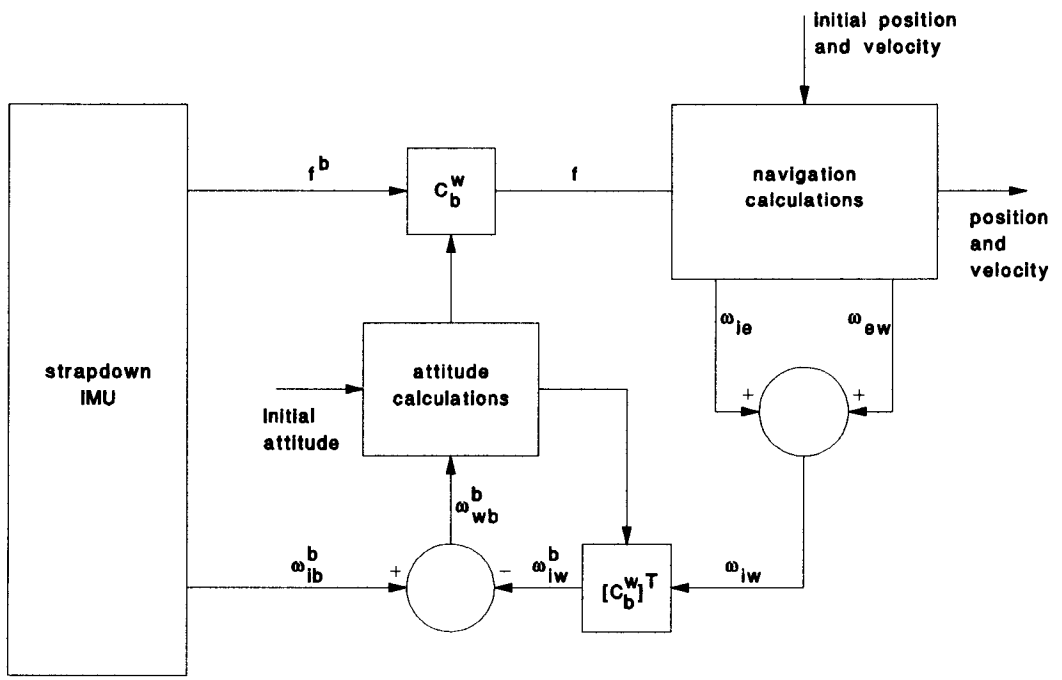


Figure 2-7 Mechanization of Strapdown Inertial Navigation System

and ϕ, θ and ξ can be obtained from the elements of $\mathbf{C}_b^w$ using the equations

$$\tan\phi = \frac{-C_{32}}{-C_{33}} \tag{2-34}$$

$$\tan\theta = \frac{C_{31}}{(C_{11}^2 + C_{21}^2)^{1/2}} \tag{2-35}$$

$$\tan\xi = \frac{-C_{21}}{C_{11}} \tag{2-36}$$

Also, heading relative to north is given by

$$\psi = \xi - \alpha \tag{2-37}$$

Figure 2-7 illustrates the mechanization of a strapdown inertial navigation system performing the computations in wander azimuth coordinates.

2.4.3 Vertical Channel Mechanization

Integrating vertical acceleration without some form of error control using a stable external altitude reference results in an unstable system. This is because altitude errors are reinforced by positive feedback through the gravity calculation. Typically, barometric altitude is used as the reference in a PID (proportional, integral, derivative) controller. The resulting third order baro-inertial loop is shown in Figure 2-8. In the figure, A_z is the total vertical acceleration (including gravity and the Coriolis correction), h_b is the barometric reference altitude, and k_1, k_2 and k_3 are the gains. A typical vertical channel mechanization is shown in Figure 2-9. Note that the vertical velocity is picked off before the derivative feedback through k_1. This results in reduced vertical velocity error. The altitude must track the barometric reference over the long term thus altitude error is determined primarily by the accuracy of the barometric altitude.

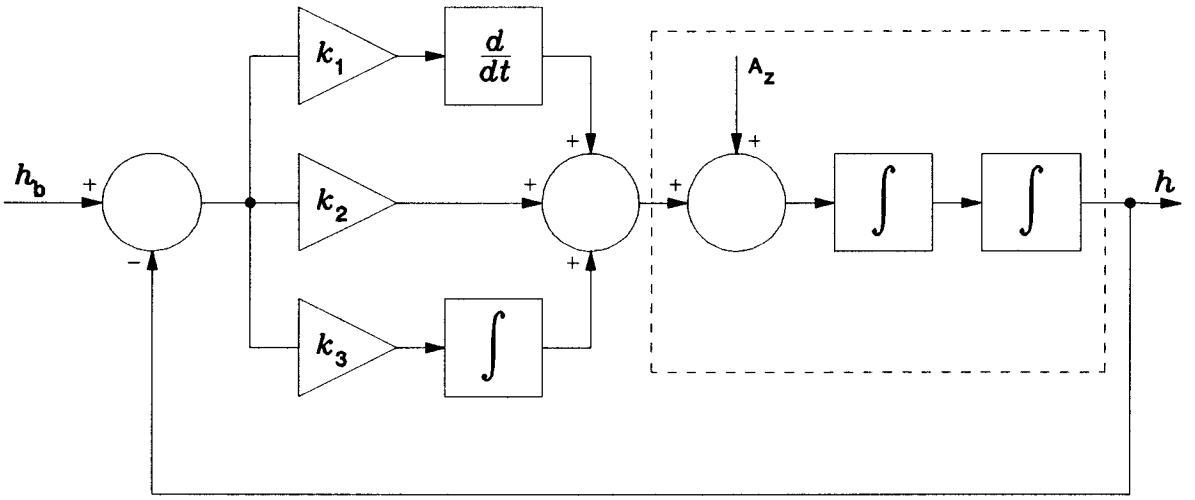


Figure 2-8 Third Order Baro-Inertial Loop

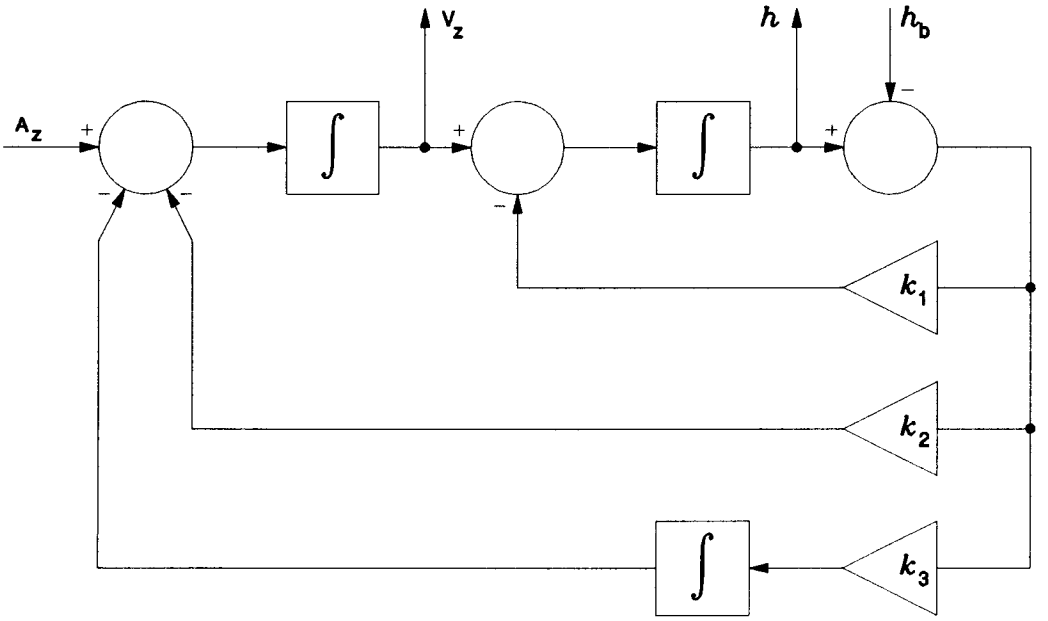


Figure 2-9 Mechanization of Vertical Channel Baro-Inertial Loop

2.5 INERTIAL NAVIGATION ERROR ANALYSIS

The inertial navigation error analysis defines three new frames of reference. These are the computer frame, the true frame and the analytic platform frame. In the absence of errors, these three frames are equivalent. The computer frame refers to a wander azimuth navigation frame constructed at the computed aircraft position. It is related to the earth-fixed frame by the computed latitude, longitude and wander angle. The true frame is related to the earth-fixed frame by the true latitude, longitude and wander angle. Due to errors in the computed values, the computer frame is slightly misaligned with respect to the true frame. Similarly, the analytic platform frame is related to the body frame of the strapdown inertial measurement instruments by the computed roll, pitch and azimuth angles. It is slightly misaligned with respect to the true frame due to errors in the computed angles. Figure 2-10 shows the relationships between the frames.

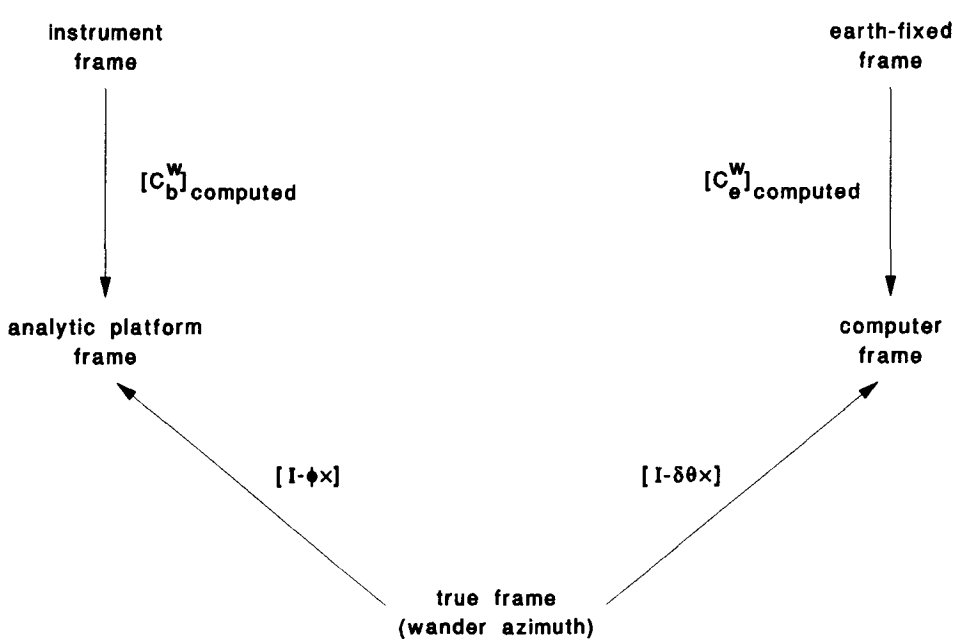


Figure 2-10 Relationships Between Inertial Navigation Error Analysis Frames

The SARMCS Kalman filter uses a perturbation (or true frame) model of the inertial navigation error dynamics. Quantities computed or measured by the navigation system are referred to as indicated quantities and are denoted by enclosure in parentheses followed by the subscript i. For the true frame error model, the relationships between indicated quantities, true quantities and error quantities are defined as follows

$$(\mathbf{R}^e)_i = \mathbf{R}_c^e = \mathbf{C}_t^e(\mathbf{R}+\delta\mathbf{R})$$

$$(\mathbf{V})_i = \mathbf{V}_c^c = \mathbf{V}+\delta\mathbf{V}$$

$$(\mathbf{C}_t^e)_i = \mathbf{C}_c^e = \mathbf{C}_t^e\mathbf{C}_c^t = \mathbf{C}_t^e[\mathbf{I}+\delta\boldsymbol{\theta}\times]$$

$$(\mathbf{C}_b^t)_i = \mathbf{C}_b^p = \mathbf{C}_t^p\mathbf{C}_b^t = [\mathbf{I}+\boldsymbol{\phi}\times]\mathbf{C}_b^t$$

$$(\omega_{ie})_i = \omega_{ie}^c = \omega_{ie}+\delta\omega_{ie}$$

$$(\omega_{et})_i = \omega_{ec}^c = \omega_{et}+\delta\omega_{et}$$

$$(\mathbf{g})_i = \mathbf{g}_c^c = \mathbf{g}+\delta\mathbf{g}$$

$$(\mathbf{f}^b)_i = \mathbf{f}^b+\delta\mathbf{f}^b$$

$$(\omega_{ib}^b)_i = \omega_{ib}^b+\epsilon^b$$

where

$\mathbf{R}_c^e$ = computed position in earth-fixed coordinates

$\mathbf{R}$ = true position in true frame coordinates

$\delta\mathbf{R}$ = position error

$\mathbf{V}_c^c$ = computed velocity in computer frame coordinates

$\mathbf{V}$ = true velocity in true frame coordinates

$\delta\mathbf{V}$ = velocity error

$\mathbf{C}_c^e$ = computer frame to earth-fixed frame direction cosine matrix

$\mathbf{C}_t^e$ = true frame to earth-fixed frame direction cosine matrix

$\mathbf{I}$ = identity matrix

$\delta\boldsymbol{\theta}$ = small angle misalignment of computer frame with respect to true frame

$\mathbf{C}_b^p$ = instrument body frame to analytic platform frame direction cosine matrix

$\mathbf{C}_b^t$ = instrument body frame to true frame direction cosine matrix

$\boldsymbol{\phi}$ = small angle misalignment of analytic platform frame with respect to true frame

$\boldsymbol{\omega}_{ie}^c$ = angular velocity of earth-fixed frame with respect to inertial frame in computer frame coordinates

$\boldsymbol{\omega}_{ie}$ = angular velocity of earth-fixed frame with respect to inertial frame in true frame coordinates

$\delta\boldsymbol{\omega}_{ie}$ = error in indicated angular velocity of earth-fixed frame with respect to inertial frame

$\boldsymbol{\omega}_{ec}^c$ = angular velocity of computer frame with respect to earth-fixed frame in computer frame coordinates

$\boldsymbol{\omega}_{et}$ = angular velocity of true frame with respect to earth-fixed frame in true frame coordinates

$\delta\boldsymbol{\omega}_{et}$ = error in indicated angular velocity of true frame with respect to earth-fixed frame

$\mathbf{g}_c^c$ = computed gravity in computer frame coordinates

$\mathbf{g}$ = true gravity in true frame coordinates

$\delta\mathbf{g}$ = gravity error

$\mathbf{f}^b$ = true specific force in instrument body coordinates

$\delta\mathbf{f}^b$ = accelerometer errors in instrument body coordinates

$\boldsymbol{\omega}_{ib}^b$ = true angular velocity of instrument body frame with respect to inertial frame in instrument body coordinates

$\boldsymbol{\epsilon}^b$ = gyro errors in instrument body coordinates

A derivation of the true frame model of the inertial navigation error dynamics for an arbitrary true navigation frame t is described by Benson (1975) and results in the following equations

$$\delta\dot{\mathbf{R}} = -\omega_{et} \times \delta\mathbf{R} - \mathbf{V} \times \delta\theta + \delta\mathbf{V} \tag{2-38}$$

$$\delta\dot{\mathbf{V}} = -(2\omega_{ie} + \omega_{et}) \times \delta\mathbf{V} + \mathbf{V} \times (2\delta\omega_{ie} + \delta\omega_{et}) + \mathbf{f} \times \boldsymbol{\phi} + \delta\mathbf{f} + \delta\mathbf{g} \tag{2-39}$$

$$\dot{\boldsymbol{\phi}} = -(\omega_{ie} + \omega_{et}) \times \boldsymbol{\phi} + (\delta\omega_{ie} + \delta\omega_{et}) - \epsilon \tag{2-40}$$

Benson expands these equations for a local-level north-pointing navigation frame. However, the SARMCS navigation calculations are performed in a wander azimuth frame. An expansion for the wander azimuth navigation frame is presented in the following paragraphs. This derivation closely follows a derivation by DiFilippo (pers. com. July 10, 1992).

An expression for ω_{ie} is obtained from equation (2-26).

$$\omega_{ie} = \begin{bmatrix} \omega_{ie}\cos\varphi\cos\alpha \\ -\omega_{ie}\cos\varphi\sin\alpha \\ \omega_{ie}\sin\varphi \end{bmatrix} \tag{2-41}$$

Expressions for the x and y components of ω_{et} are obtained from equation (2-31). The z component of ω_{ec}^{c} is, by the definition of the wander azimuth frame, identically equal to zero, therefore, the z component of ω_{et} is $-\delta\omega_{etz}$ (derived later). The complete expression for ω_{et} is

$$\omega_{et} = \begin{bmatrix} -\frac{V_y}{R_y} + \frac{V_x}{T} \\ \frac{V_x}{R_x} - \frac{V_y}{T} \\ -\delta\omega_{etz} \end{bmatrix} \tag{2-42}$$

To obtain an expression for $\delta\boldsymbol{\theta}$, first observe that $\mathbf{C}_e^c$ equals $\mathbf{C}_t^c\mathbf{C}_e^t$ which, for small angle misalignment, is approximated by

$$\mathbf{C}_e^c = [\mathbf{I} - \delta\boldsymbol{\theta}\times]\mathbf{C}_e^t$$

Now, defining the computed latitude, longitude and wander angle as

$$\varphi_c = \varphi + \delta\varphi$$

$$\lambda_c = \lambda + \delta\lambda$$

$$\alpha_c = \alpha + \delta\alpha$$

expressions can be obtained for $\mathbf{C}_e^c$, in terms of φ_c, λ_c and α_c, and $\mathbf{C}_e^t$, in terms of φ, λ and α, by using equation (2-22). Solving for $\delta\boldsymbol{\theta}$, to first order in the error quantities, results in

$$\delta\theta_x = \delta\varphi\sin\alpha + \delta\lambda\cos\varphi\cos\alpha$$

$$\delta\theta_y = \delta\varphi\cos\alpha - \delta\lambda\cos\varphi\sin\alpha$$

$$\delta\theta_z = \delta\alpha + \delta\lambda\sin\varphi$$

Substituting the first order approximations

$$\delta\varphi = \frac{\delta R_x \cos\alpha - \delta R_y \sin\alpha}{M+h} \tag{2-43}$$

and

$$\delta\lambda = \frac{-\delta R_x \sin\alpha - \delta R_y \sin\alpha}{(N+h)\cos\varphi} \tag{2-44}$$

into the expressions for $\delta\theta_x$ and $\delta\theta_y$ gives, after some manipulation,

$$\delta\theta_x = -\frac{\delta R_y}{R_y} + \frac{\delta R_x}{T}$$

$$\delta\theta_y = \frac{\delta R_x}{R_x} - \frac{\delta R_y}{T}$$

Since $1/R_x$ and $1/R_y$ are always much larger than $1/T$, the terms containing $1/T$ can be ignored.

$\delta\theta_z$ defines the azimuth misalignment of the computer frame due to the error in the computed wander angle $\delta\alpha$ and the error in the knowledge of the north direction owing to the error in the computed longitude $\delta\lambda$. Since $\delta\theta_x$ and $\delta\theta_y$ are sufficient to define the level position error, there is a form of redundancy in the azimuth misalignment definition. This redundancy is exploited in the error analysis by defining

$$\delta\theta_z \equiv 0 \tag{2-45}$$

and, hence,

$$\delta\alpha = -\delta\lambda \sin\varphi \tag{2-46}$$

The expression for $\delta\theta$ is then

$$\delta\theta = \begin{bmatrix} -\dfrac{\delta R_y}{R_y} \\ \dfrac{\delta R_x}{R_x} \\ 0 \end{bmatrix} \tag{2-47}$$

Using the definition

$$\delta\omega_{ie} = \omega_{ie}^c - \omega_{ie}$$

and noting that ω_{ie}^c equals $\mathbf{C}_i^c\omega_{ie}$ which, for small angle misalignment, is approximated by $[\mathbf{I}\text{-}\delta\theta\times]\omega_{ie}$, an expression for $\delta\omega_{ie}$ is

$$\delta\omega_{ie} = -\delta\theta \times \omega_{ie}$$

or, using (2-47),

$$\delta\omega_{ie} = \begin{bmatrix} -\dfrac{\omega_{iez}}{R_x}\delta R_x \\ -\dfrac{\omega_{iez}}{R_y}\delta R_y \\ \dfrac{\omega_{iex}}{R_x}\delta R_x + \dfrac{\omega_{iey}}{R_y}\delta R_y \end{bmatrix} \tag{2-48}$$

Expressions for $\delta\omega_{etx}$ and $\delta\omega_{ety}$ are obtained by perturbing ω_{etx} and ω_{ety}. Consider

$$\omega_{etx} = -\frac{V_y}{R_y} + \frac{V_x}{T}$$

which leads to

$$\delta\omega_{etx} = -\frac{1}{R_y}\delta V_y + \frac{V_y}{R_y^2}\delta R_y + \frac{1}{T}\delta V_x - \frac{V_x}{T^2}\delta T$$

For velocities less than 200 metres per second, position errors less than 1000 metres, altitude errors less than 100 metres and velocity errors on the order of 1 metre per second, only the first term is significant and $\delta\omega_{etx}$ is

$$\delta\omega_{etx} = -\frac{1}{R_y}\delta V_y$$

The expression for $\delta\omega_{ety}$ is similarly obtained as

$$\delta\omega_{ety} = \frac{1}{R_x}\delta V_x$$

To find an expression for $\delta\omega_{etz}$, consider the definitions

$$\delta\dot{\theta} = \omega_{tc}^{c} = \omega_{ec}^{c} - \omega_{et}^{c}$$

and

$$\omega_{ec}^{c} = \omega_{et} + \delta\omega_{et}$$

and also observe that ω_{et}^{c} equals $\mathbf{C}_{t}^{c}\omega_{et}$ which, for small angle misalignment, is approximated by $[\mathbf{I}-\delta\boldsymbol{\theta}\times]\omega_{et}$, so that

$$\delta\dot{\boldsymbol{\theta}}=\delta\omega_{et}+\delta\boldsymbol{\theta}\times\omega_{et}$$

The z component of this expression is

$$\delta\dot{\theta}_{z}=\delta\omega_{etz}-\omega_{etx}\delta\theta_{y}+\omega_{ety}\delta\theta_{x}$$

Since $\delta\theta_{z}$ is, by the definition in equation (2-45), identically equal to zero, its derivative is also zero and the expression for $\delta\omega_{etz}$ becomes

$$\delta\omega_{etz}=\omega_{etx}\delta\theta_{y}-\omega_{ety}\delta\theta_{x}$$

or, using (2-47),

$$\delta\omega_{etz}=\frac{\omega_{etx}}{R_{x}}\delta\mathrm{R}_{x}+\frac{\omega_{ety}}{R_{y}}\delta\mathrm{R}_{y}$$

The complete expression for $\delta\omega_{et}$ is then

$$\delta\omega_{et}=\begin{bmatrix}-\frac{1}{R_{y}}\delta\mathrm{V}_{y}\\ \frac{1}{R_{x}}\delta\mathrm{V}_{x}\\ \frac{\omega_{etx}}{R_{x}}\delta\mathrm{R}_{x}+\frac{\omega_{ety}}{R_{y}}\delta\mathrm{R}_{y}\end{bmatrix} \tag{2-49}$$

The gravity error $\delta\mathbf{g}$ is a function of latitude and altitude errors as well as vertical deflections and gravity anomalies. An expression for the gravity error can be obtained using the definition

$$\delta\mathbf{g}=\mathbf{g}_c^c-\mathbf{g}$$

The magnitude of the true gravity vector is g but, due to the small angle vertical deflection $\boldsymbol{\eta}$, the true gravity in true frame coordinates is

$$\mathbf{g}=\left[\mathbf{I}+\boldsymbol{\eta}\times\right]\begin{bmatrix}0\\0\\-g\end{bmatrix}=\begin{bmatrix}-\eta_y g\\ \eta_x g\\ -g\end{bmatrix}$$

The magnitude of the computed gravity vector differs from the true value due to errors in the knowledge of latitude and altitude and due to the gravity anomaly. The gravity anomaly is Δg. Of the remaining gravity error, only that due to the altitude error is significant assuming position errors of less than 1000 metres. Approximating the local gravity using an inverse square law with a gravity value of g at radius $R+h$, the computed gravity can be written as

$$g_c=g+\Delta g-\frac{2g}{R+h}\delta h$$

Replacing $R+h$ with an approximate value equal to the earth equatorial radius a and noting that δh is equal to the vertical position error δR_z, the computed gravity vector in computer frame coordinates is

$$\mathbf{g}_c^c = \begin{bmatrix} 0 \\ 0 \\ -g - \Delta g + \frac{2g}{a}\delta R_z \end{bmatrix}$$

The expression for the gravity error is then

$$\delta \mathbf{g} = \begin{bmatrix} \eta_y g \\ -\eta_x g \\ -\Delta g + \frac{2g}{a}\delta R_z \end{bmatrix} \tag{2-50}$$

Now, substituting (2-42) and (2-47) through (2-50) into equations (2-38) through (2-40), the inertial navigation error dynamics can be expressed in the state-space form

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{w}$$

where the state vector $\mathbf{x}$ is given by

$$\mathbf{x} = \begin{bmatrix} \delta R_x & \delta R_y & \delta R_z & \delta V_x & \delta V_y & \delta V_z & \phi_x & \phi_y & \phi_z \end{bmatrix}^T \tag{2-51}$$

the forcing function vector **w** is given by

$$\mathbf{w}=\begin{bmatrix}0 & 0 & 0 & \delta f_x+\eta_y g & \delta f_y-\eta_x g & \delta f_z-\Delta g & -\epsilon_x & -\epsilon_y & -\epsilon_z\end{bmatrix}^T \tag{2-52}$$

and the system dynamics matrix **F** is as shown in Figure 2-11.

The error analysis presented in this subsection does not include any dynamics due to barometric altitude damping of the vertical channel. Baro-inertial vertical channel equations corresponding to the mechanization shown in Figure 2-9 are

$$\dot{h}=V_z-k_1(h-h_b)$$

$$\dot{V}_z=A_z-k_2(h-h_b)-\Delta A_z$$

$$\Delta\dot{A}_z=k_3(h-h_b)$$

where

h	=	inertial altitude
h_b	=	barometric altitude
V_z	=	vertical velocity
A_z	=	vertical acceleration
ΔA_z	=	estimated vertical acceleration error
k_1, k_2, k_3	=	baro-inertial loop gains

The corresponding vertical channel error dynamics can be obtained by perturbing these equations. However, in Section 3, the vertical channel error states are shown to be unimportant in the Kalman filter formulation and so the results are not included here.

$$
\begin{bmatrix}
\frac{V_z}{R_x} & 0 & -\omega_{ewy} & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{V_z}{R_y} & \omega_{ewx} & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\frac{(2\omega_{iex}+\omega_{ewx})V_y}{R_x} & \frac{2\omega_{iez}V_z+(2\omega_{iey}+\omega_{ewy})V_y}{R_y} & 0 & -\frac{V_z}{R_x} & 2\omega_{iez} & -(2\omega_{iey}+\omega_{ewy}) & 0 & -f_z & f_y \\
-\frac{2\omega_{iez}V_z+(2\omega_{iex}+\omega_{ewx})V_x}{R_x} & -\frac{(2\omega_{iey}+\omega_{ewy})V_x}{R_y} & 0 & -2\omega_{iez} & -\frac{V_z}{R_y} & 2\omega_{iex}+\omega_{ewx} & f_z & 0 & -f_x \\
\frac{2V_y\omega_{iez}}{R_x} & -\frac{2V_x\omega_{iez}}{R_y} & \frac{2g}{a} & 2(\omega_{iey}+\omega_{ewy}) & -2(\omega_{iex}+\omega_{ewx}) & 0 & -f_y & f_x & 0 \\
-\frac{\omega_{iez}}{R_x} & 0 & 0 & 0 & -\frac{1}{R_y} & 0 & 0 & \omega_{iez} & -(\omega_{iey}+\omega_{ewy}) \\
0 & -\frac{\omega_{iez}}{R_y} & 0 & \frac{1}{R_x} & 0 & 0 & -\omega_{iez} & 0 & \omega_{iex}+\omega_{ewx} \\
\frac{\omega_{iex}+\omega_{ewx}}{R_x} & \frac{\omega_{iey}+\omega_{ewy}}{R_y} & 0 & 0 & 0 & 0 & \omega_{iey}+\omega_{ewy} & -(\omega_{iex}+\omega_{ewx}) & 0
\end{bmatrix}
$$

Figure 2-11 Inertial Navigation Error Dynamics Matrix

SECTION THREE

KALMAN FILTER FORMULATION

Optimal estimation of the state of a linear system by Kalman filtering is well known (Gelb 1974; Brown and Hwang 1992). The discrete form of the Kalman filter is used here because of the discrete nature of the available sensor information and because it is well suited to computer implementation. Application of the discrete Kalman filter requires the process dynamics and the measurement relationship to have the form

$$\mathbf{x}_{k+1}=\mathbf{\Phi}_k\mathbf{x}_k+\mathbf{w}_k \tag{3-1}$$

$$\mathbf{z}_k=\mathbf{H}_k\mathbf{x}_k+\mathbf{v}_k \tag{3-2}$$

where

$\mathbf{x}_k$ = $(n\times 1)$ process state vector at time t_k

$\mathbf{\Phi}_k$ = $(n\times n)$ state transition matrix from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$

$\mathbf{w}_k$ = $(n\times 1)$ process noise vector

$\mathbf{z}_k$ = $(m\times 1)$ measurement vector at time t_k

$\mathbf{H}_k$ = $(m\times n)$ measurement matrix relating the measurement to the state at time t_k

$\mathbf{v}_k$ = $(m\times 1)$ measurement noise vector

The process noise $\mathbf{w}_k$ must be white with zero mean and is assumed to be normally distributed with $(n\times n)$ covariance matrix $\mathbf{Q}_k$. Similarly, the measurement noise $\mathbf{v}_k$ must be white with zero mean and is assumed to be normally distributed with $(m\times m)$ covariance matrix $\mathbf{R}_k$.

Inertial navigation error equations have been derived in many references (Benson 1975; Huddle 1983). The result is a nine state system (three position errors, three velocity errors, three misalignments) driven by accelerometer and gyro errors and errors in the computed gravity. By comparing the inertially computed position with the position provided by a GPS receiver, measurements of position error are obtained (referred to as position matching measurements). These process and measurement models form the basis for the current SARMCS Kalman filter.

The antenna boresight calibration Kalman filter adds an antenna azimuth misalignment state corresponding to a random constant process model. Azimuth matching measurements are obtained by comparing the TAU/azimuth encoder measurements with the azimuth computed from the known target location together with GPS-aided inertial position and heading information.

3.1 THE SARMCS KALMAN FILTER

The antenna boresight calibration Kalman filter is intended to be incorporated into the SARMCS as a special operating mode. As such, it is appropriate to formulate the calibration Kalman filter as an extension of the current SARMCS Kalman filter. The current SARMCS Kalman filter formulation is described here. This description is based on a description by DiFilippo (Dec. 23, 1992).

The strapdown IMU, employed in the SARMCS to measure SAR antenna motion, uses small, low-cost gyros that can result in very large misalignment of the IMU navigator analytic platform. Control of this misalignment, so that the IMU navigator provides accurate motion information for SAR motion compensation, is the objective of the SARMCS Kalman filter. The basic process model consists of the inertial navigation error dynamics associated with the INS plus the

inertial navigation error dynamics associated with the IMU navigator. INS/GPS position matching measurements are used to control the INS errors providing a very stable INS platform. A transfer of alignment from the stable INS platform to the IMU navigator is accomplished using IMU/INS position matching measurements. The SARMCS Kalman filter design recognizes that measurements constructed by comparing information from two systems with essentially the same error dynamics allows observability of only the relative error between the systems. The IMU/INS position matching measurements are of this type and consequently the IMU navigator errors are modelled relative to the INS rather than as absolute errors. INS/GPS position matching measurements, however, do allow observability of absolute INS errors. Absolute IMU navigator errors are obtained as the sum of the absolute INS errors plus the relative IMU navigator errors.

There are nine error states for each inertial navigator as described in subsection 2.5, however, since the vertical channel errors are kept small by baroaltitude feedback and since there is little cross coupling between the vertical axis and the level axes, the vertical channel states are not modelled in the SARMCS Kalman filter. The basic SARMCS process model includes two level position error states, two level velocity error states and three misalignment states for each inertial navigator for a total of 14 states.

The inertial navigation error process is driven by accelerometer errors, gyro errors and vertical deflections of the gravity vector. In the SARMCS Kalman filter, INS level accelerometer biases, INS gyro biases and IMU gyro biases are modelled as random processes that are exponentially correlated in time with long correlation times. The process model is augmented to include first-order Markov processes for these errors adding eight new states. Certain other

time-correlated or distance-correlated errors (for example, sensor scale factor errors and misalignments, gyro mass unbalance, and vertical deflections) are only modelled, if at all, as white noise processes, driving the inertial error states, because their effects are too small to warrant modelling as separate states in the Kalman filter.

The errors in the GPS-indicated level position are correlated in time and consequently are modelled as states in the SARMCS Kalman filter. First-order Markov processes with long correlation times are used to model these errors adding two more states to the process model. Other Kalman filter position matching measurement errors, due primarily to timing errors when comparing measurements from different sensors, are modelled as white noise processes.

The complete 24-element state vector of the current SARMCS Kalman filter is shown in Table 3-1. The subvectors $\mathbf{x}_M$ and $\mathbf{x}_S$ contain INS (or master navigator) system states and IMU navigator (or slave navigator) system states respectively, $\mathbf{x}_{INS}$ and $\mathbf{x}_{IMU}$ contain INS and IMU instrument error states, and $\mathbf{x}_{GPS}$ contains GPS position error states. The states in $\mathbf{x}_M$, $\mathbf{x}_{INS}$, $\mathbf{x}_{GPS}$ and $\mathbf{x}_{IMU}$ are defined as indicated or apparent quantities minus true quantities while the states in $\mathbf{x}_S$ are defined as IMU navigator indicated quantities minus INS indicated quantities. For this Kalman filter configuration, the estimated IMU navigation errors must be constructed when needed as $\mathbf{x}_S+\mathbf{x}_M$.

The continuous time process model has the form

$$\dot{\mathbf{x}}(t)=\mathbf{F}(t)\mathbf{x}(t)+\mathbf{w}(t)$$

Table 3-1 SARMCS Kalman Filter States

Subvector	State	Description	Coordinate Frame
$\mathbf{x}_M$	δR_{Mx}	INS position error along x axis	INS wander azimuth
	δR_{My}	INS position error along y axis	
	δV_{Mx}	INS velocity error along x axis	
	δV_{My}	INS velocity error along y axis	
	ϕ_{Mx}	INS platform misalignment about x axis	
	ϕ_{My}	INS platform misalignment about y axis	
	ϕ_{Mz}	INS platform misalignment about z axis	
$\mathbf{x}_{INS}$	A_{Mx}	INS x accelerometer bias	Aircraft body
	A_{My}	INS y accelerometer bias	
	G_{Mx}	INS x gyro bias	
	G_{My}	INS y gyro bias	
	G_{Mz}	INS z gyro bias	
$\mathbf{x}_{GPS}$	δR_{Gx}	GPS position error along x axis	INS wander azimuth
	δR_{Gy}	GPS position error along y axis	
$\mathbf{x}_S$	δR_{Sx}	IMU navigator position error along x axis	IMU navigator wander azimuth
	δR_{Sy}	IMU navigator position error along y axis	
	δV_{Sx}	IMU navigator velocity error along x axis	
	δV_{Sy}	IMU navigator velocity error along y axis	
	ϕ_{Sx}	IMU navigator platform misalignment about x axis	
	ϕ_{Sy}	IMU navigator platform misalignment about y axis	
	ϕ_{Sz}	IMU navigator platform misalignment about z axis	
$\mathbf{x}_{IMU}$	G_{Sx}	IMU x gyro bias	IMU body
	G_{Sy}	IMU y gyro bias	
	G_{Sz}	IMU z gyro bias	

where $\mathbf{x}(t)$ is the 24-element state vector, $\mathbf{F}(t)$ is the 24×24 system dynamics matrix, and the random forcing function $\mathbf{w}(t)$ is a 24-element vector of zero-mean white noise processes. The random forcing function has the same structure as the state vector, namely

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_M \\ \mathbf{w}_{INS} \\ \mathbf{w}_{GPS} \\ \mathbf{w}_S \\ \mathbf{w}_{IMU} \end{bmatrix}$$

The system dynamics matrix has the structure

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_M & \mathbf{F}_{M/INS} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{INS} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{F}_{GPS} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{F}_S & \mathbf{F}_{S/IMU} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{F}_{IMU} \end{bmatrix}$$

where $\mathbf{F}_M$ is a 7×7 matrix, $\mathbf{F}_{INS}$ is a 5×5 matrix, $\mathbf{F}_{M/INS}$ is a 7×5 matrix, $\mathbf{F}_{GPS}$ is a 2×2 matrix, $\mathbf{F}_S$ is a 7×7 matrix, $\mathbf{F}_{IMU}$ is a 3×3 matrix, and $\mathbf{F}_{S/IMU}$ is a 7×3 matrix.

The elements of $\mathbf{F}_M$, shown in Figure 3-1, correspond to the inertial error model expressed in wander azimuth coordinates, as developed in subsection 2.5, but with the vertical channel deleted. $\mathbf{F}_S$ has the same form as $\mathbf{F}_M$ but the quantities apply to the IMU navigator rather than the INS.

$$
\begin{bmatrix}
\frac{V_z}{R_x} & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & \frac{V_z}{R_y} & 0 & 1 & 0 & 0 & 0 \\
\frac{(2\omega_{iex}+\omega_{ewx})V_y}{R_x} & \frac{2\omega_{iez}V_z+(2\omega_{iey}+\omega_{ewy})V_y}{R_y} & -\frac{V_z}{R_x} & 2\omega_{iez} & 0 & -f_z & f_y \\
-\frac{2\omega_{iez}V_z+(2\omega_{iex}+\omega_{ewx})V_x}{R_x} & -\frac{(2\omega_{iey}+\omega_{ewy})V_x}{R_y} & -2\omega_{iez} & -\frac{V_z}{R_y} & f_z & 0 & -f_x \\
-\frac{\omega_{iez}}{R_x} & 0 & 0 & -\frac{1}{R_y} & 0 & \omega_{iez} & -(\omega_{iey}+\omega_{ewy}) \\
0 & -\frac{\omega_{iez}}{R_y} & \frac{1}{R_x} & 0 & -\omega_{iez} & 0 & \omega_{iex}+\omega_{ewx} \\
\frac{\omega_{iex}+\omega_{ewx}}{R_x} & \frac{\omega_{iey}+\omega_{ewy}}{R_y} & 0 & 0 & \omega_{iey}+\omega_{ewy} & -(\omega_{iex}+\omega_{ewx}) & 0
\end{bmatrix}
$$

Figure 3-1 Elements of $\mathbf{F}_M$ and $\mathbf{F}_S$

The dynamics of the error states in $\mathbf{x}_{INS}$, $\mathbf{x}_{GPS}$ and $\mathbf{x}_{IMU}$ are modelled by first-order Markov processes, resulting in the submatrices

$$\mathbf{F}_{INS} = \begin{bmatrix} -\frac{1}{\tau_{MAB}} & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{\tau_{MAB}} & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\tau_{MGB}} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{\tau_{MGB}} & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{\tau_{MGB}} \end{bmatrix}$$

$$\mathbf{F}_{GPS} = \begin{bmatrix} -\frac{1}{\tau_{GP}} & 0 \\ 0 & -\frac{1}{\tau_{GP}} \end{bmatrix}$$

$$\mathbf{F}_{IMU} = \begin{bmatrix} -\frac{1}{\tau_{SGB}} & 0 & 0 \\ 0 & -\frac{1}{\tau_{SGB}} & 0 \\ 0 & 0 & -\frac{1}{\tau_{SGB}} \end{bmatrix}$$

where the variables τ_{MAB}, τ_{MGB}, τ_{GP} and τ_{SGB} represent the correlation times associated with the INS accelerometer bias, INS gyro bias, GPS position error and IMU gyro bias states.

The submatrix $\mathbf{F}_{M/INS}$ has the form

$$\mathbf{F}_{M/INS} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ C_{11} & C_{12} & 0 & 0 & 0 \\ C_{21} & C_{22} & 0 & 0 & 0 \\ 0 & 0 & -C_{11} & -C_{12} & -C_{13} \\ 0 & 0 & -C_{21} & -C_{22} & -C_{23} \\ 0 & 0 & -C_{31} & -C_{32} & -C_{33} \end{bmatrix}$$

where C_{ij} is the element in the ith row and the jth column of the direction cosine matrix $[\mathbf{C}_b^w]_M$ which transforms a vector in aircraft body frame coordinates to INS wander azimuth coordinates, and $[\mathbf{C}_b^w]_M$ is computed using equation (2-33). Similarly, the submatrix $\mathbf{F}_{S/IMU}$ has the form

$$\mathbf{F}_{S/IMU} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -C_{11} & -C_{12} & -C_{13} \\ -C_{21} & -C_{22} & -C_{23} \\ -C_{31} & -C_{32} & -C_{33} \end{bmatrix}$$

where the C_{ij} are elements of the direction cosine matrix $[\mathbf{C}_b^w]_S$ which transforms a vector in IMU body frame coordinates to IMU navigator wander azimuth coordinates, and $[\mathbf{C}_b^w]_S$ is computed using equation (2-33) but with IMU navigator quantities rather than INS quantities.

The vector of continuous time, white noise forcing functions $\mathbf{w}(t)$ is described in terms of its covariance matrix given by

$$E\left[\mathbf{w}(t)\mathbf{w}^T(\tau)\right]=\mathbf{Q}(t)\delta(t-\tau)$$

where $\mathbf{Q}(t)$ is the process noise spectral density matrix and $\delta(t-\tau)$ is the Dirac delta function. The process noise spectral density matrix has the structure

$$\mathbf{Q}=\begin{bmatrix} \mathbf{Q}_M & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{INS} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_{GPS} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{Q}_S & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{Q}_{IMU} \end{bmatrix}$$

Vertical deflections of the gravity vector and several of the less significant errors associated with the INS and IMU sensors are not estimated by the SARMCS Kalman filter, but instead are modelled as motion dependent white noise processes driving the inertial error states $\mathbf{x}_M$ and $\mathbf{x}_S$. The techniques used to determine suitable spectral densities for these white noise processes are not important for the present work and are not examined in detail. Briefly, a white noise process having spectral density q produces an incremental variance in the error state of $q\Delta t$ and a value for q is chosen by matching either the spectral density or the variance to that which would result from the more complex error model under some assumed motion.

The white noise driving the master navigator is described by the spectral density submatrix

$$\mathbf{Q}_M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_\eta + q_{MASF} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_\eta + q_{MASF} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_{MGN} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_{MGN} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q_{MGN} \end{bmatrix}$$

where q_η and q_{MASF} are the spectral densities of the white noise models to account for vertical deflections and INS accelerometer scale factor errors, and q_{MGN} is the spectral density of the white noise associated with the random drift of the INS ring laser gyros.

The white noise driving the slave navigator is described by the spectral density submatrix

$$\mathbf{Q}_S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_{SASF} + q_{MASF} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_{SASF} + q_{MASF} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \begin{matrix} q_{SAB} + q_{SGM} \\ + q_{SGMU} + q_{MGN} \end{matrix} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \begin{matrix} q_{SAB} + q_{SGM} \\ + q_{SGMU} + q_{MGN} \end{matrix} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \begin{matrix} q_{SGSF} + q_{SGMU} \\ + q_{MGN} \end{matrix} \end{bmatrix}$$

where q_{MASF} and q_{MGN} are previously defined and q_{SASF}, q_{SAB}, q_{SGM}, q_{SGMU} and q_{SGSF} are the spectral densities of the white noise models to account for the effects of IMU accelerometer scale factor errors, IMU accelerometer biases, IMU gyro misalignments, IMU gyro mass unbalances and IMU gyro scale factor errors.

The spectral densities appearing in $\mathbf{Q}_M$ and $\mathbf{Q}_S$ are given by the expressions

$$q_\eta = \frac{2d_\eta g^2}{V_{gnd}} \sigma_\eta^2$$

$$q_{MASF} = \frac{4V_{gnd} a}{\pi} \sigma_{MASF}^2$$

$$q_{MGN} = \frac{\sigma_{MGN}^2(t)}{t} = \text{constant}$$

$$q_{SASF} = \frac{4V_{gnd} a}{\pi} \sigma_{SASF}^2$$

$$q_{SAB} = \frac{4|\omega_z|}{\pi g} \sigma_{SAB}^2$$

$$q_{SGM} = \frac{4|\omega_z|}{\pi}\sigma_{SGM}^2$$

$$q_{SGMU} = \frac{4V_{gnd}a}{\pi}\sigma_{SGMU}^2$$

$$q_{SGSF} = \pi|\omega_z|\sigma_{SGSF}^2$$

where

σ_η = standard deviation of the deflection of the vertical

σ_{MASF} = standard deviation of the INS accelerometer scale factor error

$\sigma_{MGN}(t)$ = time varying standard deviation of the INS gyro random drift (random walk process implies standard deviation increases as the square root of time)

σ_{SASF} = standard deviation of the IMU accelerometer scale factor error

σ_{SAB} = standard deviation of the IMU accelerometer bias

σ_{SGM} = standard deviation of the IMU gyro misalignments with respect to the accelerometers

σ_{SGMU} = standard deviation of the IMU gyro mass unbalance

σ_{SGSF} = standard deviation of the IMU gyro scale factor error

V_{gnd} = aircraft ground speed

a = magnitude of aircraft acceleration vector

$|\omega_z|$ = absolute value of the aircraft heading rate

g = nominal gravity

d_η = correlation distance of random deflections of the vertical

White noise drives the first-order Markov sensor error models, resulting in the process noise spectral density submatrices

$$\mathbf{Q}_{INS} = \begin{bmatrix} 2\frac{\sigma_{MAB}^2}{\tau_{MAB}} & 0 & 0 & 0 & 0 \\ 0 & 2\frac{\sigma_{MAB}^2}{\tau_{MAB}} & 0 & 0 & 0 \\ 0 & 0 & 2\frac{\sigma_{MGB}^2}{\tau_{MGB}} & 0 & 0 \\ 0 & 0 & 0 & 2\frac{\sigma_{MGB}^2}{\tau_{MGB}} & 0 \\ 0 & 0 & 0 & 0 & 2\frac{\sigma_{MGB}^2}{\tau_{MGB}} \end{bmatrix}$$

$$\mathbf{Q}_{GPS} = \begin{bmatrix} 2\frac{\sigma_{GP}^2}{\tau_{GP}} & 0 \\ 0 & 2\frac{\sigma_{GP}^2}{\tau_{GP}} \end{bmatrix}$$

$$\mathbf{Q}_{IMU} = \begin{bmatrix} 2\frac{\sigma_{SGB}^2}{\tau_{SGB}} & 0 & 0 \\ 0 & 2\frac{\sigma_{SGB}^2}{\tau_{SGB}} & 0 \\ 0 & 0 & 2\frac{\sigma_{SGB}^2}{\tau_{SGB}} \end{bmatrix}$$

where the variables σ_{MAB}, σ_{MGB}, σ_{GP} and σ_{SGB} represent the standard deviations associated with the INS accelerometer bias, INS gyro bias, GPS position error and IMU gyro bias states.

The SARMCS Kalman filter uses INS/GPS position matching measurements to estimate INS position errors, velocity errors and misalignments, and IMU/INS position matching measurements to accomplish the transfer of alignment to the IMU navigator. Computation of the measurement vectors involves quantities having the same or similar definitions but determined separately in each of the three navigation systems (the INS, the GPS and the IMU navigator). Quantities associated with a particular navigation system are indicated here by enclosure in square brackets followed by a subscript *M*, *GPS* or *S*. *M* refers to INS (master navigator) quantities, *GPS* refers to GPS quantities, and *S* refers to IMU navigator (slave navigator) quantities.

The INS/GPS position matching measurements are obtained by subtracting the GPS-indicated position vector from the INS-indicated position vector and adjusting for the INS-to-GPS lever arm. The measurement vector $\mathbf{z}_G$ is computed as

$$\mathbf{z}_G = \left[\mathbf{C}_w^e\right]_M^T \left(\left[\mathbf{R}^e\right]_M - \left[\mathbf{R}^e\right]_{GPS}\right) + \left[\mathbf{C}_b^w\right]_M \left[\mathbf{r}_{GPS}^b\right]_M$$

where

$[\mathbf{R}^e]_M$ = INS position vector expressed in earth-fixed coordinates

$[\mathbf{R}^e]_{GPS}$ = GPS position vector expressed in earth-fixed coordinates

$[\mathbf{C}_w^e]_M$ = direction cosine matrix that transforms a vector in INS wander azimuth coordinates to earth-fixed coordinates

$[\mathbf{C}_b^w]_M$ = direction cosine matrix that transforms a vector in aircraft body coordinates to INS wander azimuth coordinates

$[\mathbf{r}_{GPS}^b]_M$ = INS-to-GPS lever arm vector expressed in aircraft body coordinates

The IMU/INS position matching measurements are obtained by subtracting the INS-indicated position vector from the IMU navigator position vector and adjusting for the INS-to-IMU lever arm. The measurement vector $\mathbf{z}_P$ is computed as

$$\mathbf{z}_P = \left[\mathbf{C}_w^e\right]_S^T \left(\left[\mathbf{R}^e\right]_S - \left[\mathbf{R}^e\right]_M\right) - \left[\mathbf{C}_w^e\right]_S^T \left[\mathbf{C}_w^e\right]_M \left[\mathbf{C}_b^w\right]_M \left[\mathbf{r}_{IMU}^b\right]_M$$

where

$[\mathbf{R}^e]_S$ = IMU navigator position vector expressed in earth-fixed coordinates

$[\mathbf{C}_w^e]_S$ = direction cosine matrix that transforms a vector in IMU navigator wander azimuth coordinates to earth-fixed coordinates

$[\mathbf{r}_{IMU}^b]_M$ = INS-to-IMU lever arm vector expressed in aircraft body coordinates

The SARMCS Kalman filter does not process the vertical components of $\mathbf{z}_G$ and $\mathbf{z}_P$ but only the horizontal components. Therefore, the Kalman filter measurement vector is given by

$$\mathbf{z} = \begin{bmatrix} z_{G_x} \\ z_{G_y} \\ z_{P_x} \\ z_{P_y} \end{bmatrix}$$

The linear model for the discrete Kalman filter measurements is

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

where $\mathbf{H}_k$ is the 4×24 measurement matrix and $\mathbf{v}_k$ is a 4-element vector of zero-mean, white noise measurement errors. The measurement matrix is obtained by perturbing the $\mathbf{z}_G$ and $\mathbf{z}_P$ measurement equations and expressing the results in terms of the modelled error states. Making the assumptions that the effect of errors in computing the lever arms can be neglected and that $[\mathbf{C}_w^e]_M$ and $[\mathbf{C}_w^e]_S$ are approximately equal, the measurement matrix is

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{H}_{G/M} & \mathbf{0} & \mathbf{H}_{G/GPS} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{H}_{P/S} & \mathbf{0} \end{bmatrix}$$

where

$$\mathbf{H}_{G/M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{H}_{G/GPS} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\mathbf{H}_{P/S} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The measurement noise vector $\mathbf{v}_k$ has the same form as the measurement vector, namely

$$\mathbf{v}_k = \begin{bmatrix} \mathrm{v}_{G_x} \\ \mathrm{v}_{G_y} \\ \mathrm{v}_{P_x} \\ \mathrm{v}_{P_y} \end{bmatrix}$$

and is characterized by its covariance matrix $\mathbf{R}_k$ given by

$$\mathbf{R}_k = E\left[\mathbf{v}_k \mathbf{v}_k^T\right] = \begin{bmatrix} \sigma_G^2 & 0 & 0 & 0 \\ 0 & \sigma_G^2 & 0 & 0 \\ 0 & 0 & \sigma_P^2 & 0 \\ 0 & 0 & 0 & \sigma_P^2 \end{bmatrix}$$

where σ_G and σ_P are the standard deviations of the INS/GPS position matching measurement noise and the IMU/INS position matching measurement noise. The major contributor to these measurement noises is timing errors that result when comparing measurements from the different sensors.

To begin the Kalman filter estimation process, the state vector and the error covariance matrix must be initialized. The SARMCS Kalman filter initial state vector estimate is $\hat{\mathbf{x}}(0) = \mathbf{0}$. The initial error covariance matrix $\mathbf{P}(0)$ is diagonal and contains the variances of the errors in the initial state estimates.

Finally, it should be noted that the SARMCS Kalman filter makes use of the fact that certain parts of the system state vector are only weakly coupled. In particular, the states $[\mathbf{x}_M\ \mathbf{x}_{INS}\ \mathbf{x}_{GPS}]^T$, corresponding to the INS stabilization part of the filter, and $[\mathbf{x}_S\ \mathbf{x}_{IMU}]^T$, corresponding to the transfer-of-alignment part of the filter, are decoupled in the SARMCS design. This is achieved by neglecting 1) a weak coupling between $\mathbf{x}_{INS}$ and $\mathbf{x}_S$ in the system dynamics matrix, 2) a small correlation between $\mathbf{w}_M$ and $\mathbf{w}_S$ in the spectral density matrix, and 3) a small initial cross-covariance between $\mathbf{x}_M$ and $\mathbf{x}_S$. The effect on the estimation of IMU slave navigator error states is minimal because of the weak coupling. It has the benefit, however, of preventing corruption of the INS master navigator error states that, otherwise, could result from slight mismodelling of the lower quality IMU.

3.2 ANTENNA AZIMUTH MISALIGNMENT ESTIMATION

Accurate antenna azimuth misalignment estimation depends on accurate IMU navigator position and heading information. IMU navigator position error is bounded by the GPS position measurements while control of the IMU navigator heading error is one of the objectives of the current SARMCS Kalman filter. Consequently, the SARMCS Kalman filter can be modified to estimate the antenna azimuth misalignment by adding a new state, ε, and a new measurement, z_A. The new state models the antenna azimuth misalignment as a random constant

$$\dot{\varepsilon}=0 \tag{3-3}$$

and the new measurement of antenna azimuth misalignment is given by

$$z_A = \vartheta_{TAU} - \vartheta_C \tag{3-4}$$

where ϑ_{TAU} is the TAU-determined antenna azimuth and ϑ_C is the computed value based on the IMU navigator position and heading and the known target location.

Modelling of a state by a random constant is often considered risky. This is because, with no process noise, the Kalman filter begins to rely on the process model after a few measurements and will not follow any small or slow variations which may occur in the true state. Estimation of the antenna azimuth misalignment is expected to be completed relatively quickly, however, and so the random constant model is reasonable here. Should testing reveal problems, a small amount of process noise can be added to produce a random walk model. Alternatively, an exponentially correlated Markov process with a very long correlation time could be used to model the antenna misalignment.

The linear measurement model corresponding to (3-2) is obtained by perturbing (3-4). First, ϑ_{TAU} and ϑ_C are written as

$$\vartheta_{TAU} = \vartheta + \varepsilon + v_A$$

$$\vartheta_C = \vartheta + \delta\vartheta$$

where ϑ is the true antenna azimuth, ε is the antenna azimuth misalignment, v_A is the noise associated with the TAU/azimuth encoder measurement and $\delta\vartheta$ is the error in the computed antenna azimuth. The azimuth matching measurement is then

$$z_A = \vartheta_{TAU} - \vartheta_C = \varepsilon - \delta\vartheta + v_A$$

Using $\vartheta = \beta - \psi$ and $\psi = \xi - \alpha$ the measurement can be re-written as

$$z_A = \varepsilon - \delta\beta + \delta\xi - \delta\alpha + v_A \tag{3-5}$$

It remains to express $\delta\beta$, $\delta\xi$ and $\delta\alpha$ in terms of the defined states. In subsection 2.3, bearing is given by

$$\beta = \arctan\left[\frac{x}{y}\right]$$

where x and y are computed using (2-14) through (2-16). Perturbing this equation yields

$$\delta\beta = \frac{y}{x^2+y^2}\delta x - \frac{x}{x^2+y^2}\delta y$$

Within the operating limits on aircraft altitude and TAU range, δx is approximately equal to the west position error and δy is approximately equal to the south position error (see Figure 2-4) so that $\delta\beta$ can be written in terms of the position errors in wander azimuth coordinates as

$$\delta\beta = \frac{y}{x^2+y^2}(\delta R_x \sin\alpha + \delta R_y \cos\alpha) - \frac{x}{x^2+y^2}(-\delta R_x \cos\alpha + \delta R_y \sin\alpha) \tag{3-6}$$

Note that target position errors are assumed negligible and IMU position errors are "corrected" IMU navigator quantities and consequently are small (approximately equal to the GPS position errors). Since the target range is typically tens of thousands of metres, the ratio of position error to range is on the order of 1/1000 making the linearized $\delta\beta$ approximation very good.

An expression for $\delta\xi$ is obtained by considering that

$$\mathbf{C}_b^p = \mathbf{C}_t^p \mathbf{C}_b^t = [\mathbf{I} - \boldsymbol{\phi}\times]\mathbf{C}_b^t$$

where $\mathbf{C}_b^t$ is the direction cosine matrix that relates the IMU body frame to the true wander azimuth navigation frame (defined by the true roll, pitch and azimuth: ϕ, θ and ξ) and $\mathbf{C}_b^p$ is the direction cosine matrix that relates the IMU body frame to the analytic platform frame (defined by the computed roll, pitch and azimuth: $\phi+\delta\phi$, $\theta+\delta\theta$ and $\xi+\delta\xi$). Solving for $\delta\xi$ results in the desired expression

$$\delta\xi = \phi_z - (\phi_x \cos\xi - \phi_y \sin\xi)\tan\theta \tag{3-7}$$

An expression for $\delta\alpha$ was found in subsection 2.5 and given in equation (2-46) which is repeated here for convenience

$$\delta\alpha = -\delta\lambda \sin\varphi$$

Substituting for $\delta\lambda$ in terms of the position errors, as given in equation (2-44), the desired expression for $\delta\alpha$ is

$$\delta\alpha = \left[\frac{\delta R_x \sin\alpha + \delta R_y \cos\alpha}{N+h} \right] \tan\varphi \tag{3-8}$$

Recalling that the IMU navigator error quantities in (3-6) through (3-8) must be constructed from $\mathbf{x}_S + \mathbf{x}_M$ and substituting the resulting equations into (3-5), the antenna azimuth misalignment measurement model is obtained. Since the TAU/azimuth encoder measurement error v_A is modeled as white noise, the measurement model is in the form required by the Kalman filter. The measurement model can be simplified by noting that the IMU is roll and pitch stabilized so the terms in (3-7) which are multiplied by tan θ are negligible and can be dropped. The resulting measurement model is

$$\begin{aligned} z_A = & \left[\frac{-x\cos\alpha - y\sin\alpha}{x^2+y^2} - \frac{\tan\varphi \sin\alpha}{N+h} \right] (\delta R_{Sx} + \delta R_{Mx}) \\ & + \left[\frac{x\sin\alpha - y\cos\alpha}{x^2+y^2} - \frac{\tan\varphi \cos\alpha}{N+h} \right] (\delta R_{Sy} + \delta R_{My}) + (\phi_{Sz} + \phi_{Mz}) + \varepsilon + v_A \end{aligned} \tag{3-9}$$

The complete antenna azimuth calibration Kalman filter process and measurement models are now obtained by adding ε and z_A to the SARMCS Kalman filter described in subsection 3.1. The calibration Kalman filter 25-element state vector has the form

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_M \\ \mathbf{x}_{INS} \\ \mathbf{x}_{GPS} \\ \mathbf{x}_S \\ \mathbf{x}_{IMU} \\ \mathbf{x}_A \end{bmatrix}$$

where the subvectors $\mathbf{x}_M$, $\mathbf{x}_{INS}$, $\mathbf{x}_{GPS}$, $\mathbf{x}_S$ and $\mathbf{x}_{IMU}$ are defined in subsection 3.1, and $\mathbf{x}_A = \varepsilon$. Since the new state models a random constant, there is no associated dynamics or process noise and the corresponding elements of the calibration Kalman filter system dynamics matrix **F**, process noise vector **w**, and process noise spectral density matrix **Q** are zeros.

The calibration Kalman filter 5-element measurement vector has the form

$$\mathbf{z}_k = \begin{bmatrix} z_{G_x} \\ z_{G_y} \\ z_{P_x} \\ z_{P_y} \\ z_A \end{bmatrix}$$

where the z_G and z_P elements are defined in subsection 3.1.

The measurement noise vector has the same form as the measurement vector, namely

$$\mathbf{v}_k = \begin{bmatrix} v_{G_x} \\ v_{G_y} \\ v_{P_x} \\ v_{P_y} \\ v_A \end{bmatrix}$$

and the measurement noise covariance matrix is given by

$$\mathbf{R}_k = \begin{bmatrix} \sigma_G^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_G^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_P^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma_P^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma_A^2 \end{bmatrix}$$

where v_G, v_P, σ_G and σ_P are defined in subsection 3.1 and σ_A is the standard deviation of the azimuth matching measurement noise due to the TAU/azimuth encoder measurement noise v_A.

Finally, the measurement matrix $\mathbf{H}_k$ has the form

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{H}_{G/M} & \mathbf{0} & \mathbf{H}_{G/GPS} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{H}_{P/S} & \mathbf{0} & \mathbf{0} \\ \mathbf{H}_{A/M} & \mathbf{0} & \mathbf{0} & \mathbf{H}_{A/S} & \mathbf{0} & \mathbf{H}_{A/A} \end{bmatrix}$$

where the submatrices $\mathbf{H}_{G/M}$, $\mathbf{H}_{G/GPS}$ and $\mathbf{H}_{P/S}$ are defined in subsection 3.1 and, from (3-9), the submatrices $\mathbf{H}_{A/M}$ and $\mathbf{H}_{A/S}$ are given by

$$\mathbf{H}_{A/M} = \mathbf{H}_{A/S} = \begin{bmatrix} \frac{-x\cos\alpha - y\sin\alpha}{x^2+y^2} - \frac{\tan\varphi\sin\alpha}{N+h} & \frac{x\sin\alpha - y\cos\alpha}{x^2+y^2} - \frac{\tan\varphi\cos\alpha}{N+h} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and $\mathbf{H}_{A/A}$ is

$$\mathbf{H}_{A/A} = 1$$

3.3 DISCRETE TIME COMPUTATION

For digital computer implementation, the discrete form of the Kalman filter is most appropriate. The computations can be divided into extrapolation of the state vector and error covariance matrix using the process model, followed by an update using the measurement data and the measurement model. The discrete time Kalman filter extrapolation equations are

$$\hat{\mathbf{x}}_k^- = \mathbf{\Phi}_{k-1}\hat{\mathbf{x}}_{k-1}^+ \tag{3-10}$$

$$\mathbf{P}_k^- = \mathbf{\Phi}_{k-1}\mathbf{P}_{k-1}^+\mathbf{\Phi}_{k-1}^T + \mathbf{Q}_{k-1} \tag{3-11}$$

where $\hat{\mathbf{x}}_k$ is the estimated state vector, $\mathbf{P}_k$ is the error covariance matrix and the minus and plus superscripts indicate values prior to incorporation of the measurement and after incorporation of the measurement, respectively. The discrete time Kalman filter update equations are

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\left[\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-\right] \tag{3-12}$$

$$\mathbf{P}_k^+ = \left[\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right]\mathbf{P}_k^- \tag{3-13}$$

where $\mathbf{K}_k$ is the Kalman gain matrix given by

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T\left[\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k\right]^{-1} \tag{3-14}$$

The discrete time formulation requires a process model having the form of equation (3-1) and a measurement model having the form of equation (3-2). In subsection 3.2, the measurement model is given in discrete form but the process model is in continuous form. One method for computing the discrete time state transition matrix $\mathbf{\Phi}_k$ and process noise covariance matrix $\mathbf{Q}_k$ from the continuous time system dynamics matrix $\mathbf{F}(t)$ and process noise spectral density matrix $\mathbf{Q}(t)$ is described here.

A discrete time processing interval of ten seconds is used for the current SARMCS Kalman filter. Since the modification for antenna azimuth calibration has not introduced any short time-constant dynamics, a time interval of comparable magnitude is also appropriate for the calibration Kalman filter. This time interval is very much less than the system time constants

(ie., 5000 seconds for the Schuler period and typically greater than 10,000 seconds for the exponentially correlated processes) and is generally much less than the time required for significant changes in the time varying system dynamics (ie., no high rate manoeuvres). $\mathbf{\Phi}_k$ and $\mathbf{Q}_k$ can, therefore, be approximated by Taylor series expansions having only a few terms.

If the state transition matrix from time τ to time t is $\mathbf{\Phi}(t,\tau)$ and the process noise covariance matrix is $\mathbf{Q}_k(t)$ then the differential equations describing $\mathbf{\Phi}_k$ and $\mathbf{Q}_k$ (Gelb 1974) can be written as

$$\frac{d\mathbf{\Phi}(t,\tau)}{dt} = \mathbf{F}(t)\mathbf{\Phi}(t,\tau) \ , \qquad \mathbf{\Phi}(t,t) = \mathbf{I}$$

$$\frac{d\mathbf{Q}_k(t)}{dt} = \mathbf{F}(t)\mathbf{Q}_k(t) + \mathbf{Q}_k(t)\mathbf{F}^T(t) + \mathbf{Q}(t) \ , \qquad \mathbf{Q}_k(0) = \mathbf{0}$$

Using these equations, the truncated Taylor series approximation for $\mathbf{\Phi}_k$ can be found as

$$\mathbf{\Phi}(t+\Delta t,t) \approx \sum_{n=0}^{N_\phi} \frac{\Delta t^n}{n!}\mathbf{F}^n \tag{3-15}$$

where $\mathbf{F}$ is assumed constant over the interval Δt and $N_\Phi + 1$ is the number of terms in the series. Similarly, the process noise covariance matrix $\mathbf{Q}_k$ is computed from the process noise spectral density matrix $\mathbf{Q}$ using the truncated Taylor series approximation

$$\mathbf{Q}_k \approx \sum_{n=1}^{N_Q} \frac{\Delta t^n}{n!} \sum_{m=1}^{n} \binom{n-1}{m-1} \mathbf{F}^{n-m} \mathbf{Q} (\mathbf{F}^T)^{m-1} \tag{3-16}$$

where both $\mathbf{F}$ and $\mathbf{Q}$ are assumed constant over the interval Δt and $N_Q + 1$ is the number of terms in the series. The coefficients $\binom{n-1}{m-1}$ are binomial coefficients.

3.4 FEEDBACK CONFIGURATION

The high drift IMU gyros can lead to large navigation errors in the IMU inertial navigator. Since the linear inertial navigation error model assumes small errors, error control feedback is employed.

Consider a system whose error dynamics are described by (3-1) and assume that aiding sources provide for measurements of the system output errors according to (3-2). The objective is to estimate the errors and correct the system outputs using a feedback configuration. To do this, the system outputs are corrected at each discrete Kalman filter update time t_k using the best available Kalman filter estimated errors. In a real-time implementation, the best available estimate of the error state vector at time t_k is $\hat{\mathbf{x}}_{k-1}$ since the calculation of $\mathbf{\Phi}_{k-1}$ and the incorporation of the measurements $\mathbf{z}_k$ are still in progress. The corrected system error dynamics and measurements are given by

$$\mathbf{x}_k = \mathbf{\Phi}_{k-1}\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1} + \mathbf{w}_{k-1}$$

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

The corresponding Kalman filter extrapolation and update equations become

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

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\left[\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-\right]$$

where the measurements $\mathbf{z}_k$ are obtained after the corrections are applied to the system outputs.

In applying the foregoing results to the antenna boresight calibration Kalman filter, only the $\mathbf{x}_S$ and $\mathbf{x}_{IMU}$ states are affected since feedback is only applied to the IMU navigation quantities and the IMU gyro bias calibration quantities. Furthermore, the estimates used to correct the IMU navigation quantities are obtained from $\mathbf{x}_S + \mathbf{x}_M$ rather than $\mathbf{x}_S$ alone. Table 3-2 lists the operations performed at each Kalman filter update time t_k.

Table 3-2 Operations Performed at Each Kalman Filter Update Time

1. Correct the IMU navigation quantities using $\mathbf{x}_S+\mathbf{x}_M$ and the IMU gyro bias quantities using $\mathbf{x}_{IMU}$.

2. Construct $\mathbf{z}_k$, $\mathbf{H}_k$ and $\mathbf{R}_k$ (using corrected quantities).

3. Construct $\mathbf{F}$ and $\mathbf{Q}$ (using corrected quantities).

4. Compute $\mathbf{\Phi}_{k-1}$ and $\mathbf{Q}_{k-1}$ from $\mathbf{F}$ and $\mathbf{Q}$.

5. Extrapolate using

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

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

6. Subtract the corrections applied in step 1 from the $\mathbf{x}_S$ and $\mathbf{x}_{IMU}$ subvectors.

7. Update using

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T\left[\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k\right]^{-1}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\left[\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-\right]$$

$$\mathbf{P}_k^+ = \left[\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right]\mathbf{P}_k^-$$

SECTION FOUR

SIMULATION SOFTWARE

A computer simulation software package was developed to analyze the antenna boresight calibration Kalman filter. The simulation software is coded in the C++ language and runs on a personal computer under the DOS operating system. It is divided into a synthesis package, a processing package and an evaluation package.

The synthesis package consists of programs to generate a reference (error free) trajectory and synthesized sensor data. The package is designed to make specification of the flight profile and generation of the corresponding synthesized sensor data a simple and comparatively quick process to facilitate experimentation. This complements the SARMCS synthesis package which more accurately simulates the sensor environment and sensor errors but at the cost of considerably more set-up effort and run time.

The processing package performs the Kalman filter estimation of the antenna azimuth misalignment using the synthesized sensor data as input. During processing, various intermediate results of interest are saved to files for subsequent analysis. The package can also process data that has been generated using the SARMCS synthesis package or has been recorded from real sensors during flight trials of the SARMCS.

The evaluation package facilitates reading of the generated data into MATLAB for plotting and analysis. MATLAB is a commercially available product for scientific and engineering numeric computation and graphing.

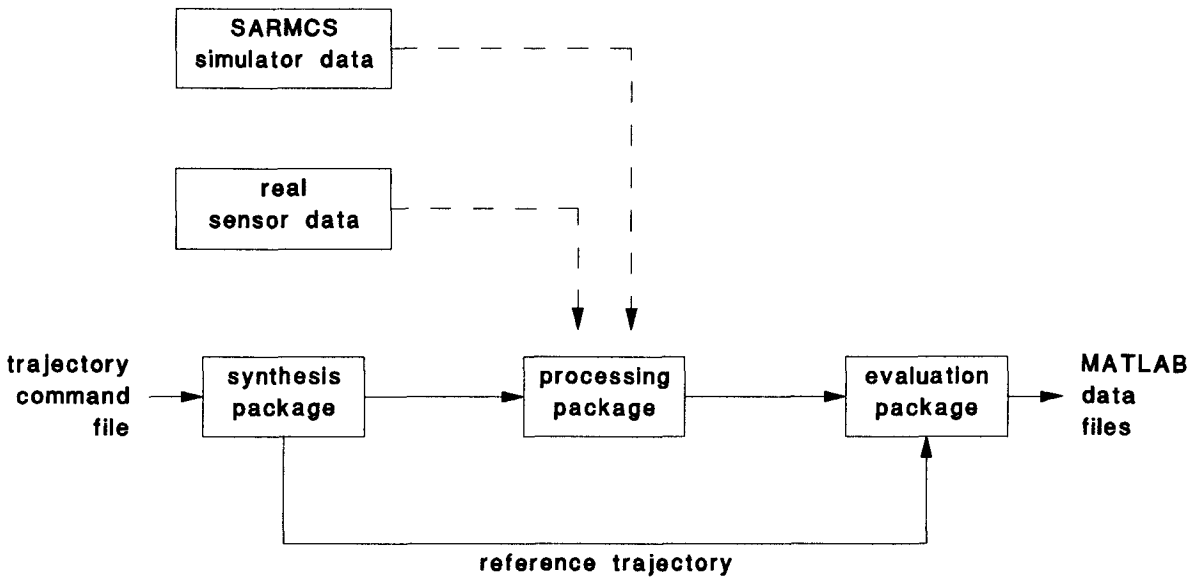


Figure 4-1 Block Diagram of Simulation Package

Figure 4-1 shows the relationship between the synthesis package, the processing package and the evaluation package.

4.1 TRAJECTORY GENERATION

The trajectory generator consists of two programs called `TRJO` and `TRJC`. Program `TRJO` reads a file of English text commands describing the flight and produces a file of records containing position (earth-fixed x, y, z coordinates) and attitude (roll, pitch, heading) sampled at discrete time intervals. Each record also contains the sample time beginning at time zero for the first record. The input command file has a default file name extension of `.CMD`. The output trajectory file takes a default file name from the command file name and applies a default file name extension of `.TJO`.

Program TRJC reads the trajectory file and produces a file of coefficients for cubic spline interpolation of the position and attitude. The trajectory interpolation coefficient file takes a default file name from the trajectory file name and has a default file name extension of .TJC. As an example, given a trajectory generation command file named TEST.CMD, the following sequence of DOS commands generate a trajectory file named TEST.TJO and a trajectory interpolation coefficient file named TEST.TJC:

```
TRJO TEST
TRJC TEST
```

4.1.1 The Command File

The trajectory generation command file is produced using a text editor. A typical command file is shown in Table 4-1.

Following is an example of a typical command:

```
at 100 change altitude to 1000 in 100;
```

In general, a command consists of a condition part (at 100) an action part (change altitude to 1000), a duration part (in 100) and a terminator (;). The condition part specifies the time at which the action is to be initiated, relative to the start of the simulation or to the most recently defined reference mark. The action part defines the action to be performed. The duration part specifies the duration over which the action is performed.

Table 4-1 Typical Trajectory Generation Command File

```
% TEST.CMD
% Test command file to generate a 5020 second trajectory.


% Initial stationary position at beginning of runway.

initialize position to (49:12:00,-123:45:00);
initialize altitude to 0;
initialize groundspeed to 0;
initialize track to 45;
at 10 mark;


% Takeoff and initial climb.

at 0 change groundspeed to 120 in 20;
at 20 change altitude to 1000 in 100;
at 50 change groundspeed to 200 in 100;
at 120 change altitude to 5000 in 1000;
mark;


% Manoeuvres at altitude.

at 70 change track by 180 in 60;
at 400 change track by -90 in 30;


% End.
at 3890 end;
```

Every command must include an action part but not all actions support a condition part and a duration part. When supported, the condition and duration parts are optional. When a condition part is supported but not specified, the action is initiated when all previous actions are completed. When a duration part is supported but not specified, the action is performed in a predetermined default duration of nominal value.

There are four action types identified by the action keywords **initialize**, **change**, **mark** and **end**, and there are two action modifiers identified by the keywords **to** and **by**. When present, the condition part is introduced by the keyword **at** and the duration part is introduced by the keyword **in**. Allowed commands take one of the following forms:

```
initialize variable to value;
[at time] change variable to¦by value [in time];
[at time] mark;
[at time] end;
```

where the square brackets indicate an optional part and the vertical bar indicates a choice.

Initialize commands set the initial values of the navigation variables and must appear before any other commands. Change commands describe the dynamics of the flight by specifying how the navigation variables change. Either absolute (`to`) or relative (`by`) change values may be specified. The mark command allows the reference time to be changed. This permits command sequences for special manoeuvres to be reused. The end command indicates the end of the trajectory generation. It must appear exactly once at the end of the command file.

In addition to the time variable, there are four action variables whose identifiers are: **position**, **altitude**, **track** and **groundspeed**. Time values are always specified in seconds, horizontal position values are specified as latitude/longitude pairs in units of degrees, minutes and seconds, altitude is specified in feet, track angle is in units of degrees and ground speed is specified in nautical miles per hour.

Comments may also be included in the command file. The percent character % introduces a comment. All characters between the percent sign and the end of the line are ignored by the command parser.

4.1.2 Trajectory Calculation

The generated trajectory is defined by a sequence of position and attitude samples at 0.1 second intervals and the use of cubic spline interpolation. Cubic spline interpolation is convenient because it not only provides values on a continuous curve through the sample points (position), but also values on a continuous first derivative (velocity) and a piecewise continuous second derivative (acceleration). The cubic spline interpolation algorithm is described in (Press et al. 1986).

Calculation of the position and attitude samples is performed in three stages. First, the trajectory generation commands are processed and coarse values of position and attitude are computed. At this point, the altitude profile consists of linear segments and the horizontal position profile consists of segments of great circles and small circles (turns) with a piecewise linear ground speed. A roll angle (bank angle) is computed during turns (that is, during a track change command) from simple equilibrium conditions. The pitch angle is computed using a simplified model relating lift force to angle of attack, velocity and altitude. For details of the roll and pitch calculations see (Von Mises 1959). The second stage smooths the trajectory using a zero phase (no delay) low pass filter. The filter is applied to each of the x, y and z earth-fixed position coordinates and to the roll and pitch angles. The smoothing, combined with a small sampling interval, results in a well behaved cubic spline interpolation. Note that turbulence and flexible body dynamics are not included in this simulation package (they are included in the

SARMCS simulation package). In the final stage, the heading is computed. A zero yaw angle is assumed so that heading is equal to the track angle. Since track angle is dependent on the already computed trajectory, heading is not an independent parameter, however, it is convenient to include heading (and consequently, its first and second derivatives) in the trajectory file. To obtain heading, the direction of the horizontal component of velocity, relative to north, is computed.

To simplify the calculations, horizontal position is computed using a spherical earth model and assuming zero altitude. Motion along a particular great circle or small circle is calculated by rotating the position vector about a fixed axis of rotation at an angular rate which depends on the ground speed. The resulting geocentric latitude and longitude are then converted to rectangular earth-fixed coordinates as if they were geodetic latitude and longitude computed on the ellipsoid. The effect of these approximations is a ground speed which differs slightly from the commanded value due of the different radius of curvature. It is most noticeable in a turn because the radius of curvature of the ellipsoid is dependent on the direction of travel which changes quickly during the turn.

Figure 4-2 shows the horizontal component of a flight profile generated using the trajectory command file in Table 4-1. The evaluation package was used to convert the trajectory earth-fixed coordinates to geodetic latitude and longitude and plot them in a map-like format.

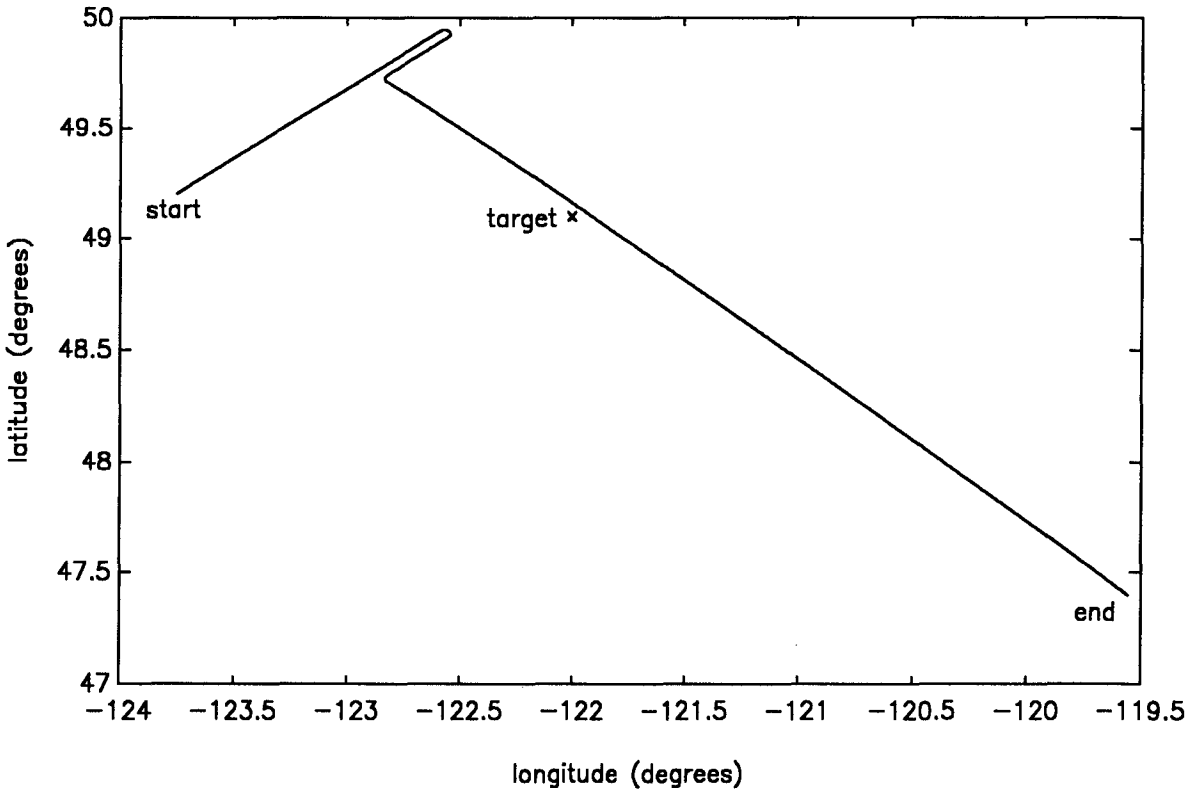


Figure 4-2 Flight Profile

4.2 SENSOR DATA SYNTHESIS

Sensor data synthesis is accomplished by a set of sensor simulation programs (a dedicated program for each sensor). These programs use the trajectory interpolation coefficient file to compute ideal sensor measurements which are then corrupted according to the sensor error model. Table 4-2 lists the sensor data synthesis programs and summarizes the synthesized data characteristics.

Table 4-2 Sensor Data Synthesis Programs

Sensor Program	Synthesized Data	Error Model	Sample Interval
INS	latitude longitude ground speed track angle heading pitch roll body pitch rate body roll rate body yaw rate platform heading altitude vertical speed north velocity east velocity	accelerometer errors: first-order Gauss-Markov (bias) gyro errors: first-order Gauss-Markov (bias) plus white noise (random drift)	62.5 ms
IMU	accelerometer ΔV's gyro $\Delta\theta$'s	accelerometer errors: first-order Gauss-Markov (bias) gyro errors: first-order Gauss-Markov (bias)	20 ms
GPS	latitude longitude altitude north velocity east velocity down velocity	position errors: first-order Gauss-Markov; independent when expressed in geographic coordinates velocity errors: white noise; independent when expressed in geographic coordinates	1 s
AZM	antenna azimuth	white noise	10 s
ALT	altitude	none	0.5 s

To synthesize data from a sensor, the appropriate sensor simulation program is run with the name of the desired trajectory interpolation coefficient file specified on the DOS command line. The simulation program requests start and end times for the simulation (within the limits of the

trajectory data) then generates a file containing the time stamped sensor data records. By default, the sensor data file takes the name of the trajectory coefficient data file with a file name extension dependent on the sensor being simulated. For example, the following DOS command line and prompt responses generate 1000 seconds of synthesized IMU data in a file named `TEST.IMU` from a trajectory interpolation coefficient file named `TEST.TJC`:

```
IMU TEST
Enter interpolation start time between 0 and 5020: 1000
Enter interpolation end time between 1000 and 5020: 2000
```

The sensor simulation programs are independent of each other, however, to generate a consistent set of synthesized sensor data files corresponding to a simulated flight, the same trajectory coefficient file must be used by each program and, typically, the same start and end times are specified for each sensor simulation.

4.2.1 INS Data Synthesis

The INS data synthesis program is called `INS`. It generates simulated INS data in a file with a default file name extension of `.INS`. The generated INS data includes latitude, longitude, ground speed, track angle, heading, pitch, roll, body pitch rate, body roll rate, body yaw rate, platform heading, altitude, vertical speed, north velocity and east velocity, all at an update rate of 16 Hz. To generate the INS data, ideal accelerometer and gyro measurements are computed from the trajectory data and corrupted with random errors then integrated in a strapdown inertial navigation algorithm.

The INS is assumed to be located at the centre of mass of the aircraft so that no lever arm correction to the trajectory data is required. Specific force and angular velocity are computed in body coordinates at a 256 Hz rate and integrated using a five point Newton-Cotes algorithm (Press et al. 1986) to obtain the accelerometer ΔV's and the gyro $\Delta\theta$'s at 64 Hz. These measurements are corrupted by errors before being used in the strapdown inertial navigation algorithm. The modeled errors include accelerometer and gyro biases and gyro random drift. The biases are modeled as first-order Gauss-Markov processes with long correlation times and the random drift is modeled using white noise. This simple error model is adequate for evaluating the effects of flight trajectory and aircraft/target geometry on the Kalman filter performance. Finally, the 64 Hz navigation quantities are subsampled to 16 Hz and written to the INS data file.

4.2.2 IMU Data Synthesis

The IMU data synthesis program is called `IMU`. It generates simulated accelerometer and gyro measurements in a file with a default file name extension of `.IMU`.

The IMU is mounted on the roll and pitch stabilized antenna ring gear. Synthesis of the IMU data must take into account the lever arms from the aircraft centre of mass to the antenna roll/pitch intersection and from the roll/pitch intersection to the IMU. The first lever arm is fixed in vehicle body coordinates and the second is fixed in stabilized antenna ring gear coordinates. Since the second lever arm is much shorter than the first, it is not significant for simulating the IMU dynamical environment. Nor is it significant relative to the large flight trajectory and aircraft/target geometry distances. Therefore, the IMU data synthesis calculations are simplified by assuming that the IMU is located at the antenna roll/pitch intersection.

Specific force and angular velocity are computed in IMU body coordinates at a 200 Hz rate and integrated to obtain accelerometer ΔV's and gyro $\Delta\theta$'s at 50 Hz. These measurements are corrupted according to the IMU error model. A first-order Gauss-Markov error model is used to generate accelerometer and gyro noise that is exponentially correlated in time. The measurements are then written to the IMU data file.

4.2.3 GPS Data Synthesis

The GPS data synthesis program, called `GPS`, generates simulated GPS position and velocity measurements in a file having a default file name extension of `.GPS`. The position and velocity data describe motion at the GPS antenna which is affixed to the top of the aircraft fuselage and, therefore, is displaced from the aircraft centre of mass by a fixed lever arm. Lever arm compensated position (latitude, longitude and altitude) and velocity (north, east and down) are computed from the trajectory data at one second intervals and corrupted according to the GPS error model before being written to the GPS data file.

The errors in GPS-indicated latitude, longitude and altitude are modelled as first-order Gauss-Markov processes and the errors in the GPS indicated north, east and down velocities are assumed to be uncorrelated from measurement to measurement. Although the three components of GPS position error (or velocity error) might properly be considered to be independent when expressed along the pseudo range directions, they are likely to be correlated with each other when expressed in geographic coordinates. The model used for the simulator, however, generates errors which are uncorrelated when expressed in geographic coordinates.

4.2.4 Azimuth Encoder Data Synthesis

Synthesized azimuth encoder measurements are generated by a program called `AZM` and written to a file having a default file name extension of `.AZM`. During calibration, the TAU keeps the antenna pointing at the target and the azimuth encoder measures the antenna azimuth angle. This angle is given by the target bearing minus the aircraft heading plus the antenna azimuth misalignment.

To synthesize the antenna azimuth encoder data, the position of the antenna roll/pitch intersection is computed at ten second intervals using the trajectory data and correcting for the lever arm from the aircraft centre of mass. Then, the target bearing is determined using the computed antenna roll/pitch intersection position and the known target position. After subtracting aircraft heading obtained from the trajectory data, and adding a small fixed error representing the antenna azimuth misalignment, the resulting azimuth value is corrupted using the TAU error model. Simulated TAU errors are uncorrelated from measurement to measurement. Finally, the synthesized measurements are written to the azimuth encoder data file.

Figure 4-3 is a plot of the synthesized azimuth encoder measurements corresponding to the flight profile and target location shown in Figure 4-2. A section of the graph is expanded in Figure 4-4 to show the measurement noise introduced by the TAU error model. Note that synthesized azimuth encoder measurements are generated for the entire flight profile. For the purpose of analyzing the Kalman filter, only those measurements corresponding to realistic TAU operating limits are used.

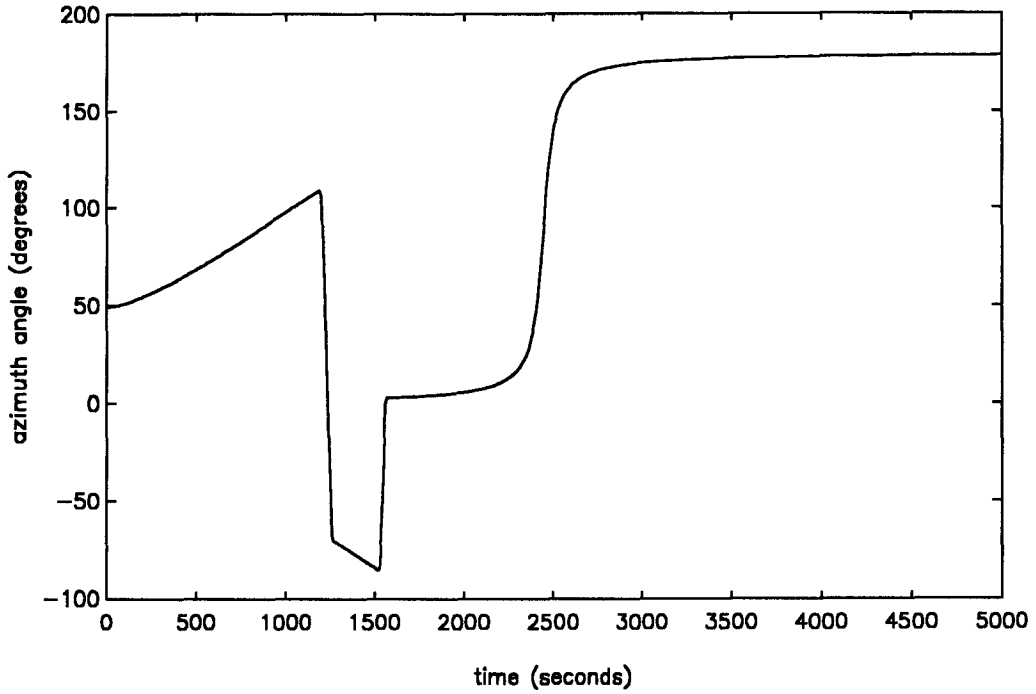


Figure 4-3 Azimuth Encoder Measurements

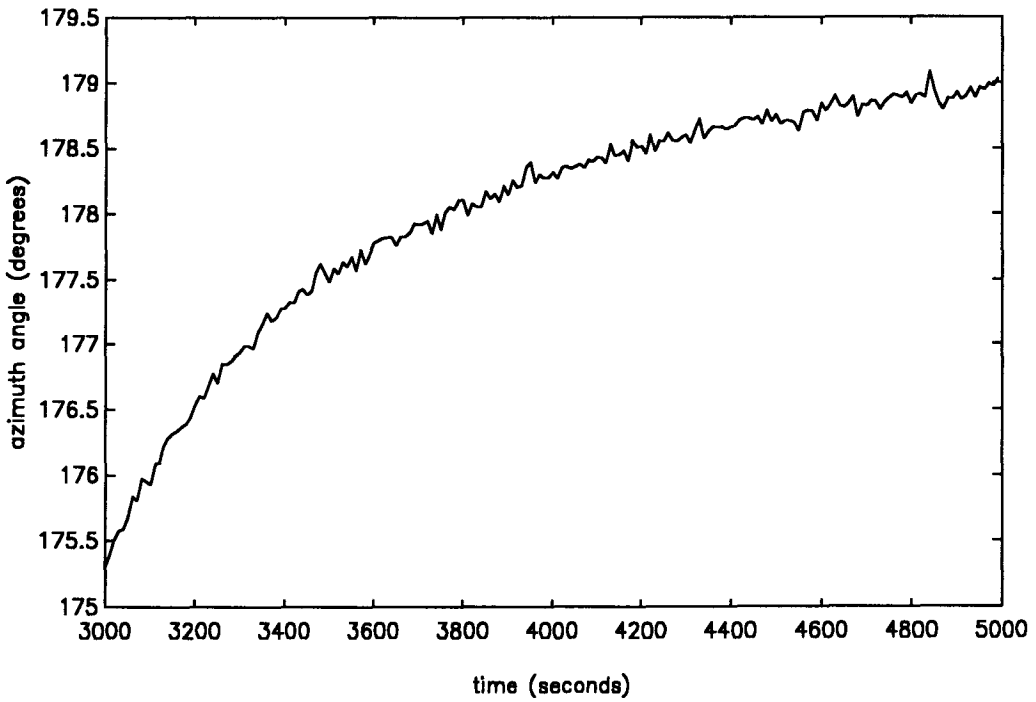


Figure 4-4 Azimuth Encoder Measurements Showing TAU Error

4.2.5 Baroaltitude Data Synthesis

A reference altitude is required to stabilize the vertical channel of the strapdown inertial navigator. Synthesized barometric altitude data is generated by a program called `ALT` and written to a file with a default file name extension of `.ALT`. Since altitude errors do not directly affect bearing calculations and are not modelled in the Kalman filter, no altitude errors are included in the simulator. Altitude is computed at 0.5 second intervals using the trajectory data and written to the altitude data file.

4.3 DATA PROCESSING

The data processing package implements the antenna boresight calibration Kalman filter in a program called `CALPRO`. The software is divided into a main program, a navigation component, a generic Kalman filter component and a system model component.

The main program handles housekeeping chores associated with the reading of files containing sensor data, the writing of processing results to files for later analysis and the execution of the inertial navigation and Kalman filtering procedures.

The navigation component includes an implementation of the strapdown inertial navigation algorithm in a procedure called `navgtr` and a collection of commonly used, navigation related procedures in a library named `NAVLIB`.

The generic Kalman filter component implements the basic Kalman filter equations for state extrapolation and measurement updates. Also included is a procedure for Taylor series approximation of the discrete time state transition matrix and process noise covariance matrix

given the continuous time system dynamics matrix and process noise spectral density matrix. The procedures, called `extrap`, `updat` and `PHIQd`, respectively, are independent of any particular system model. The basic Kalman filter equations are directly implemented except that only the upper triangular portion of the error covariance matrix is computed and used (the symmetry property is used to determine the lower triangular portion). Since the simulated flights are of relatively short duration and the Kalman filter calculations are performed using double precision arithmetic, no special precautions are necessary in the simulator software to ensure the positive definiteness of the error covariance matrix.

The system model is defined in a module with the generic name `MODEL`. The name is easily changed to something more descriptive or to accommodate multiple, uniquely named models. `MODEL` contains a framework for all of the data storage allocations and data processing procedures which are required by the generic Kalman filter component and which are specific to a particular system model. Working storage is allocated for: the state vector; the measurement vector; the continuous time system dynamics matrix; the continuous time process noise spectral density matrix; the discrete time state transition matrix; the discrete time process noise covariance matrix; the measurement matrix; the measurement noise covariance matrix; the error covariance matrix; the Kalman gain matrix; and temporary vectors and matrices for intermediate results. Procedures are provided to: return pointers to the allocated working storage vectors and matrices; initialize the state vector and the error covariance matrix; construct the continuous time system dynamics matrix and process noise spectral density matrix; and construct the measurement vector, measurement matrix and measurement noise covariance matrix. The generic procedure names are `model_wrk`, `model_x0P0`, `model_FQ` and `model_mes`, respectively. To create a specific system model, the number of states and the

maximum number of measurements must be specified and the bodies of three of the procedures (`model_x0P0`, `model_FQ` and `model_mes`) must be provided. The specific system model and the generic Kalman filter combine to form a basic Kalman filter implementation as shown in Table 4-3.

Table 4-3 Basic Kalman Filter Using System Model and Generic Kalman Filter Procedures

Perform once at initialization:

`model_wrk`	Enable access to the working storage.
`model_x0P0`	Initialize the state vector and the error covariance matrix.

Perform at each filter iteration (F and Q are assumed to be time varying):

`model_FQ`	Construct the continuous time system dynamics matrix and process noise spectral density matrix.
`PHIQd`	Compute the discrete time state transition matrix and process noise covariance matrix.
`extrap`	Extrapolate the state vector and the error covariance matrix based on the process model.
`model_mes`	Construct the measurement vector, the measurement matrix and the measurement noise covariance matrix.
`updat`	Update the state vector and the error covariance matrix based on the measurement model and the measurement.

4.4 DATA ANALYSIS

The evaluation package relies on MATLAB for data analysis and plotting. Within MATLAB, entire data vectors are viewed as individual objects (the fundamental data object in MATLAB is the matrix which includes vectors and scalars as special cases). MATLAB permits mathematical operations and graphing operations to be performed on the data vectors either interactively or by executing files of MATLAB statements called M-files.

A data conversion program called `MATCNV` converts data files generated by the synthesis package and the processing package to MATLAB format. `MATCNV` reads binary files of fixed length records containing only double precision elements (all of the files generated by the simulation software package conform to this format) and generates a set of MATLAB files (one file for each record element) containing a single data vector each. M-files have been written to automate the process of converting the data and reading it into MATLAB. `MATCNV` also provides the capability to subsample the input data file. For example, a data file from a simulated two hour flight containing sample records at 20 millisecond intervals is too large to be plotted conveniently. `MATCNV` can be used to subsample by a factor of, say, 50 so that only the samples at one second intervals are read into MATLAB for plotting.

The evaluation package also includes a program called `NAVO` which converts the reference trajectory file position and velocity from earth-fixed coordinates into the more standard navigation quantities of geodetic latitude, longitude, altitude, ground speed, track angle and vertical speed.

Figure 4-5 demonstrates some of the capabilities of the evaluation package. It is a comparison between the IMU ground speed and the reference ground speed. The trajectory was generated using the command file in Table 4-1. IMU ground speed was computed by the inertial navigation component of CALPRO using synthesized IMU accelerometer and gyro data. The reference ground speed is the error free aircraft centre of mass (or INS) ground speed obtained from the reference trajectory file using the NAVO conversion program to convert from earth-fixed coordinates. MATCNV and the appropriate M-files were used to load the data into MATLAB where it was differenced and then plotted.

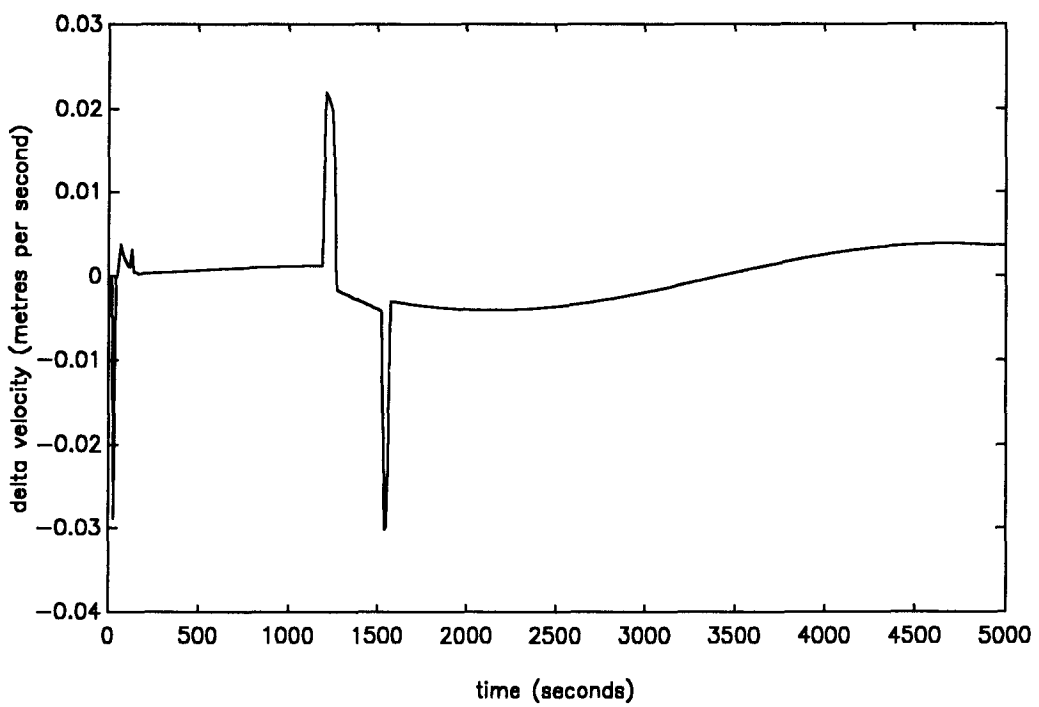


Figure 4-5 Difference Between Computed IMU Ground Speed and Reference Ground Speed

The spikes in the graph are the result of the INS-to-IMU lever arm in combination with pitch changes during takeoff and roll changes during turns. Also seen is the 84-minute Schuler oscillation (observable during the period of straight and level flight starting from about 2000 seconds) which is characteristic of inertial navigation errors. For this example, the IMU error model was disabled, thus, the velocity error is due primarily to numerical integration errors in the inertial navigation algorithm.

SECTION FIVE

FLIGHT TRAJECTORY AND AIRCRAFT/TARGET GEOMETRY

It is necessary to determine a flight profile and aircraft/target geometry that will allow the antenna azimuth misalignment to be rapidly and accurately estimated. This determination is performed by using the simulation software package to generate various profiles and geometries and to analyze the results. By comparing error covariances, an appropriate choice of flight profile and aircraft/target geometry can be made.

5.1 PRELIMINARY CONSIDERATIONS

The selection of possible flight profiles and aircraft/target geometries can be narrowed by considering the theoretical requirements for estimating the antenna azimuth misalignment and the operational limitations of the aircraft and other equipment.

Nominally straight and level flight at constant ground speed is operationally convenient. It also minimizes the effects of instrument scale factor and misalignment errors, and minimizes errors in constructing the time varying state transition matrix. Conversely, manoeuvres are required to generate the horizontal components of acceleration necessary for separation of accelerometer biases from misalignments and, more important, for prompt estimation of azimuth misalignment. When manoeuvres are performed while making TAU antenna azimuth measurements, however, the measurements are susceptible to timing related errors due to the high antenna azimuth rates. Given these considerations, a reasonable flight profile consists of manoeuvres followed by a period of straight and level flight during which the antenna azimuth misalignment is estimated.

A simple manoeuvre that provides the necessary horizontal accelerations is a 180 degree turn. Another simple manoeuvre, that also preserves the original direction of travel, is an S-turn (two consecutive 180 degree turns in opposite directions). During the straight portion of the flight profile, antenna azimuth rates can be kept near zero by flying directly toward the target. Also, potential problems associated with the antenna elevation angle and elevation rate can be minimized if the range between the aircraft and the target remains much greater than the aircraft height above the target.

A nominal altitude of 3300 feet (1000 metres) is a reasonable choice for the simulations (the process and measurement models used for the Kalman filter are virtually independent of aircraft altitude). A convenient cruising speed for the aircraft is 210 nautical miles per hour (108 metres per second). Turn rates of 180 degrees/minute are appropriate for the manoeuvres. While the TAU azimuth measurements are being made, a maximum target range in the neighbourhood of 100 kilometres and a minimum target range of about 10 kilometres represent reasonable limits.

5.2 SIMULATION METHODS

The purpose of these simulations is to determine a suitable flight profile and aircraft/target geometry by analyzing the error covariance. The simulations are performed using the computer simulation software package (listings of the software modules comprising the data processing package are included in Appendix C). To calculate the error covariance $\mathbf{P}_k$, it is necessary to know $\mathbf{\Phi}_k$, $\mathbf{Q}_k$, $\mathbf{H}_k$, $\mathbf{R}_k$ and $\mathbf{P}_0^-$. Although it is not strictly necessary to process measurements and generate the state estimate, these operations are performed to provide a reasonability check as well as additional insight into the filter operation.

Formulation of the $\mathbf{F}$, $\mathbf{Q}$ and $\mathbf{H}_k$ matrices is described in Section 3. For these simulations, a Kalman filter processing cycle is performed for each GPS sensor data record (that is, at fixed one second time increments), except for antenna azimuth matching measurements (which are only available at ten second intervals). Truncated Taylor series approximations are used to obtain $\mathbf{\Phi}_k$ (equation (3-15) with $N_\Phi=3$) and $\mathbf{Q}_k$ (equation (3-16) with $N_Q=6$). The simulations are performed using values for the diagonal matrices $\mathbf{Q}$, $\mathbf{P}_0^-$ and $\mathbf{R}_k$ as shown in Tables 5-1, and 5-2. In Table 5-1, the process noise spectral densities corresponding to exponentially correlated processes are written in the form $2\times\sigma^2\div\tau$ so that the mean square values σ^2 and the correlation times τ (in seconds) are easily seen. The quantities a, V_{gnd} and $|\omega_z|$ appearing in some of the spectral densities are defined as follows:

a = magnitude of the aircraft acceleration vector in m/s^2.

V_{gnd} = aircraft ground speed in m/s (hard-limited to a minimum value of 70 m/s).

$|\omega_z|$ = absolute value of the aircraft heading rate in rad/s.

The values for the rms error in the uncalibrated antenna azimuth (see the initial error covariance in Table 5-1) and the rms error in the antenna azimuth matching measurements (see the noise covariance in Table 5-2) have been somewhat arbitrarily chosen for these simulations. The uncalibrated antenna azimuth rms error is assumed to be of the order of one degree. TAU azimuth measurements have an rms error of 0.05 degrees based on experiments with a stationary SARMCS radar and target. The in-flight antenna azimuth matching measurements are assumed to include an additional error component of about 0.1 degrees due to time offsets between the compared azimuth values (more on this later).

Table 5-1 Process Noise Spectral Density and Initial Error Covariance

State	Process Noise Spectral Density		Initial Error Covariance	
δR_{Mx}	0	$(m/s)^2/Hz$	1×10^6	$(m)^2$
δR_{My}	0	$(m/s)^2/Hz$	1×10^6	$(m)^2$
δV_{Mx}	$(6\times10^{-3})/V_{gnd} + (3.2\times10^{-9})aV_{gnd}$	$(m/s^2)^2/Hz$	4	$(m/s)^2$
δV_{My}	$(6\times10^{-3})/V_{gnd} + (3.2\times10^{-9})aV_{gnd}$	$(m/s^2)^2/Hz$	4	$(m/s)^2$
ϕ_{Mx}	7.6×10^{-7}	$(mrad/s)^2/Hz$	0.09	$(mrad)^2$
ϕ_{My}	7.6×10^{-7}	$(mrad/s)^2/Hz$	0.09	$(mrad)^2$
ϕ_{Mz}	7.6×10^{-7}	$(mrad/s)^2/Hz$	25	$(mrad)^2$
A_{Mx}	$2\times(2.4\times10^{-7})\div360{,}000$	$(m/s^3)^2/Hz$	2.4×10^{-7}	$(m/s^2)^2$
A_{My}	$2\times(2.4\times10^{-7})\div360{,}000$	$(m/s^3)^2/Hz$	2.4×10^{-7}	$(m/s^2)^2$
G_{Mx}	$2\times(2.4\times10^{-9})\div360{,}000$	$(mrad/s^2)^2/Hz$	2.4×10^{-9}	$(mrad/s)^2$
G_{My}	$2\times(2.4\times10^{-9})\div360{,}000$	$(mrad/s^2)^2/Hz$	2.4×10^{-9}	$(mrad/s)^2$
G_{Mz}	$2\times(2.4\times10^{-9})\div360{,}000$	$(mrad/s^2)^2/Hz$	2.4×10^{-9}	$(mrad/s)^2$
δR_{Gx}	$2\times400\div10{,}800$	$(m/s)^2/Hz$	400	$(m)^2$
δR_{Gy}	$2\times400\div10{,}800$	$(m/s)^2/Hz$	400	$(m)^2$
δR_{Sx}	0	$(m/s)^2/Hz$	100	$(m)^2$
δR_{Sy}	0	$(m/s)^2/Hz$	100	$(m)^2$
δV_{Sx}	$(5.42\times10^{-8})aV_{gnd}$	$(m/s^2)^2/Hz$	4×10^{-4}	$(m/s)^2$
δV_{Sy}	$(5.42\times10^{-8})aV_{gnd}$	$(m/s^2)^2/Hz$	4×10^{-4}	$(m/s)^2$
ϕ_{Sx}	$(1.2\times10^{-8})aV_{gnd} + (1.32\times10^{-1})\lvert\omega_z\rvert + 7.6\times10^{-7}$	$(mrad/s)^2/Hz$	400	$(mrad)^2$
ϕ_{Sy}	$(1.2\times10^{-8})aV_{gnd} + (1.32\times10^{-1})\lvert\omega_z\rvert + 7.6\times10^{-7}$	$(mrad/s)^2/Hz$	400	$(mrad)^2$
ϕ_{Sz}	$(1.2\times10^{-8})aV_{gnd} + (7.1\times10^{-2})\lvert\omega_z\rvert + 7.6\times10^{-7}$	$(mrad/s)^2/Hz$	225	$(mrad)^2$
G_{Sx}	$2\times(2.1\times10^{-6})\div36{,}000$	$(mrad/s^2)^2/Hz$	2.1×10^{-6}	$(mrad/s)^2$
G_{Sy}	$2\times(2.1\times10^{-6})\div36{,}000$	$(mrad/s^2)^2/Hz$	2.1×10^{-6}	$(mrad/s)^2$
G_{Sz}	$2\times(2.1\times10^{-6})\div36{,}000$	$(mrad/s^2)^2/Hz$	2.1×10^{-6}	$(mrad/s)^2$
ε	0	$(mrad/s)^2/Hz$	400	$(mrad)^2$

Table 5-2 Measurement Noise Covariance

Measurement	Measurement Noise Covariance	
INS/GPS x-position matching	25	$(m)^2$
INS/GPS y-position matching	25	$(m)^2$
IMU/INS x-position matching	4	$(m)^2$
IMU/INS y-position matching	4	$(m)^2$
Antenna azimuth matching	4	$(mrad)^2$

Each sensor provides measurements at a particular location on the aircraft. For these simulations, the following location and lever arm assumptions are made:

- The INS is located at the aircraft centre of mass (the aircraft body frame origin).
- Barometric altitude measurements are taken at the aircraft centre of mass.
- The GPS antenna is attached to the fuselage on the top of the aircraft and the INS-to-GPS lever arm is $[-3.0, -0.5, -2.0]^T$ metres in aircraft body coordinates.
- The IMU is located at the intersection of the antenna roll and pitch axes and the INS-to-IMU lever arm is $[6.0, -0.5, -0.1]^T$ metres in aircraft body coordinates.

In order to properly test the effectiveness of the flight profile manoeuvres, the effects of any prior manoeuvres (for example, takeoff) must be excluded. This is achieved in the simulations by initiating the data processing during a straight and level segment of the flight profile included prior to the test manoeuvres. At least 1000 seconds of data processing is performed before initiating the test manoeuvres to allow the Kalman filter to settle.

The following points must be kept in mind when analyzing the simulation results:

- Turbulence and flexible body dynamics are not simulated.
- Vertical deflections and gravity anomalies are not simulated.
- Accelerometer and gyro scale factor errors, misalignments and g-sensitive errors are not simulated.
- Barometric altitude errors are not simulated.
- Instrument quantization errors are not simulated.
- Timing errors in matching measurements are not simulated.

A consequence of the limited error models used in the data synthesis package is nearly optimal Kalman filter performance when processing this data. The only significant errors not fully modelled by the filter are the horizontal IMU accelerometer biases (vertical channel errors are minimized by baroaltitude damping). But SARMCS experience has shown that modelling these states in the filter does not significantly change steady-state performance. These simulations, therefore, provide a best performance baseline for subsequent evaluations.

5.3 SIMULATION RESULTS

This subsection presents the results of the simulation runs. In analyzing the results, emphasis is placed on the ultimate objective of estimating the antenna boresight azimuth misalignment to the required accuracy of 5 arc minutes rms or better in a minimum amount of time. Subsection 5.3.5 summarizes the findings of many simulations. Detailed results of three representative simulations are contained in subsections 5.3.2, 5.3.3 and 5.3.4 using flight profiles described in subsection 5.3.1. These simulation runs are referred to as 1a, 2a and 3a, where the number

identifies the flight profile and the letter is used to distinguish between different target locations or calibration times with the same flight profile. For convenience, the large number of plots associated with these simulations have been placed in Appendix A. Only selected plots are included in this section. All of the plots were produced using the computer simulation evaluation package described in Section 4.

5.3.1 Flight Profiles

Three flight profiles are referenced in the following subsections. These flight profiles were generated using the computer simulation software package trajectory generator described in Section 4. The command files used to generate the flight profiles are listed in Tables 5-3 through 5-5. In the command files, times and durations are specified in seconds, horizontal position values are specified as latitude/longitude pairs in units of degrees, minutes and seconds, altitude is specified in feet, track angle is in units of degrees and ground speed is specified in nautical miles per hour.

Following the takeoff and initial climb, each flight profile has a nominal altitude of 3300 feet and a nominal ground speed of 210 nautical miles per hour. The manoeuvres are performed at a turn rate of 180 degrees per minute. These values correspond to those specified in subsection 5.1. In addition, each profile is generated at a mid latitude of about 45 degrees and is arbitrarily oriented in a nominally east/west direction. Plots of the horizontal plane of each flight profile are shown in Figures 5-1 through 5-3. The "+" marks along the profile indicate 100 second intervals from time 0 seconds at position 45 degrees north, 75 degrees west to time 5000 seconds at the end of the profile.

Table 5-3 Flight Profile 1 Command File

```
% CALPRO.CMD
% Command file to generate flight profile #1.

% Initial stationary position at beginning of runway.
initialize position to (45:00:00,-75:00:00);
initialize altitude to 0;
initialize groundspeed to 0;
initialize track to 90;  % And initialize wander angle to -90.

% Takeoff and initial climb.
at 10 change groundspeed to 120 in 20;
at 30 change altitude to 1000 in 100;
at 60 change groundspeed to 210 in 100;
at 130 change altitude to 3300 in 660;

% Manoeuvres.
at 2200 change track by -180 in 60;
at 2360 change track by 180 in 60;
at 3800 change track by 180 in 60;

% End.
at 5020 end;
```

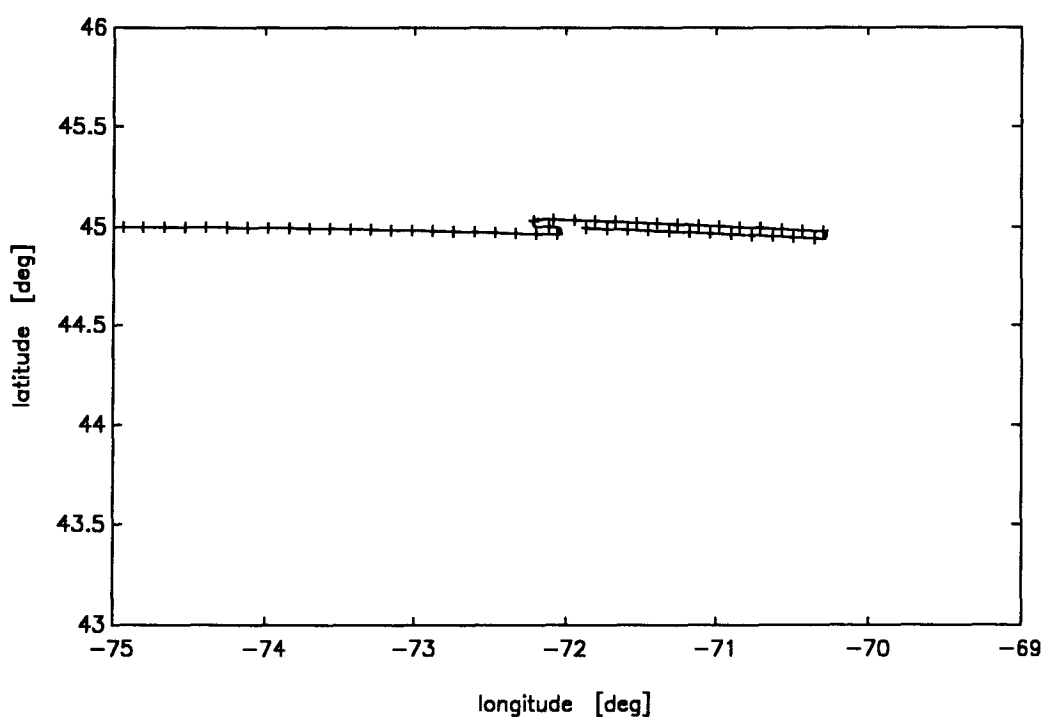


Figure 5-1 Flight Profile 1

Table 5-4 Flight Profile 2 Command File

```
% CALPRO.CMD
% Command file to generate flight profile #2.

% Initial stationary position at beginning of runway.
initialize position to (45:00:00,-75:00:00);
initialize altitude to 0;
initialize groundspeed to 0;
initialize track to 90;  % And initialize wander angle to -90.

% Takeoff and initial climb.
at 10 change groundspeed to 120 in 20;
at 30 change altitude to 1000 in 100;
at 60 change groundspeed to 210 in 100;
at 130 change altitude to 3300 in 660;

% Manoeuvres.
at 2200 change track by -180 in 60;
at 3260 change track by -180 in 60;

% End.
at 5020 end;
```

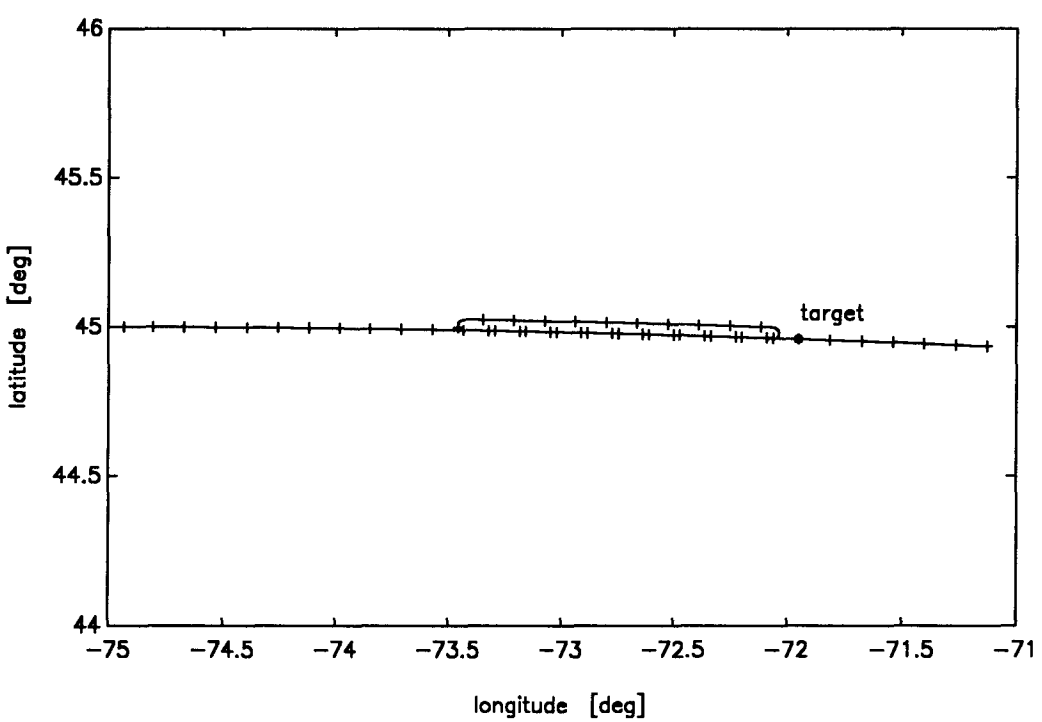


Figure 5-2 Flight Profile 2

Table 5-5 Flight Profile 3 Command File

```
% CALPRO.CMD
% Command file to generate flight profile #3.

% Initial stationary position at beginning of runway.
initialize position to (45:00:00,-75:00:00);
initialize altitude to 0;
initialize groundspeed to 0;
initialize track to 90;  % And initialize wander angle to -90.

% Takeoff and initial climb.
at 10 change groundspeed to 120 in 20;
at 30 change altitude to 1000 in 100;
at 60 change groundspeed to 210 in 100;
at 130 change altitude to 3300 in 660;

% Manoeuvres.
at 2200 change track by -180 in 60;
at 2360 change track by 180 in 60;
at 3420 change track by 180 in 60;
at 3580 change track by -180 in 60;

% End.
at 5020 end;
```

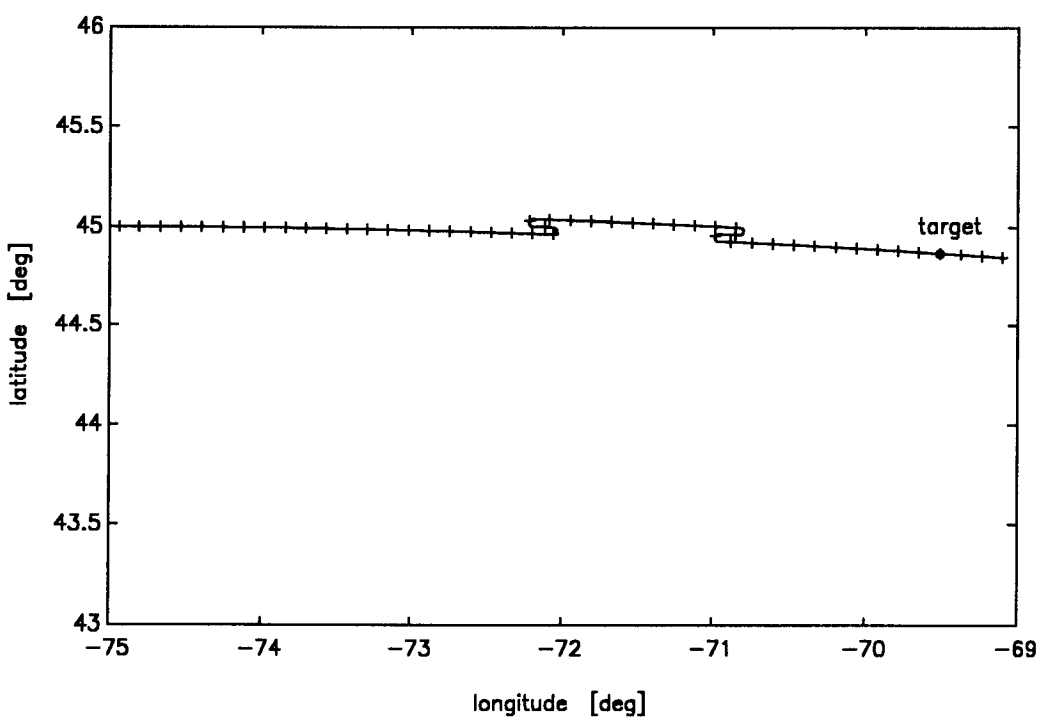


Figure 5-3 Flight Profile 3

5.3.2 Simulation Run 1a

Flight Profile:	#1 (see subsection 5.3.1)
Target Location:	n/a
Data Processing Start Time:	1000 s
Data Processing End Time:	5000 s
Calibration Start Time:	n/a
Calibration End Time:	n/a
Special Conditions:	• Data processing initiated during straight and level flight prior to manoeuvres. • Manoeuvres consist of an S-turn at 2200 seconds and a 180 degree turn at 3800 seconds. • No antenna boresight calibration.

Plots: (see Appendix A)	1a-1	True and estimated x INS position error
	1a-2	True and estimated y INS position error
	1a-3	True and estimated x INS velocity error
	1a-4	True and estimated y INS velocity error
	1a-5	True and estimated x INS platform misalignment
	1a-6	True and estimated y INS platform misalignment
	1a-7	True and estimated z INS platform misalignment
	1a-8	True and estimated x INS accelerometer bias
	1a-9	True and estimated y INS accelerometer bias
	1a-10	True and estimated x INS gyro bias
	1a-11	True and estimated y INS gyro bias
	1a-12	True and estimated z INS gyro bias
	1a-13	Error in x INS position error estimate
	1a-14	Error in y INS position error estimate
	1a-15	Error in x INS velocity error estimate
	1a-16	Error in y INS velocity error estimate
	1a-17	Error in x INS platform misalignment estimate
	1a-18	Error in y INS platform misalignment estimate
	1a-19	Error in z INS platform misalignment estimate
	1a-20	Error in x INS accelerometer bias estimate
	1a-21	Error in y INS accelerometer bias estimate

1a-22 Error in x INS gyro bias estimate
1a-23 Error in y INS gyro bias estimate
1a-24 Error in z INS gyro bias estimate
1a-25 Error in x GPS position error estimate
1a-26 Error in y GPS position error estimate
1a-27 Error in corrected x IMU position
1a-28 Error in corrected y IMU position
1a-29 Error in corrected x IMU velocity
1a-30 Error in corrected y IMU velocity
1a-31 Error in corrected x IMU platform alignment
1a-32 Error in corrected y IMU platform alignment
1a-33 Error in corrected z IMU platform alignment
1a-34 Error in x IMU gyro bias estimate
1a-35 Error in y IMU gyro bias estimate
1a-36 Error in z IMU gyro bias estimate
1a-37 x INS/GPS position matching measurement residual
1a-38 y INS/GPS position matching measurement residual
1a-39 x IMU/INS position matching measurement residual
1a-40 y IMU/INS position matching measurement residual

Simulation run 1a is performed with antenna boresight calibration disabled. The simulation serves three purposes. First, it demonstrates the operation of the current SARMCS Kalman filter design as implemented in the simulator data processing package. Second, with antenna boresight calibration disabled, it forms a basis for observing the effects of enabling antenna calibration. Third, it confirms the effectiveness of the S-turn and the 180 degree turn for azimuth misalignment estimation and it provides additional insight into manoeuvre requirements. To serve these purposes, an extensive set of plots (1a-1 through 1a-40) is included in Appendix A.

Plots 1a-1 through 1a-12 show true and Kalman filter estimated INS errors on the same graph. In each graph, the total INS error and the relative magnitude of the corresponding estimation error are readily observed. Plots 1a-13 to 1a-24 graph the difference between the estimated and

true INS errors and Plots 1a-25 and 1a-26 graph the difference between the estimated and true GPS errors. In these plots, the Kalman filter estimated standard deviation bounds are indicated by dashed lines. The corrected IMU navigation quantities are shown in Plots 1a-27 through 1a-33 and the estimated IMU gyro biases are shown in Plots 1a-34 to 1a-36. These plots also show the Kalman filter estimated standard deviation bounds as dashed lines. Comparison of the Kalman filter estimation errors with the predicted error standard deviations, provides confidence that the filter is adequately estimating INS errors and transferring position and alignment to the IMU navigator. In all of the plots, the estimation error is fairly consistent with the standard deviation bounds indicating that the filter is performing well. Plots 1a-37 through 1a-40 show the measurement residuals along with the Kalman filter predicted rms values. The residuals are substantially less than the predicted levels due to the absence of timing related measurement errors in the simulator.

The importance of accurate IMU position and azimuth alignment for estimation of the antenna boresight misalignment is indicated by Equation (3-9). Figure 5-4 shows that the Kalman filter predicted IMU position error bounds are essentially dictated by the rms error in the GPS-indicated position. Observability and estimation of the of azimuth misalignment, on the other hand, requires manoeuvres as described in subsection 5.1. Flight profile number one includes an S-turn manoeuvre at 2200 seconds and a 180 degree turn manoeuvre at 3800 seconds. Figure 5-5 shows the error in the IMU platform misalignment about the z axis. Examination of the Kalman filter predicted rms error verifies that both manoeuvres are effective for the intended purpose of estimating IMU azimuth misalignment. In Figure 5-6, the filter predicted rms error in the z IMU gyro bias demonstrates that a single manoeuvre does not ensure adequate estimation of this gyro bias but that two separated manoeuvres do provide for adequate

estimation of the z IMU gyro bias. The importance of controlling z IMU gyro bias is indicated in Figure 5-5 by the rapid deterioration of the z IMU platform misalignment estimate following the first manoeuvre.

In this and subsequent plots, solid lines represent the actual quantity (position error, for example) and dashed lines lines represent the Kalman filter predicted rms error bounds.

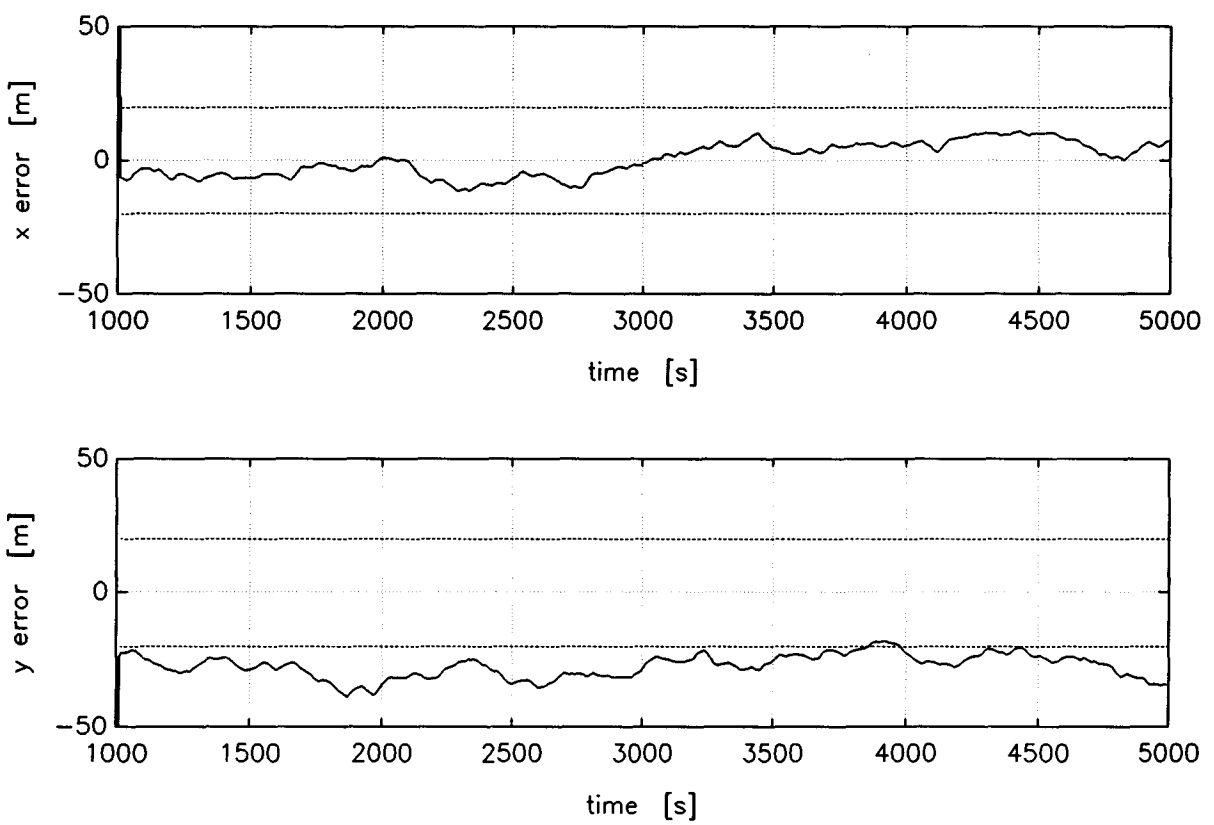


Figure 5-4 Error in Corrected IMU Position for Simulation Run 1a

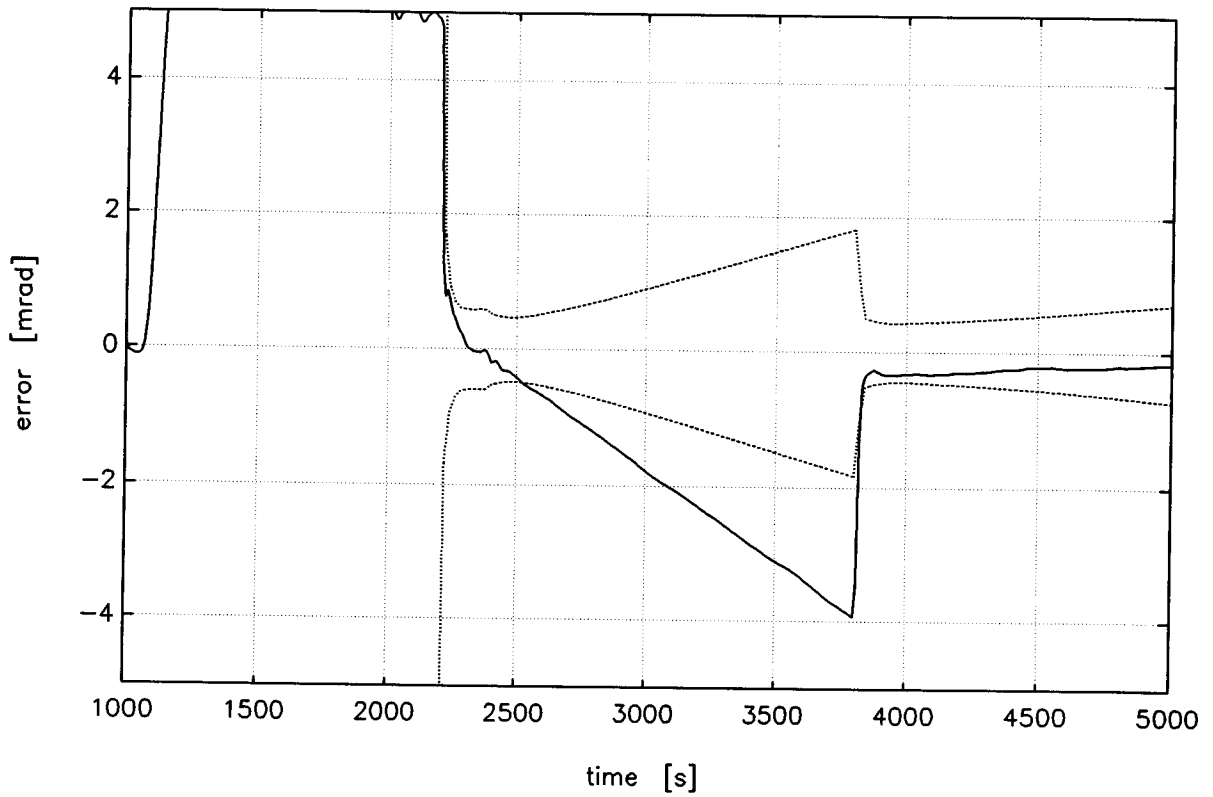


Figure 5-5 Error in Corrected z IMU Platform Alignment for Simulation Run 1a

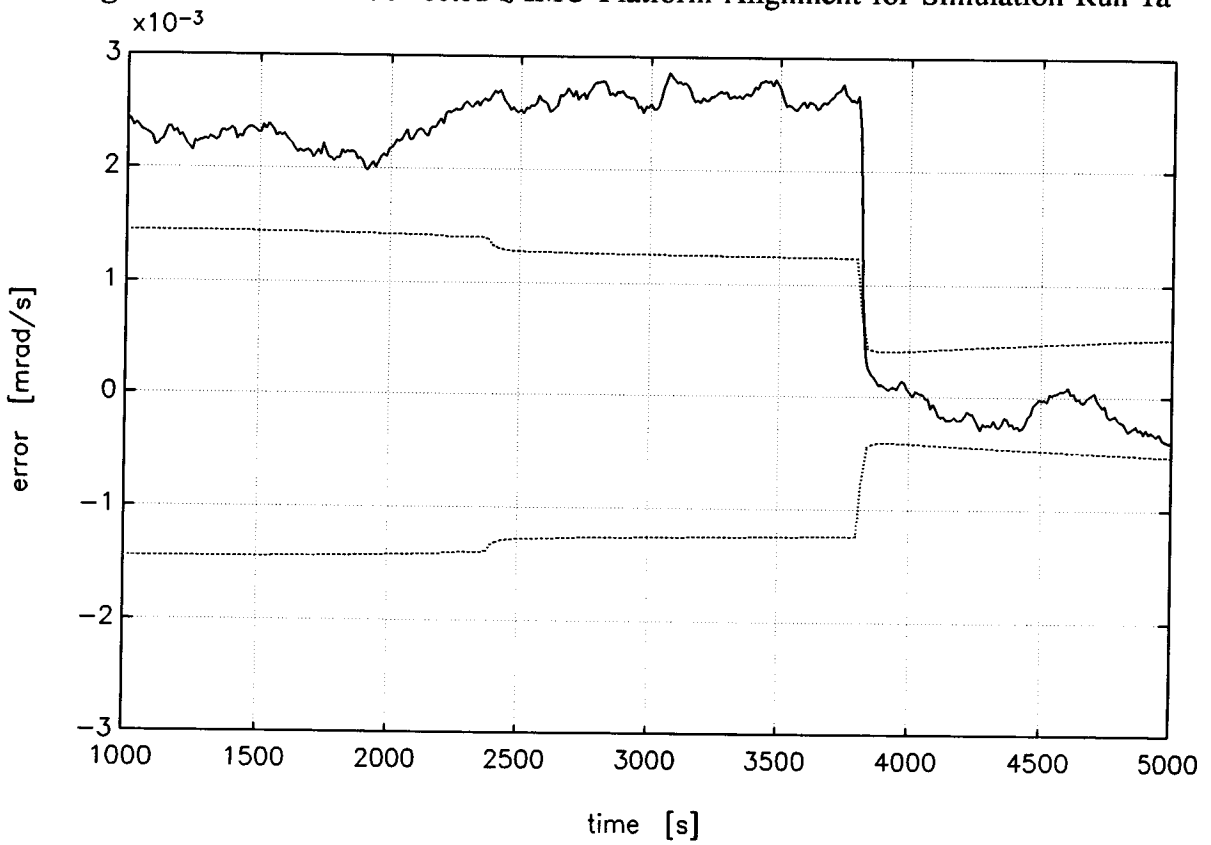


Figure 5-6 Error in z IMU Gyro Bias Estimate for Simulation Run 1a

5.3.3 Simulation Run 2a

Flight Profile:	#2 (see subsection 5.3.1)
Target Location:	44° 57′ 34″ N 71° 56′ 57″ W
Data Processing Start Time:	1000 s
Data Processing End Time:	5000 s
Calibration Start Time:	3400 s
Calibration End Time:	4300 s
Special Conditions:	• Data processing initiated during straight and level flight prior to manoeuvres. • Manoeuvres consist of 180 degree turns at 2200 seconds and 3260 seconds. • Calibration performed during straight and level flight following manoeuvres. • Antenna azimuth misalignment is 20 milliradians.

Plots: (see Appendix A)	2a-1	Error in x INS position error estimate
	2a-2	Error in y INS position error estimate
	2a-3	Error in x INS velocity error estimate
	2a-4	Error in y INS velocity error estimate
	2a-5	Error in x INS platform misalignment estimate
	2a-6	Error in y INS platform misalignment estimate
	2a-7	Error in z INS platform misalignment estimate
	2a-8	Error in x INS accelerometer bias estimate
	2a-9	Error in y INS accelerometer bias estimate
	2a-10	Error in x INS gyro bias estimate
	2a-11	Error in y INS gyro bias estimate
	2a-12	Error in z INS gyro bias estimate
	2a-13	Error in x GPS position error estimate
	2a-14	Error in y GPS position error estimate
	2a-15	Error in corrected x IMU position
	2a-16	Error in corrected y IMU position
	2a-17	Error in corrected x IMU velocity

2a-18 Error in corrected y IMU velocity
2a-19 Error in corrected x IMU platform alignment
2a-20 Error in corrected y IMU platform alignment
2a-21 Error in corrected z IMU platform alignment
2a-22 Error in x IMU gyro bias estimate
2a-23 Error in y IMU gyro bias estimate
2a-24 Error in z IMU gyro bias estimate
2a-25 Error in antenna azimuth misalignment estimate
2a-26 Antenna azimuth matching measurement residual
2a-27 Antenna azimuth matching Kalman gain

This simulation run represents a potential antenna boresight calibration flight profile and aircraft/target geometry. Two 180 degree turns separated by 1000 seconds of straight and level flight constitute the manoeuvre segment of the profile. Calibration is initiated following the second manoeuvre at a distance from the target of just over 100 kilometres. During the calibration segment, the aircraft flies directly toward the target and, after 900 seconds, calibration is terminated at just over 10 kilometres from the target.

The error in the estimated antenna azimuth misalignment is shown in Figure 5-7. The Kalman filter predicted error is seen to fall to a nearly constant level near 0.6 milliradians rms (2 arc minutes rms) within about 500 seconds. Figure 5-8 shows the antenna azimuth matching measurement residuals along with the Kalman filter predicted rms value. As previously seen for the position matching measurement residuals in simulation run 1a, the azimuth matching measurement residuals are somewhat less than predicted due to the absence of timing errors in the simulator. The Kalman gain connecting the antenna azimuth matching measurement residuals to the antenna azimuth estimate is shown in Figure 5-9. This plot indicates that the Kalman filter rapidly estimates the antenna azimuth misalignment and begins ignoring the

azimuth matching measurements, relying instead on the assumed noise free, random constant process model.

It is of interest to compare Plots 2a-1 through 2a-24 with the corresponding plots from simulation run 1a (that is, Plots 1a-13 to 1a-36). The one notable effect of the inclusion of antenna azimuth matching measurements is a reduction in the rms position error as the aircraft nears the target. This can be seen in Figure 5-10. Here, the Kalman filter is making use of the assumed perfect knowledge of the target location. The reduced position error is of little benefit to the antenna azimuth calibration, however, because it is not significant until after the filter has essentially completed estimation of the azimuth misalignment.

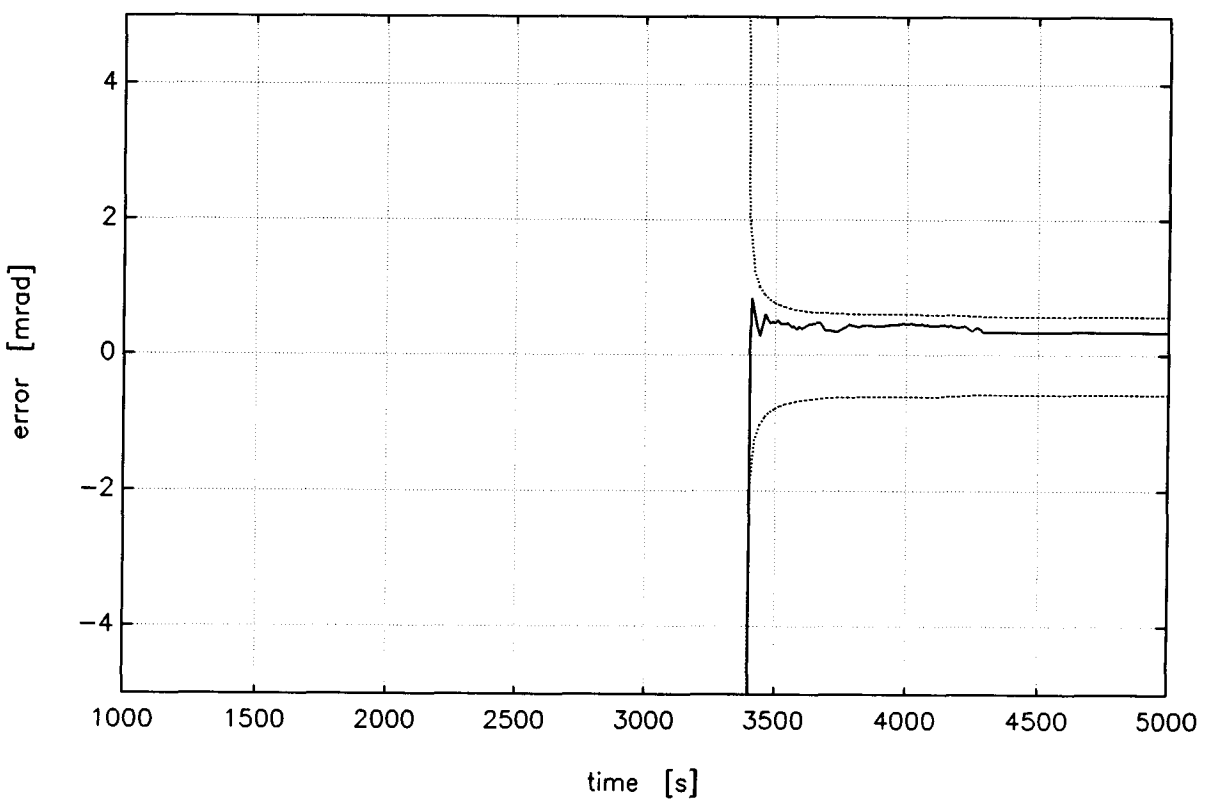


Figure 5-7 Error in Antenna Azimuth Misalignment Estimate for Simulation Run 2a

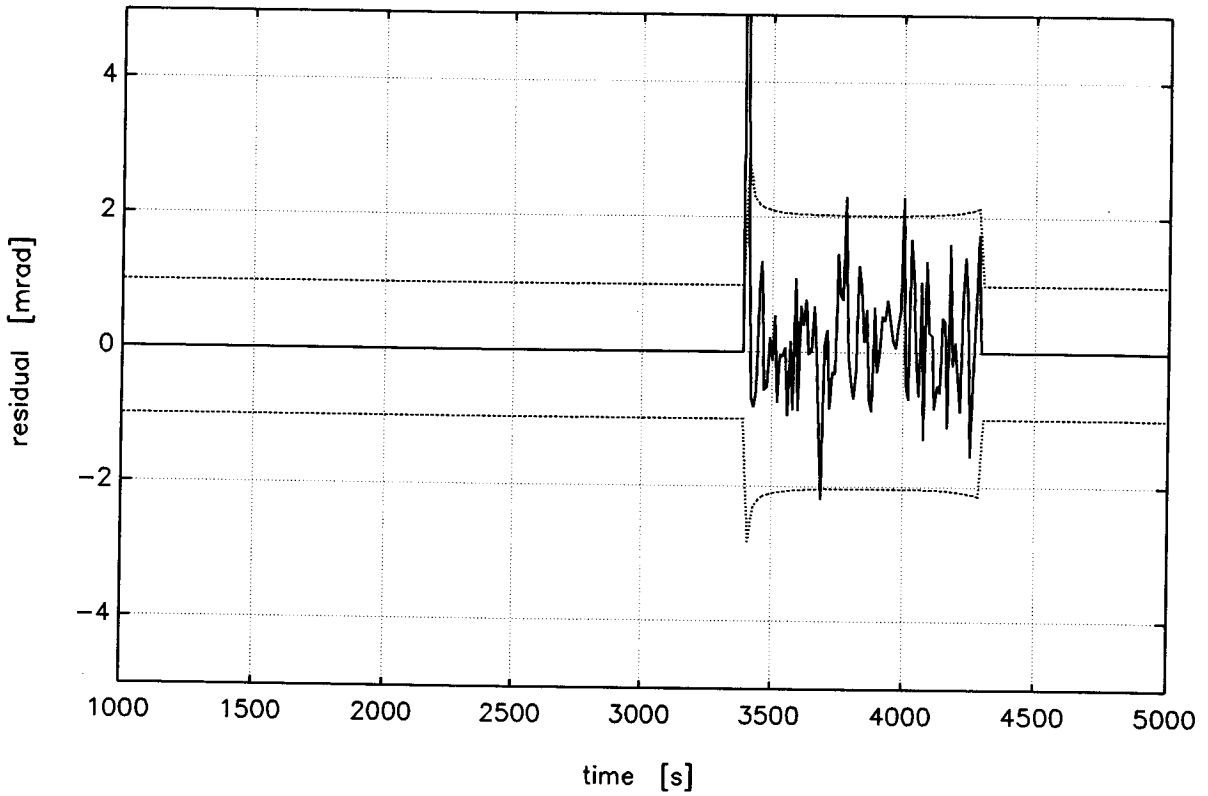


Figure 5-8 Antenna Azimuth Matching Measurement Residual

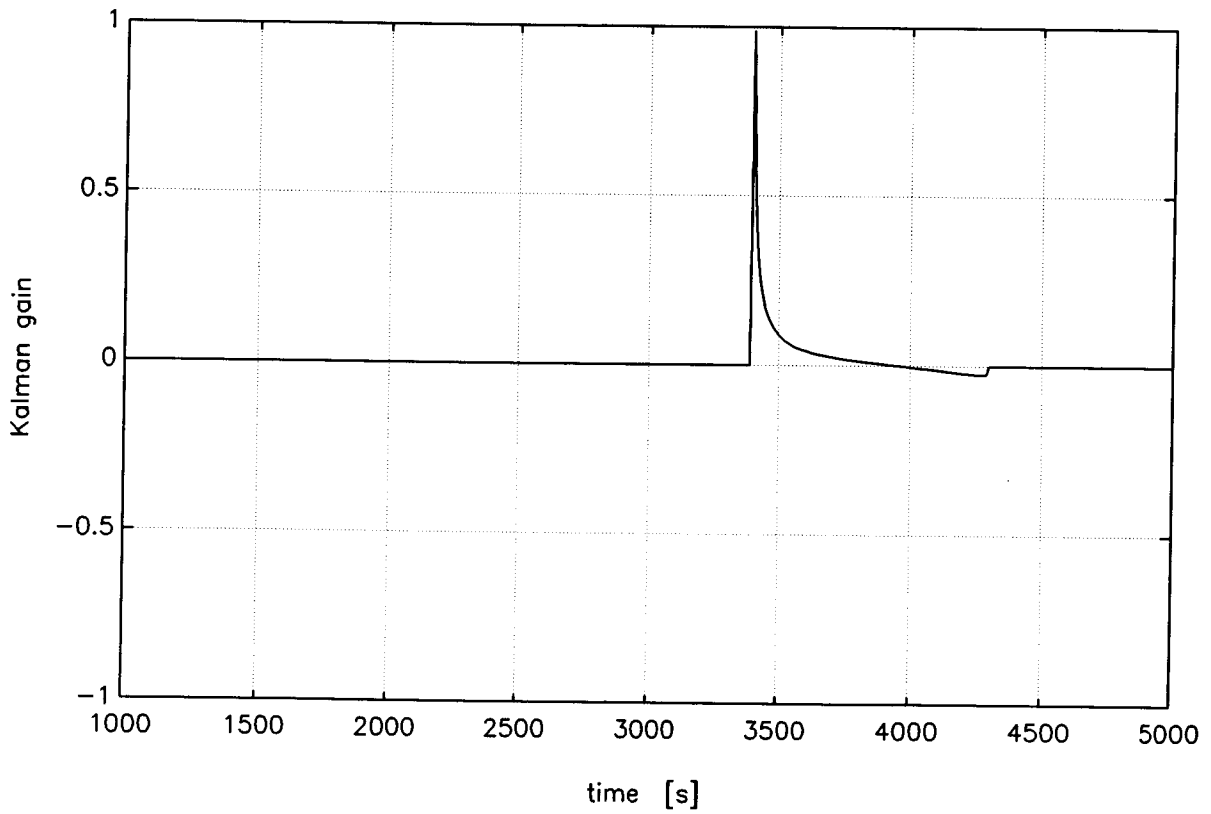


Figure 5-9 Antenna Azimuth Matching Kalman Gain for Simulation Run 2a

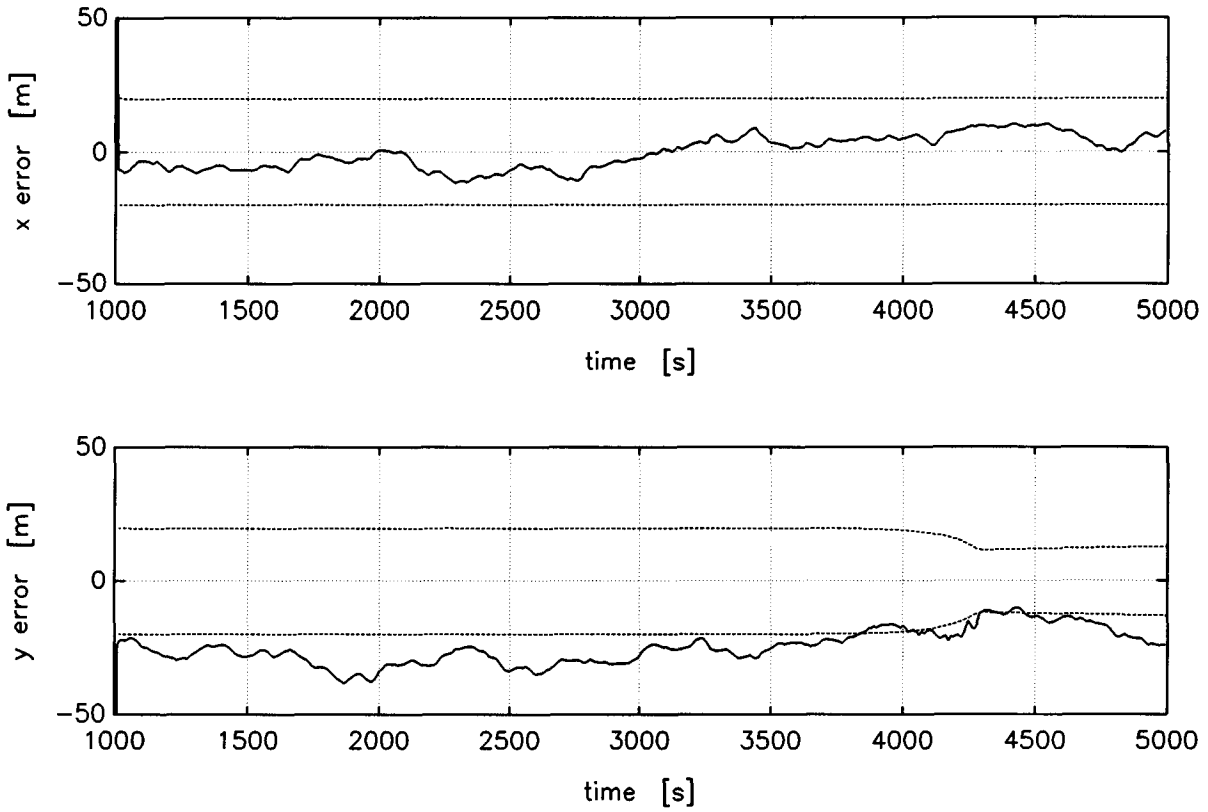


Figure 5-10 Error in Corrected IMU Position for Simulation Run 2a

5.3.4 Simulation Run 3a

Flight Profile:	#3 (see subsection 5.3.1)
Target Location:	44° 52′ 06″ N 69° 30′ 38″ W
Data Processing Start Time:	1000 s
Data Processing End Time:	5000 s
Calibration Start Time:	3700 s
Calibration End Time:	4600 s

Special Conditions:

- Data processing initiated during straight and level flight prior to manoeuvres.
- Manoeuvres consist of S-turns at 2200 seconds and 3420 seconds.
- Calibration performed during straight and level flight following manoeuvres.
- Antenna azimuth misalignment is 20 milliradians.

Plots: (see Appendix A)	3a-1	Error in x INS position error estimate
	3a-2	Error in y INS position error estimate
	3a-3	Error in x INS velocity error estimate
	3a-4	Error in y INS velocity error estimate
	3a-5	Error in x INS platform misalignment estimate
	3a-6	Error in y INS platform misalignment estimate
	3a-7	Error in z INS platform misalignment estimate
	3a-8	Error in x INS accelerometer bias estimate
	3a-9	Error in y INS accelerometer bias estimate
	3a-10	Error in x INS gyro bias estimate
	3a-11	Error in y INS gyro bias estimate
	3a-12	Error in z INS gyro bias estimate
	3a-13	Error in x GPS position error estimate
	3a-14	Error in y GPS position error estimate
	3a-15	Error in corrected x IMU position
	3a-16	Error in corrected y IMU position
	3a-17	Error in corrected x IMU velocity
	3a-18	Error in corrected y IMU velocity
	3a-19	Error in corrected x IMU platform alignment
	3a-20	Error in corrected y IMU platform alignment
	3a-21	Error in corrected z IMU platform alignment
	3a-22	Error in x IMU gyro bias estimate
	3a-23	Error in y IMU gyro bias estimate
	3a-24	Error in z IMU gyro bias estimate
	3a-25	Error in antenna azimuth misalignment estimate
	3a-26	Antenna azimuth matching measurement residual
	3a-27	Antenna azimuth matching Kalman gain

Simulation run 3a represents another potential antenna boresight calibration flight profile similar to simulation run 2a but with S-turns substituted for the 180 degree turns. For this simulation,

the S-turns were composed of two 180 degree turns in opposite directions and separated by 100 seconds of straight flight. Two S-turns separated by 1000 seconds of straight and level flight constitute the manoeuvre segment of the flight profile. Calibration is initiated following the second manoeuvre at a distance from the target of just over 100 kilometres. During the calibration segment, the aircraft flies directly toward the target and, after 900 seconds, calibration is terminated at just over 10 kilometres from the target. Results for this simulation run vary only slightly from the results for simulation run 2a. Figure 5-11 shows the error in the estimated antenna azimuth misalignment. Other combinations of 180 degree turns and S-turns could be expected to provide similar results.

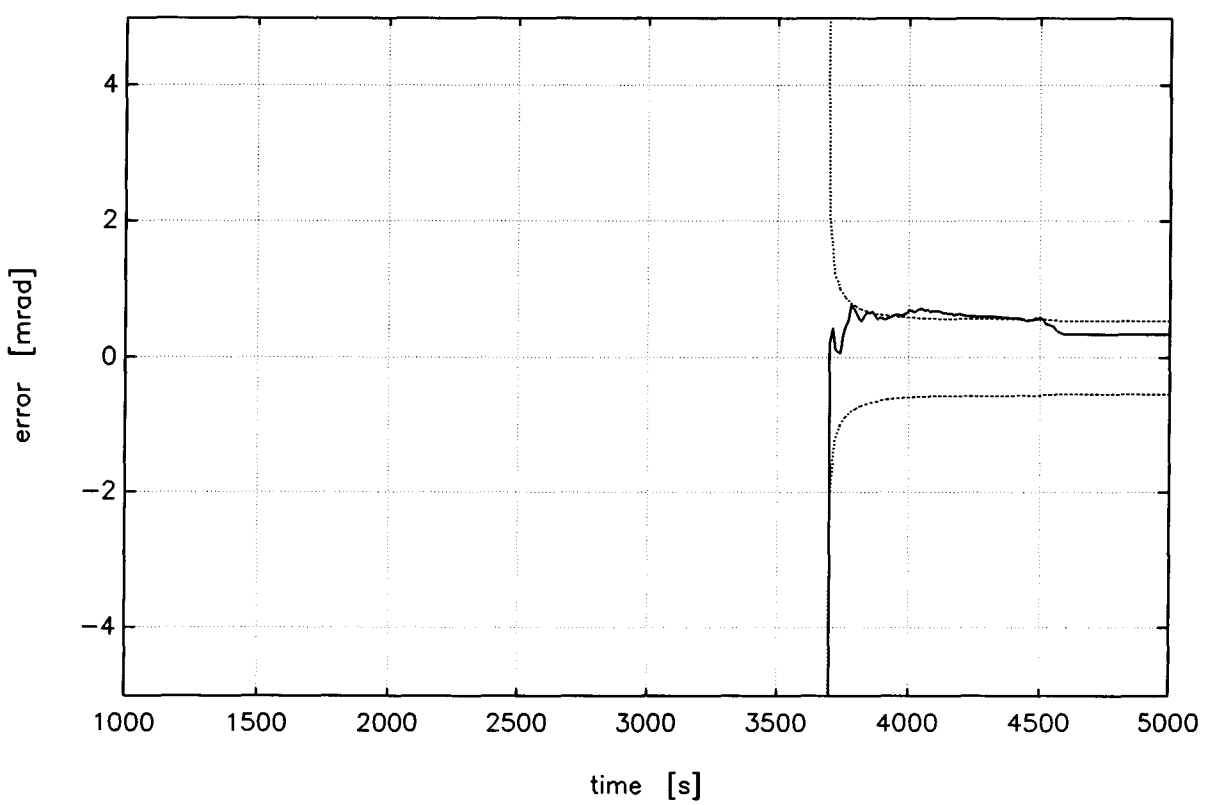


Figure 5-11 Error in Antenna Azimuth Misalignment Estimate for Simulation Run 3a

5.3.5 Summary

Accurate IMU position and azimuth alignment are important for estimating the antenna boresight misalignment. Simulation run 1a showed that Kalman filter estimated position error depends largely on the error in the GPS-indicated position and that effective estimation of z IMU misalignment can be achieved using S-turn or 180 degree turn manoeuvres. Furthermore, it was observed that minimizing z IMU gyro bias is important and that two separated manoeuvres are needed to effectively estimate this bias. Several simulation runs with various target locations and calibration start times showed that, for best calibration results, the calibration should be performed at maximum range from the target where bearing error is minimum, and immediately following the manoeuvres where z IMU platform misalignment is minimum. Simulation runs 2a and 3a conform to these conditions and represent potential antenna boresight calibration flight profiles and aircraft/target geometries. Simulation run 2a uses two 180 degree turns resulting in a compact flight profile and minimum time. Simulation run 3a uses two S-turns requiring slightly more time but maintaining a more constant direction of travel. Other combinations of 180 degree turns and S-turns may also be used. A particularly attractive combination consists of two 180 degree turns in opposite directions which minimizes both manoeuvre time and cumulative errors.

In simulation runs 2a and 3a, the Kalman filter predicted error in the estimated antenna azimuth misalignment falls to a nearly constant level near 0.6 milliradians rms (2 arc minutes rms) within 500 seconds (50 TAU measurements at 10 second intervals). This is well within the required accuracy of 5 arc minutes rms. The total of manoeuvre time plus calibration time for the runs is in the vicinity of 35 minutes. Further simulations indicate that the time between manoeuvres can be reduced from 1000 seconds to as little as 500 or 600 seconds with some degradation in

the z IMU gyro bias estimate but little impact on the antenna boresight calibration performance. Furthermore, the predicted calibration error falls to within ten percent of the final value after only about 300 seconds of calibration (30 TAU measurements). By using 600 seconds between manoeuvres and 300 to 500 seconds for azimuth misalignment estimation, the total time can be reduced to around 20 minutes.

Several other simulation runs, testing simple variations of the flight profiles and aircraft/target geometries, were performed. Variations included a change of flight profile orientation from east/west to north/south, a change of ground speed from 210 to 290 nautical miles per hour, a change of altitude from 3300 to 5000 feet and a change of latitude from 45 degrees to test values of 0 and 70 degrees. The results indicate that antenna calibration performance is essentially insensitive to the actual heading, ground speed, altitude and latitude within reasonable operating limits.

These results suggest that the in-flight antenna boresight calibration technique is indeed feasible. It has the potential to achieve antenna boresight calibration accuracies in the vicinity of 2 arc minutes rms within 20 minutes. This assumes antenna azimuth matching measurements at 10 second intervals (30 measurements in 300 seconds) with a measurement noise of 2 milliradians rms (0.05 degrees rms TAU error plus 0.1 degrees rms timing related error). Similar accuracy would be expected for different measurement intervals provided that 1) at least 30 measurements are used, and 2) the TAU range limitations are respected (maximum 100 kilometres, minimum 10 kilometres). The assumption of 0.1 degrees rms of timing related measurement noise is probably pessimistic (flight trials are required to refine this value). Simulations indicate that a smaller value might slightly improve the calibration accuracy and could substantially reduce the

number of measurements required to reach the steady-state. For example, a large reduction in the noise, from 0.1 degrees rms to 0.05 degrees rms, improved the calibration accuracy by 10 or 15 percent and required only about half as many measurements.

The filter performance is not truly optimal with respect to the synthesized sensor data because the filter models include process and measurement noise sources not in the synthesized data (gravity anomalies, for example). However, the statistical behaviour of the errors which are included in the synthesized sensor data precisely matches the error models used in the Kalman filter. In this respect, the performance is optimal and represents a baseline for subsequent performance evaluations. Because the filter retains error models to account for "real world" errors (even though not simulated in the synthesized sensor data), the calibration performance achieved here represents a reasonable "best case" performance target.

SECTION SIX

PERFORMANCE EVALUATION

The objectives of the antenna boresight calibration performance evaluation are twofold. First, the effectiveness of the formulated Kalman filter and recommended flight profile and aircraft/target geometry needs to be verified using real (or at least more realistic) sensor data. Second, since the intent is to incorporate this antenna boresight calibration method into the SARMCS, verification of its performance in the current and anticipated future SARMCS environments is important.

A full evaluation of the antenna boresight calibration performance requires real sensor data obtained during actual flights. Such data was not available for this investigation. The performance evaluation presented here, therefore, is based upon synthesized sensor data provided by DREO. This data (described in subsection 6.1) represents real SARMCS sensor data more faithfully than that used in the simulations described in Section 5.

In evaluating the antenna boresight calibration Kalman filter performance, the DREO synthesized sensor data is initially processed similarly to Section 5 for comparison with the previous results. Then the processing is modified to more closely match the anticipated SARMCS processing. This includes first, increasing the Kalman filter update interval and second, separating the INS/GPS part of the filter.

6.1 Synthesized SARMCS Sensor Data

The synthesized SARMCS sensor data provided by DREO includes INS, IMU and GPS sensor data as shown in Tables 6-1 through 6-3. Reference navigation data corresponding to the locations of the INS, IMU and GPS sensors is also provided. The reference data includes the true position (latitude, longitude, altitude), true velocity (north, east, down) and, in the case of the INS and IMU, true attitude (roll, pitch, heading) required by the evaluation package.

Table 6-1 INS Data (at 16 Hz)

Parameter	Resolution
Time	10 μs
Latitude	$180/2^{20}$ deg
Longitude	$180/2^{20}$ deg
Heading	$180/2^{15}$ deg
Pitch	$180/2^{15}$ deg
Roll	$180/2^{15}$ deg
Body pitch rate	$128/2^{15}$ deg/s
Body roll rate	$128/2^{15}$ deg/s
Body yaw rate	$128/2^{15}$ deg/s
Platform heading	$180/2^{15}$ deg
Altitude	0.125 ft
Up velocity	1 ft/min
North velocity	0.125 knots
East velocity	0.125 knots

Table 6-2 IMU Data (at 50 Hz)

Parameter	Resolution
Time	10 μs
x angular increment	3.3 arcsec
y angular increment	3.3 arcsec
z angular increment	3.3 arcsec
x velocity increment	2.5 mm/s
y velocity increment	2.5 mm/s
z velocity increment	2.5 mm/s

Table 6-3 GPS Data (at 1 Hz)

Parameter	Resolution
Time	10 μs
Latitude	2^{-23} deg
Longitude	2^{-23} deg
Altitude	1 m
North velocity	2^{-5} m/s
East velocity	2^{-5} m/s
Up velocity	2^{-5} m/s

The SARMCS data synthesis package models the sensor environment and sensor errors more faithfully than the data synthesis package described in Section 4. Specifically:

- Turbulence and flexible body dynamics are simulated.
- Vertical deflections and gravity anomalies are simulated.
- Accelerometer and gyro scale factor errors, misalignments and g-sensitive errors are simulated.
- Quantization of the synthesized sensor outputs is included.
- An additional lever arm from the antenna roll/pitch intersection to the IMU is included.

The lever arms used for the synthesized SARMCS sensor data are shown in Table 6-4. Initial INS position, velocity and attitude errors are shown in Table 6-5. Tables 6-6 and 6-7 show the accelerometer and gyro errors used for the synthesized INS and IMU sensor data.

Table 6-4 SARMCS Lever Arms

From	**To**	**Lever Arm**			**Coordinate Frame**
		***x* (m)**	***y* (m)**	***z* (m)**	
INS	Antenna roll/pitch intersection	6.05	-0.61	-0.08	Aircraft body
Antenna roll/pitch intersection	IMU	0.09	-0.24	-0.10	Antenna ring gear
INS	GPS	-4.00	0.00	-2.00	Aircraft body

Table 6-5 Initial INS System Errors

Quantity	Initial Error (wander azimuth axes)			
	x	y	z	
Position	0	0	0	m
Velocity	0	0	0	m/s
Attitude	1	1	6	arcmin

Table 6-6 INS Accelerometer and Gyro Errors

Parameter	Value	
Accelerometers		
Bias	50	μg
Scale Factor	50	ppm
Nonlinearity	10	$\mu g/g^2$
Misalignment	5	arcsec
Random Error Standard Deviation	5	μg
Random Error Correlation Time	3600	s
Gyros		
Drift	0.01	deg/hr
Scale Factor	5	ppm
Misalignment	3	arcsec
Random Drift	0.003	deg/$\sqrt{\text{hr}}$

Table 6-7 IMU Accelerometer and Gyro Errors

Parameter	Value	
Accelerometers		
Bias	100	μg
Scale Factor	200	ppm
Nonlinearity	20	$\mu g/g^2$
Misalignment	20	arcsec
Random Error Standard Deviation	15	μg
Random Error Correlation Time	3600	s
Gyros		
Drift	-0.2	deg/hr
Scale Factor	150	ppm
g-Sensitive Drift	0.2	deg/hr/g
Misalignment	20	arcsec
Random Error Standard Deviation	0.2	deg/hr
Random Error Correlation Time	1800	s

In addition to the INS, IMU and GPS sensor data, the data processing package requires altitude data and TAU antenna azimuth data. For these simulations, altitude data at two second intervals is obtained by subsampling the INS altitude, and antenna azimuth data at ten second intervals is synthesized using the IMU navigator reference data (corrected for the IMU to antenna roll/pitch intersection lever arm). The synthesized antenna azimuth data includes a fixed 20 milliradian misalignment plus an rms error of 0.05 degrees which is uncorrelated from measurement to measurement.

The evaluation data flight profile (also referred to as flight profile number four) is shown in Figure 6-1. It is flown at a nominal altitude of 3000 metres and a nominal ground speed of 100 metres per second. The profile includes two 180 degree turns in opposite directions and separated by 600 seconds of straight and level flight (turns one and two) which allow the Kalman filter to observe and estimate the azimuth misalignment. Following these manoeuvres, the profile offers two potential calibration segments. The first is the 300 second north pointing leg following turn two and the second is the 300 second south pointing leg following turn three. The "+" marks along the profile in Figure 6-1 indicate 100 second intervals.

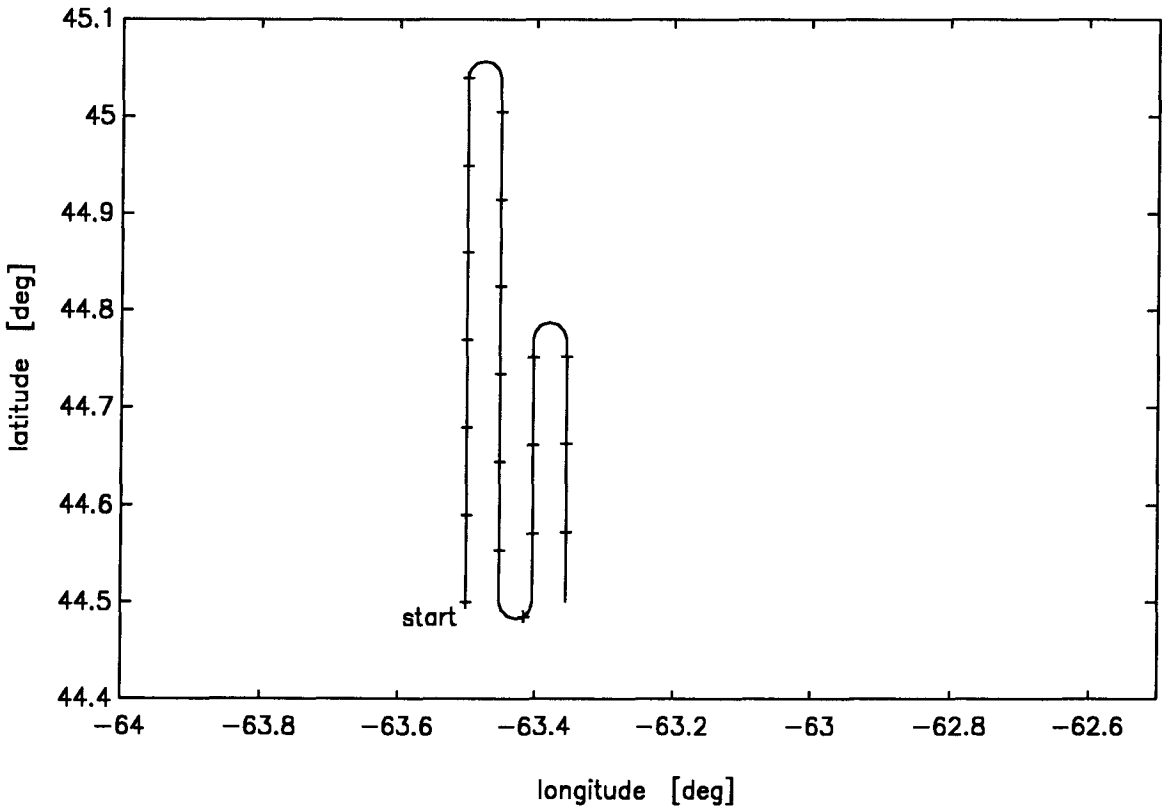


Figure 6-1 Performance Evaluation Flight Profile

6.2 Initial Performance Evaluation

For comparison purposes, it is appropriate to begin the antenna boresight calibration performance evaluation by processing the DREO synthesized sensor data similarly to Section 5. Three modifications to the data processing software are necessary. The first modification is minor. Since the earth model used in generating the evaluation data corresponds to the older WGS-72 geodetic datum rather than WGS-84 some changes to the constants used in the data processing software are necessary. It should be noted that real GPS-determined position coordinates refer to WGS-84 and that WGS-84 should be used for any real implementation of the antenna boresight calibration method.

The IMU and the antenna roll/pitch intersection are not coincident in the evaluation data. The second modification is necessary to compensate for this additional lever arm. In subsection 3.1, the expression for the IMU/INS position matching measurement is given as

$$\mathbf{z}_P = \left[\mathbf{C}_w^e\right]_S^T \left(\left[\mathbf{R}^e\right]_S - \left[\mathbf{R}^e\right]_M\right) - \left[\mathbf{C}_w^e\right]_S^T \left[\mathbf{C}_w^e\right]_M \left[\mathbf{C}_b^w\right]_M \left[\mathbf{r}_{IMU}^b\right]_M$$

But the lever arm $\mathbf{r}_{IMU}$ is now split into two parts, $\mathbf{r}_{RP}$ from the INS to the antenna roll/pitch intersection and $\mathbf{r}_{SD}$ from the roll/pitch intersection to the IMU (see Table 6-4), so that the second term in the equation becomes

$$\left[\mathbf{C}_w^e\right]_S^T \left(\left[\mathbf{C}_w^e\right]_M \left[\mathbf{C}_b^w\right]_M \left[\mathbf{r}_{RP}^b\right]_M + \left[\mathbf{C}_w^e\right]_S \left[\mathbf{C}_b^w\right]_S \left[\mathbf{r}_{SD}^b\right]_S\right)$$

where

$[\mathbf{r}^b_{RP}]_M$ = INS to antenna roll/pitch intersection lever arm vector expressed in aircraft body coordinates

$[\mathbf{r}^b_{SD}]_S$ = Antenna roll/pitch intersection to IMU lever arm vector expressed in stabilized antenna ring gear coordinates

Note that the small misalignment of the IMU body frame with respect to the antenna ring gear frame is ignored in the expression.

The third modification to the data processing software is due to the quantization of the synthesized sensor data. Quantization of the INS latitude and longitude values adds to the Kalman filter measurement noise associated with the INS/GPS position matching measurements and the IMU/INS position matching measurements. The INS latitude and longitude quantization is $180/2^{20}$ degrees (see Table 6-1) or about 19 metres in the north-south direction and between 0 and 19 metres in the east-west direction (depending on latitude). Because the wander azimuth frame is, in general, not aligned with the geographic frame, there is also a direction dependent cross-covariance between the x and y position matching measurement noise. Furthermore, the INS position quantization introduces a cross-covariance between the INS/GPS position matching measurement noise and the IMU/INS position matching measurement noise.

Quantization error is commonly assumed to be uncorrelated from measurement to measurement and uniformly distributed between $-q/2$ to $q/2$ resulting in a variance of $q^2/12$ where q is the quantization step size. The smoothly varying nature of the latitude and longitude values in the simulations makes this model less than perfect, with the quantization having correlations both in time and with the data (the quantization has a "sawtooth" appearance). Nevertheless, it is a

simple and useful model and so the quantization noise is assumed here to be uncorrelated in time with variance $q^2/12$. As will be seen in subsection 6.2.2, the effect of the quantization noise on filter performance is relatively small. Thus, there is little incentive to model the latitude and direction dependence and the x and y position matching measurement quantization noises are simply assumed to be independent with variances of 30 m^2 each.

In the absence of azimuth matching measurements, the INS/GPS and transfer-of-alignment segments of the Kalman filter formulated in Section 3 are decoupled. This is desirable because it prevents possible corruption of the INS/GPS filter states which might otherwise occur over a period of time due to slight mismodelling of the lower quality IMU. The presence of the INS position quantization, however, introduces a coupling between the two parts of the optimal Kalman filter due to the cross-correlation between the INS/GPS position matching measurements and the IMU/INS position matching measurements. A suboptimal filter, with the cross-correlation ignored, is used here since tests showed the performance of the resulting decoupled INS/GPS and transfer-of-alignment Kalman filters to be nearly the same as the optimal filter.

6.2.1 Simulation Run 4a

Flight Profile:	#4 (see subsection 6.1)
Target Location:	44° 56′ 57″ N 63° 24′ 13″ W
Data Processing Start Time:	0 s
Data Processing End Time:	1980 s

Calibration Start Time:	1320 s
Calibration End Time:	1620 s
Special Conditions:	• Calibrate during north pointing leg following turn two. • Target range is 50 kilometres at start of calibration. • One second Kalman filter update interval. • Quantized INS position used for position matching measurements.

The measure of the antenna boresight calibration Kalman filter performance is its ability to accurately estimate the antenna azimuth misalignment. Figure 6-2 shows the error in the antenna azimuth misalignment estimate for this simulation run.

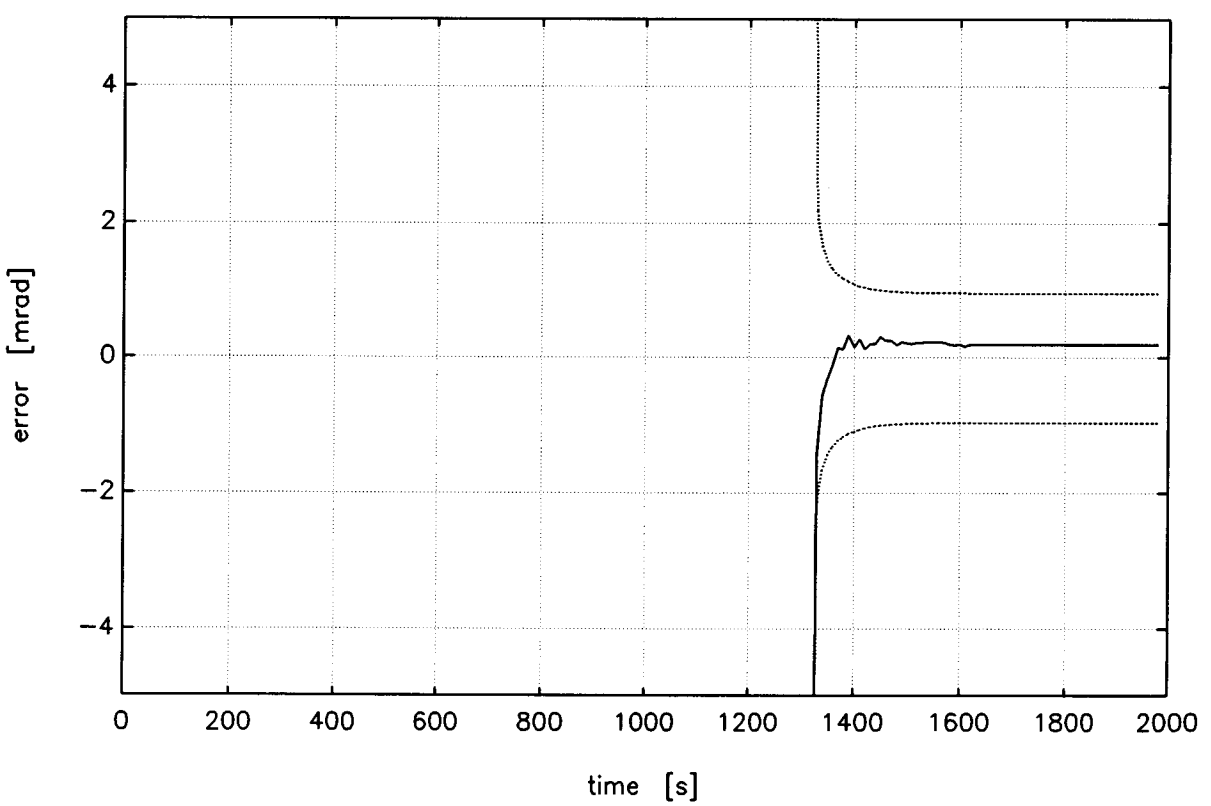


Figure 6-2 Error in Antenna Azimuth Misalignment Estimate for Simulation Run 4a

The Kalman filter predicted error is observed to be slightly less than 1 milliradian rms (about 3.3 arc minutes rms) at the completion of calibration. This is greater than the predicted errors near 2 arc minutes rms for the simulations in Section 5 but still within the required accuracy of 5 arc minutes rms. Major factors contributing to the poorer predicted performance include decreased initial target range, increased position matching measurement noise due to INS quantization, and increased process noise spectral density due to turbulence and vibration. The effects of the target range and the position matching measurement noise are examined more closely in simulation run 4b. Figure 6-3 shows the increased noise in Kalman filter position matching measurement residuals due to quantization of the INS latitude and longitude (compare Section 5, simulation run 1a).

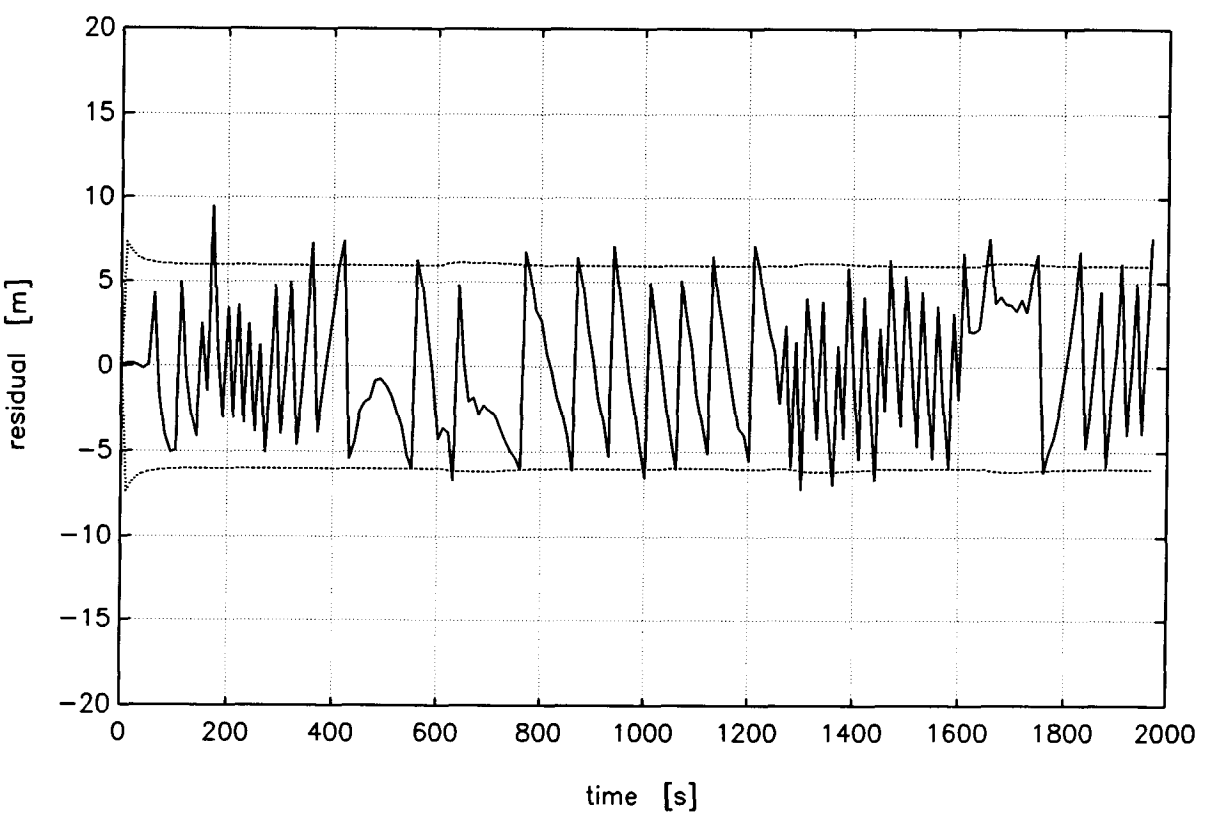


Figure 6-3 Quantization Noise in y IMU/INS Position Matching Measurement Residual

Accurate estimation of the antenna azimuth misalignment depends on how well the z IMU platform alignment and IMU position are determined. The predicted error in corrected z IMU platform alignment is increased by the added position matching measurement noise. This can be seen by comparing Figure 6-4 with the results in Section 5 and with the results of simulation run 4b in the next subsection. The filter predicted error in corrected IMU position is almost entirely due to the GPS position data rms error. This is shown in Figure 6-5 where it is seen that the predicted IMU position error immediately falls to the GPS position data rms error (20 metres) and does not change until calibration is essentially completed. In terms of antenna azimuth misalignment estimation, equation (3-9) shows that position error is more significant at shorter initial target ranges while the effect of z IMU platform misalignment is independent of target range.

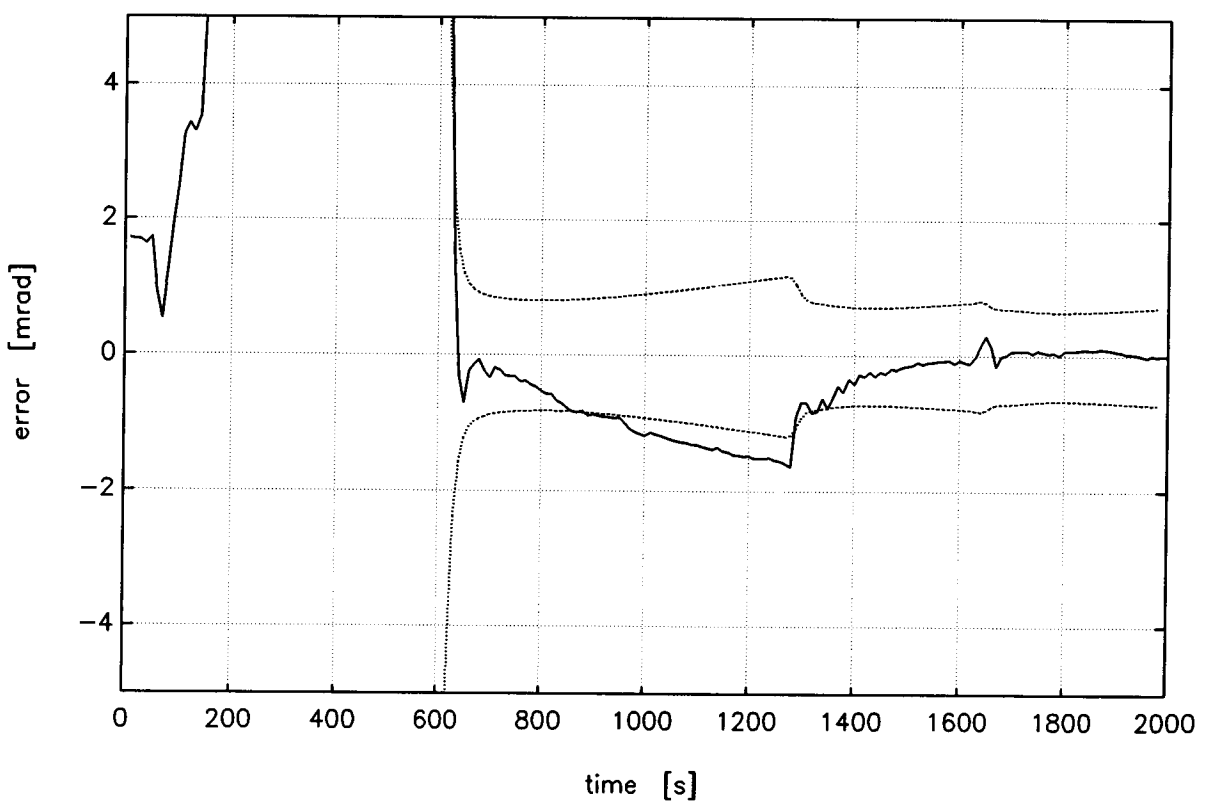


Figure 6-4 Error in Corrected z IMU Platform Alignment for Simulation Run 4a

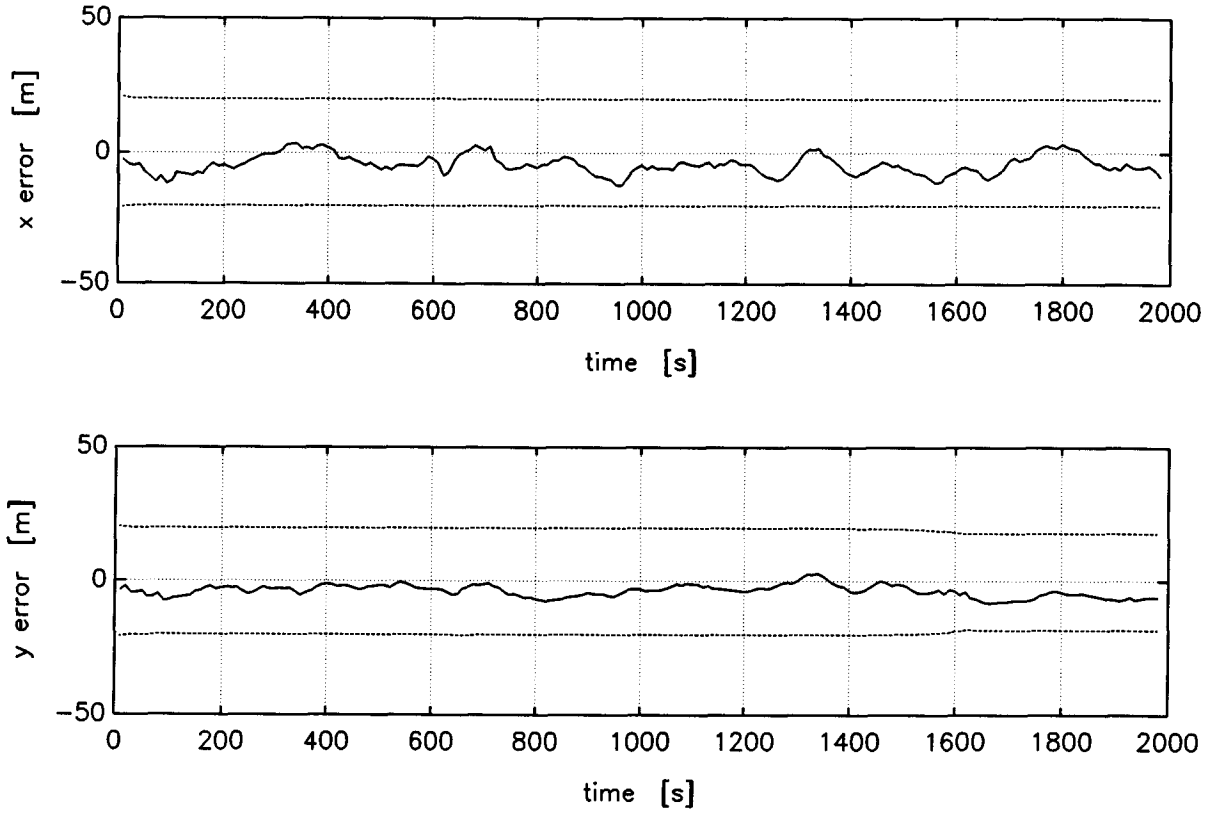


Figure 6-5 Error in Corrected IMU Position for Simulation Run 4a

6.2.2 Simulation Run 4b

Flight Profile:	#4 (see subsection 6.1)
Target Location:	45° 23′ 54″ N 63° 24′ 13″ W
Data Processing Start Time:	0 s
Data Processing End Time:	1980 s
Calibration Start Time:	1320 s
Calibration End Time:	1620 s

Special Conditions:

- Calibration during north pointing leg following turn two.
- Target range is 100 kilometres at start of calibration.
- One second Kalman filter update interval.
- Negligible quantization of INS position used for position matching measurements.

This simulation run demonstrates the effects of increased initial target range (from 50 kilometres to 100 kilometres) and decreased position matching measurement noise (INS latitude and longitude quantization is reduced to a negligible 1×10^{-7} radians) on antenna boresight calibration performance. The error in the antenna azimuth misalignment estimate is shown in Figure 6-6.

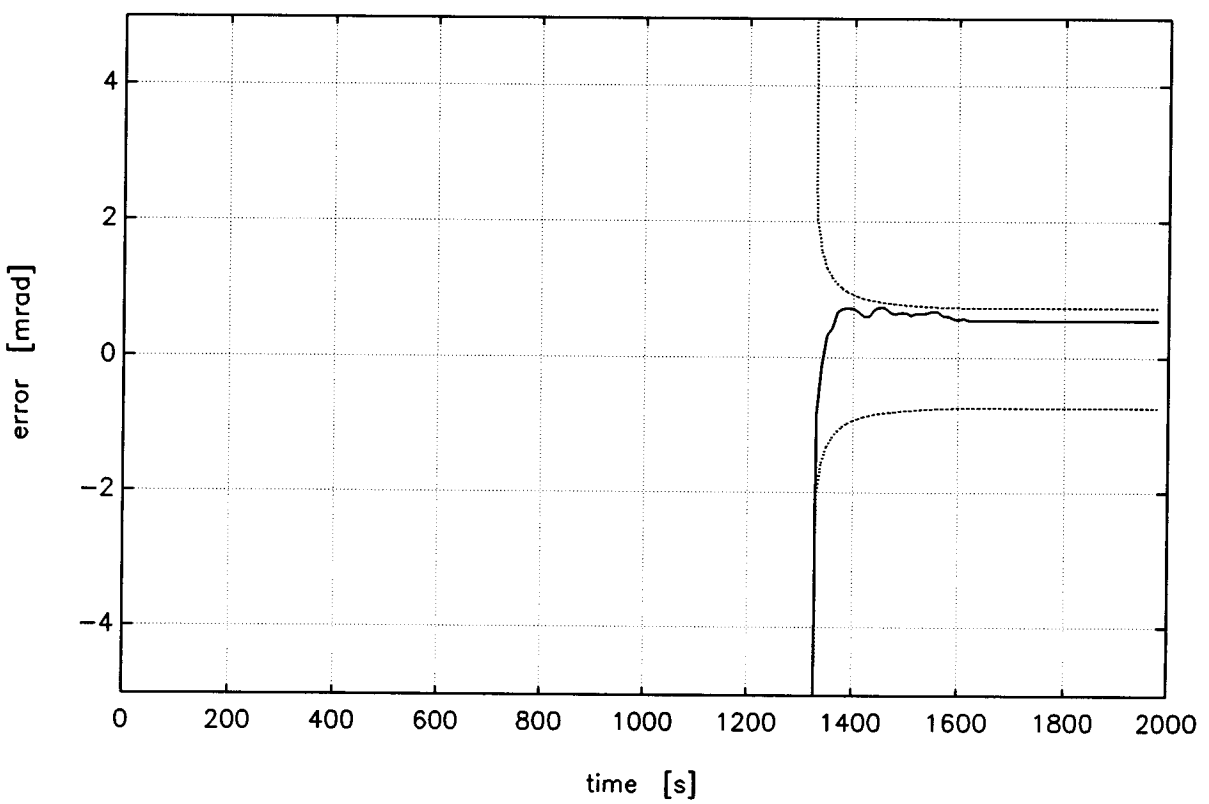


Figure 6-6 Error in Antenna Azimuth Misalignment Estimate for Simulation Run 4b

Filter predicted error for this simulation run is 0.75 milliradians rms or about 2.6 arc minutes rms versus 3.3 arc minutes rms for simulation run 4a. The errors in corrected z IMU platform alignment and corrected IMU position are shown in Figures 6-7 and 6-8 for comparison with simulation run 4a.

Simulation runs separating the target range and measurement noise effects indicate that the greatest impact on calibration performance is due to the increased target range. In particular, increased target range alone resulted in a Kalman filter predicted error in the antenna azimuth misalignment of less than 2.9 arc minutes rms while decreased position matching measurement noise alone achieved a predicted error of only about 3.1 arc minutes rms.

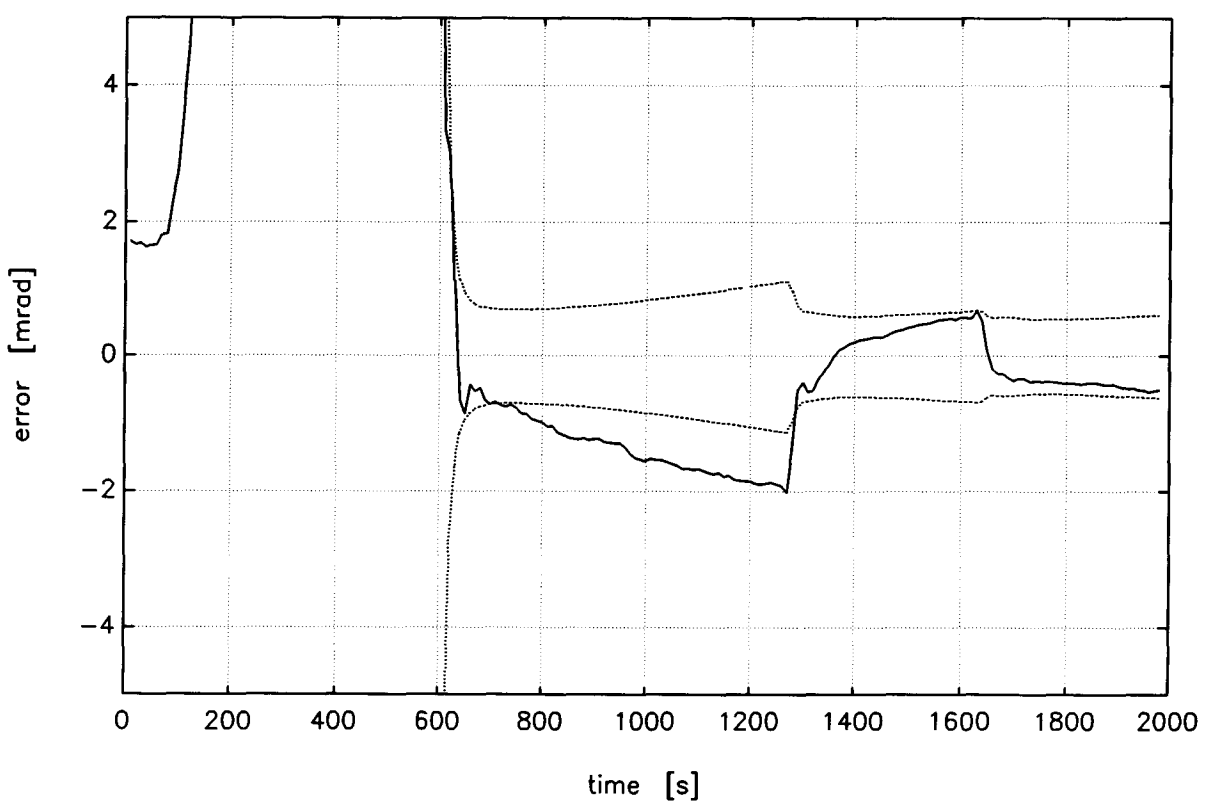


Figure 6-7 Error in Corrected z IMU Platform Alignment for Simulation Run 4b

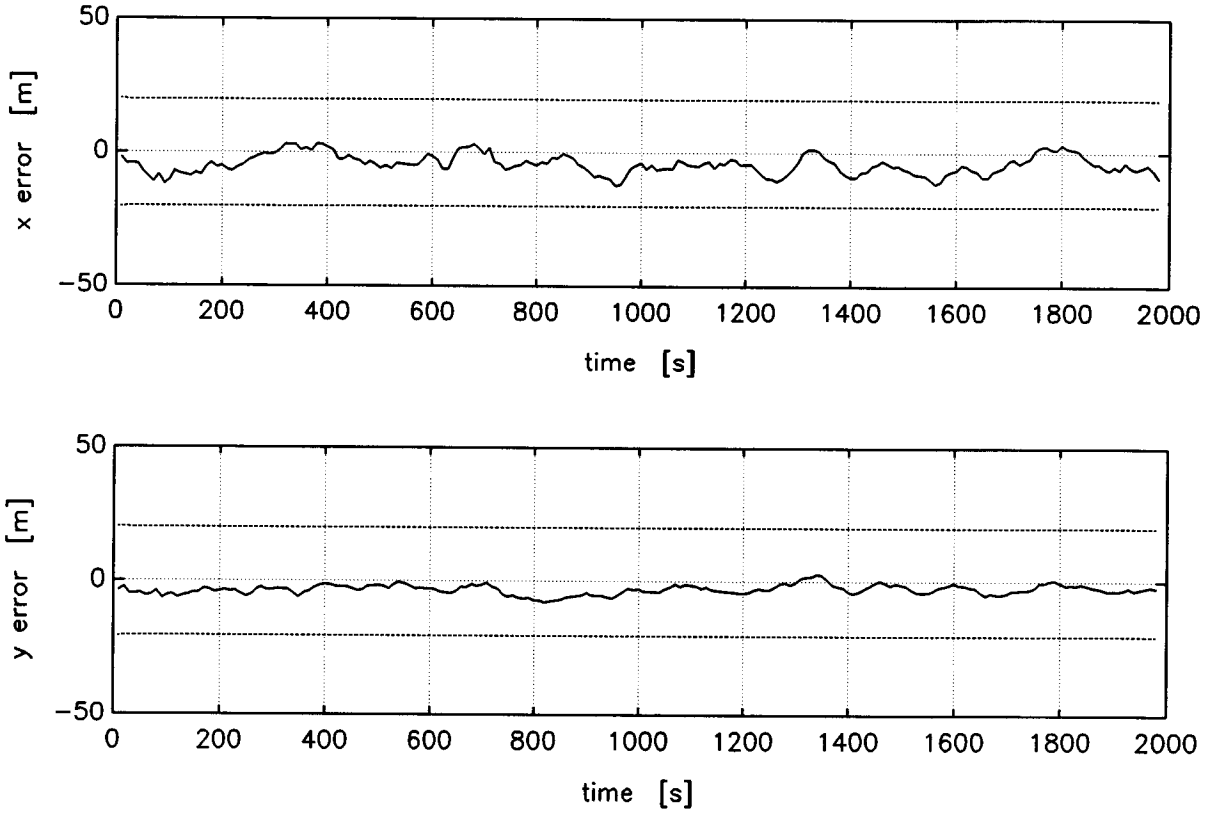


Figure 6-8 Error in Corrected IMU Position for Simulation Run 4b

6.3 Increased Kalman Filter Update Interval

For the SARMCS to achieve real-time operation, it is important to minimize the computational load on the small airborne computers. Currently, the SARMCS performs Kalman filter updates at ten second intervals. To test the effect of increasing the interval between Kalman filter updates on calibration performance, simulation run 4a is repeated here using ten second updates instead of one second updates. A simulation using the south pointing calibration leg of the flight profile is also performed.

6.3.1 Simulation Run 4c

Flight Profile:	#4 (see subsection 6.1)
Target Location:	44° 56′ 57″ N 63° 24′ 13″ W
Data Processing Start Time:	9 s
Data Processing End Time:	1980 s
Calibration Start Time:	1320 s
Calibration End Time:	1620 s
Special Conditions:	• Calibration during north pointing leg following turn two. • Target range is 50 kilometres at start of calibration. • Ten second Kalman filter update interval. • Quantized INS position used for position matching measurements.

This simulation run is intended to approximate the operation of the antenna boresight calibration Kalman filter within the current SARMCS. The Kalman filter update interval is increased to 10 seconds for the run. All other aspects of the simulation are the same as simulation run 4a. Figure 6-9 shows the error in the antenna azimuth misalignment estimate. The longer interval between Kalman filter updates has increased the filter predicted error in the antenna misalignment estimate to 1.1 milliradians or about 3.8 arc minutes. Also, a slight increase in the filter settling time is just observable between Figures 6-2 and 6-9. The 3.8 arc minute rms error is still within the required accuracy of 5 arc minutes rms and, as suggested in subsection 6.2.2, the figure could probably be slightly improved by increasing the initial target range.

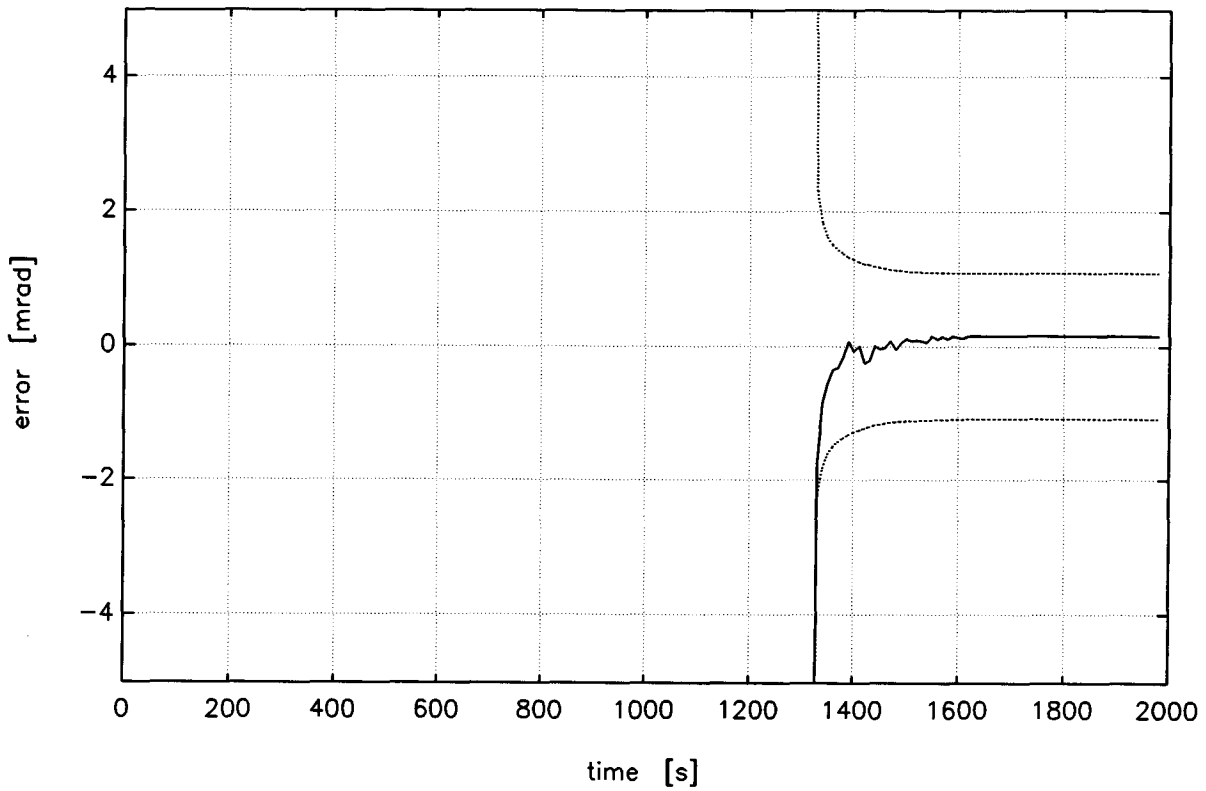


Figure 6-9 Error in Antenna Azimuth Misalignment Estimate for Simulation Run 4c

The error in the corrected IMU position and the error in the corrected z IMU platform alignment are shown Figures 6-10 and 6-11. It can be seen that the longer update interval has resulted in only a small increase in the filter predicted position error. The filter predicted error in corrected z IMU platform alignment, on the other hand, has been substantially increased by the longer update interval and accounts for much of the deterioration of the predicted error in the antenna misalignment estimate. Simulation run 4c best represents the operation of the calibration Kalman filter in the SARMCS and, as such, a full set of plots for the run are included in Appendix B.

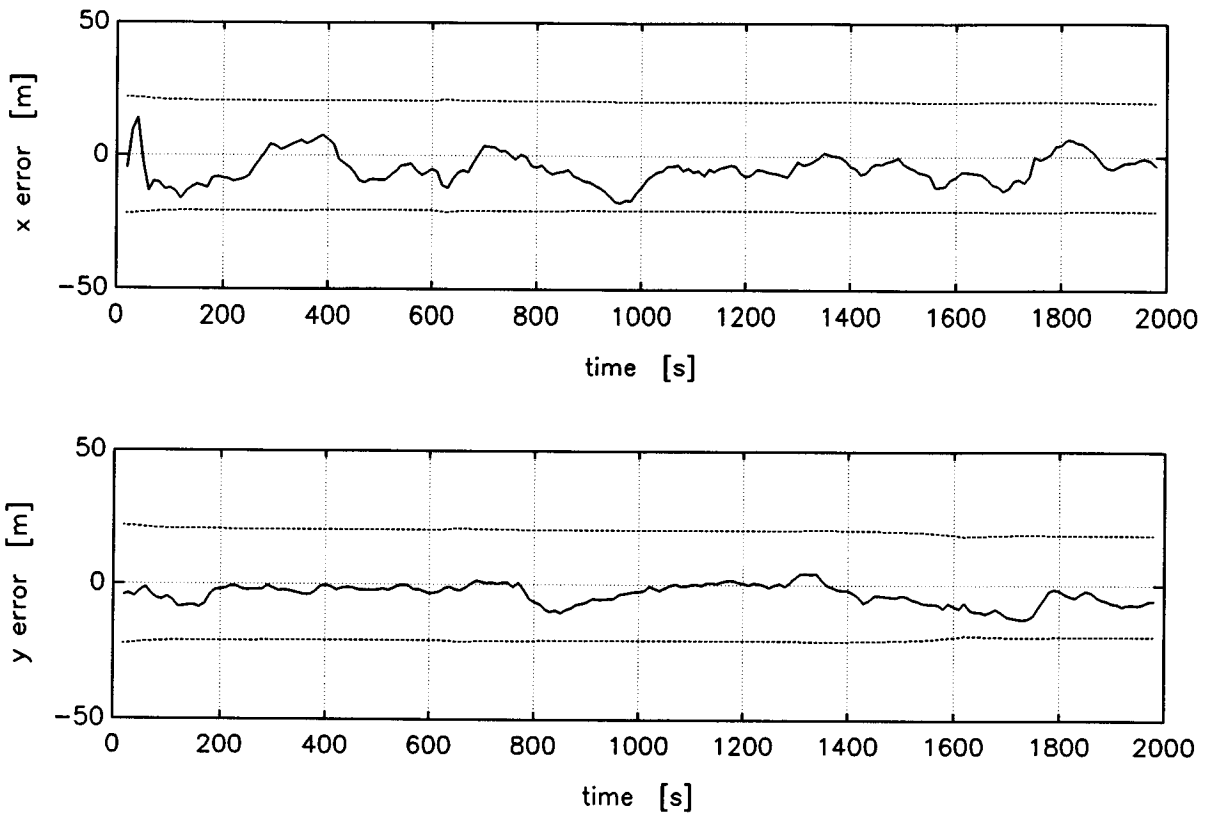


Figure 6-10 Error in Corrected IMU Position for Simulation Run 4c

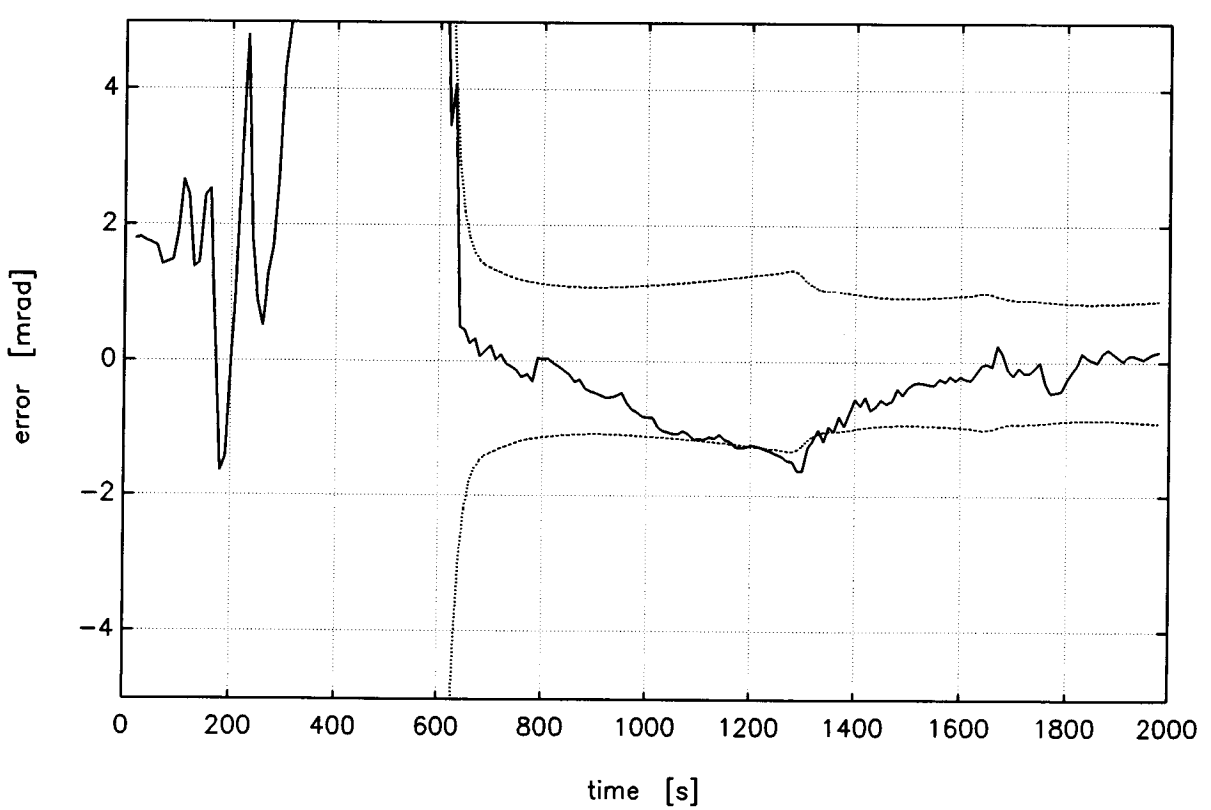


Figure 6-11 Error in Corrected z IMU Platform Alignment for Simulation Run 4c

6.3.2 Simulation Run 4d

Flight Profile:	#4 (see subsection 6.1)
Target Location:	44° 19′ 17″ N 63° 21′ 19″ W
Data Processing Start Time:	9 s
Data Processing End Time:	1980 s
Calibration Start Time:	1680 s
Calibration End Time:	1980 s
Special Conditions:	• Calibration during south pointing leg following turn three. • Target range is 50 kilometres at start of calibration. • Ten second Kalman filter update interval. • Quantized INS position used for position matching measurements.

This simulation run is similar to simulation run 4c but calibration is performed during the south pointing leg of the flight profile. It is intended to provide some additional verification of the Kalman filter performance. As Figure 6-12 shows, the filter predicted rms error in the antenna azimuth misalignment is comparable to simulation run 4c and the estimated misalignment is within the predicted rms error limits.

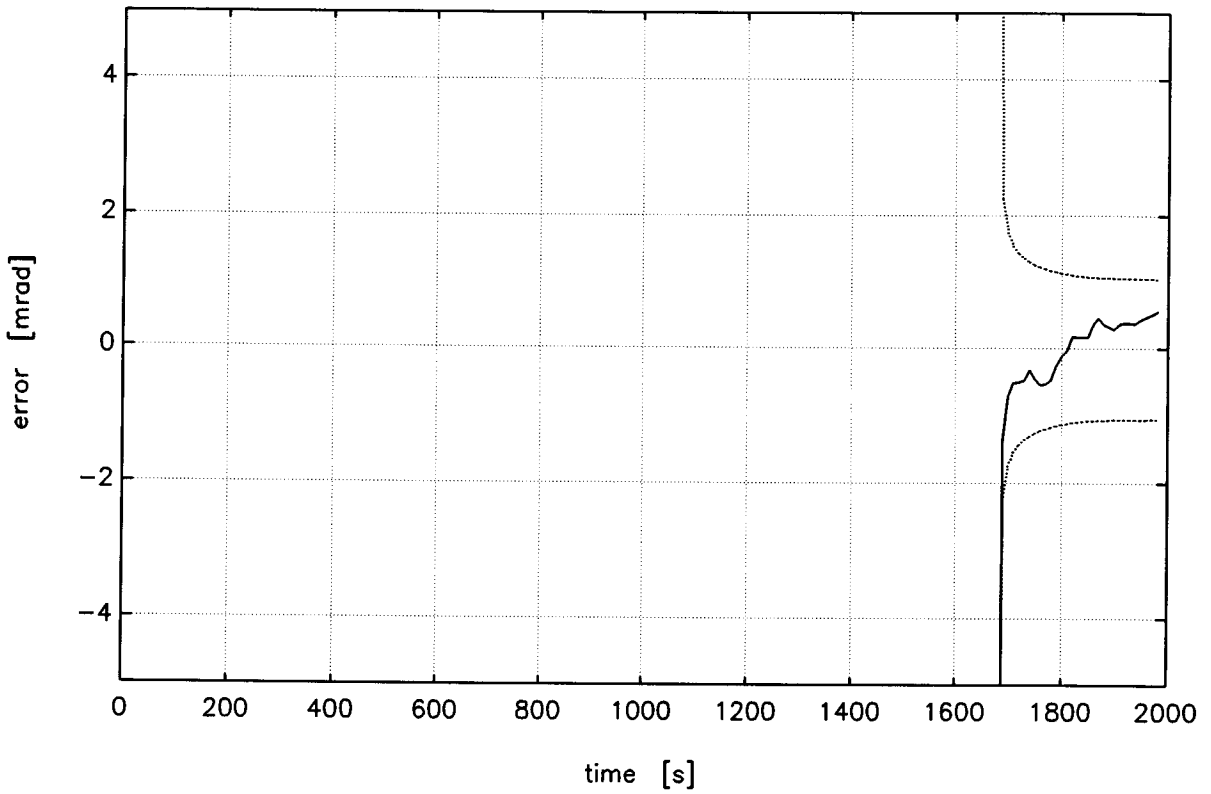


Figure 6-12 Error in Antenna Azimuth Misalignment Estimate for Simulation Run 4d

6.4 Separated INS/GPS Kalman Filter

The INS and GPS sensors currently used in the SARMCS are expected to be replaced by an integrated INS/GPS system in the future. In this scenario, the INS/GPS part of the Kalman filter is included in the INS/GPS system and is separate from the transfer-of-alignment and antenna azimuth calibration part of the filter. It is of interest, therefore, to evaluate the antenna boresight calibration performance in this cascaded Kalman filter configuration.

For this simulation, it is assumed that the INS/GPS system outputs include not only the estimated position, velocity and attitude navigation information but also that both the uncorrected INS data

and the estimated INS errors are available. In particular, the uncorrected INS position is needed for IMU/INS position matching measurements and the estimated INS errors are needed for the IMU navigator error control feedback and for the antenna azimuth matching measurements. It is further assumed that the resolution of these outputs is high enough that quantization errors can be neglected.

To incorporate an approximation for a separate INS/GPS system into the simulation, the INS/GPS part of the Kalman filter must be completely independent. Coupling of the INS/GPS part of the Kalman filter with the rest of the filter occurs through the antenna azimuth matching measurement model. The antenna azimuth misalignment measurement is given by

$$z_A = \vartheta_{TAU} - \beta + \psi$$

where ϑ_{TAU} is the TAU-determined antenna azimuth, β is the target bearing computed from the IMU navigator position and the known target location, and ψ is the IMU navigator heading. The corresponding measurement model is given in equation (3-9). For convenience, the complicated coefficient expressions in this equation are replaced here by the variables a and b and the equation is rewritten as

$$z_A = a(\delta R_{Sx} + \delta R_{Mx}) + b(\delta R_{Sy} + \delta R_{My}) + (\phi_{Sz} + \phi_{Mz}) + \varepsilon + v_A$$

where v_A is the noise associated with the TAU measurement.

To decouple the INS/GPS portion of the Kalman filter, the measurement can be redefined as

$$z=\vartheta_{TAU}-\beta+\psi-a\delta\hat{\mathrm{R}}_{Mx}-b\delta\hat{\mathrm{R}}_{My}-\hat{\phi}_{Mz}$$

where the hats indicate states estimated by the INS/GPS Kalman filter. Thus, the outputs of the INS/GPS Kalman filter become inputs to the calibration Kalman filter. The measurement model is then

$$z=a\delta\mathrm{R}_{Sx}+b\delta\mathrm{R}_{Sy}+\phi_{Sz}+\varepsilon+v$$

where v is the noise associated with the TAU measurement plus the random errors associated with the estimated INS/GPS states. But the action of the INS/GPS filter makes the random errors in its outputs complicated time correlated sequences. A simplistic approach to this problem is just to assume that the outputs are not correlated and design a suboptimal calibration Kalman filter. Schlee et al. (1988) have successfully used this technique to integrate the outputs from a GPS receiver internal Kalman filter with INS data in a cascaded external Kalman filter. A more theoretically sound approach for cascading Kalman filters has been developed by Carlson (1988) but is beyond the scope of this thesis. The suboptimal approach is used here.

One way to account for the additional antenna azimuth matching measurement noise, due to the errors in the estimated INS/GPS states, is to assume that the errors are white sequences with variances given by the error covariance matrix **P**. Assuming that the TAU measurement noise

is independent from the state estimation errors, the azimuth matching measurement noise variance is

$$\sigma_v^2=\sigma_{v_A}^2+a^2p_{\delta R_{Mx}}+b^2p_{\delta R_{My}}+p_{\phi_{Mz}}+2abp_{\delta R_{Mx}/\delta R_{My}}+2ap_{\delta R_{Mx}/\phi_{Mz}}+2bp_{\delta R_{My}/\phi_{Mz}}$$

where the p variables are the covariances of the INS/GPS state errors obtained from the error covariance matrix. A value for the additional measurement noise variance can be approximated by ignoring the cross terms appearing in the equation and using typical aircraft/target geometry to specify values for a^2 and b^2, and typical values of the INS/GPS state error variances obtained from previous simulation results. A value of 1.0 mrad^2 is used for the simulation.

The effect on calibration performance of ignoring the correlations can be predicted by modelling the error in the relevant INS/GPS states as a bias plus uncorrelated noise. This is a reasonable approximation assuming a relatively short time interval and straight and level flight, as is the case when processing azimuth matching measurements. The calibration Kalman filter would then be expected to obtain a biased estimate of the antenna azimuth misalignment. As an example, consider initiating calibration at a distance of 50,000 metres from the target with a master navigator position error bias of 20 metres perpendicular to the target direction and a master navigator azimuth misalignment bias of 0.1 milliradians. The resulting bias in the estimated antenna azimuth misalignment is nearly 2 arc minutes which, due to the white noise assumptions, the filter estimated error covariance does not account for. This also adversely affects the IMU navigator error states through the measurement model coupling. Partial compensation for these problems can be achieved by adding a small white noise forcing function

to the antenna azimuth misalignment state and tuning the spectral density so that the filter estimated antenna azimuth error covariance reaches a steady-state value near that of the optimal filter. A spectral density of 0.005 $(\text{mrad/s})^2/\text{Hz}$ is used for the simulation.

6.4.1 Simulation Run 4e

Flight Profile:	#4 (see subsection 6.1)
Target Location:	44° 56′ 57″ N 63° 24′ 13″ W
Data Processing Start Time:	9 s
Data Processing End Time:	1980 s
Calibration Start Time:	1320 s
Calibration End Time:	1620 s
Special Conditions:	• Calibration during north pointing leg following turn two. • Target range is 50 kilometres at start of calibration. • Ten second Kalman filter update interval. • Negligible quantization of INS position used for position matching measurements. • Separated INS/GPS Kalman filter.

Many simulation runs were performed to test the performance of this ad hoc design for a calibration Kalman filter incorporating a separated INS/GPS system. The simulations used both the north pointing and south pointing calibration legs of the flight profile as well as synthesized

TAU data with differing noise components. The performance of the calibration filter was adequate in all of the simulations with an error of less than 5 arc minutes in the antenna azimuth misalignment estimation in every case and less than 3.3 arc minutes in most cases. Simulation run 4e is a representative run. Figure 6-13 shows the error in the antenna azimuth misalignment estimate for this run. The error in the corrected IMU position and the error in the corrected z IMU platform alignment are shown in Figures 6-14 and 6-15. In Figure 6-16, it is observed that the Kalman gain connecting the azimuth matching measurement residuals to the antenna azimuth misalignment estimate levels out at a nonzero value. This is due to the addition of process noise to the antenna azimuth misalignment state model.

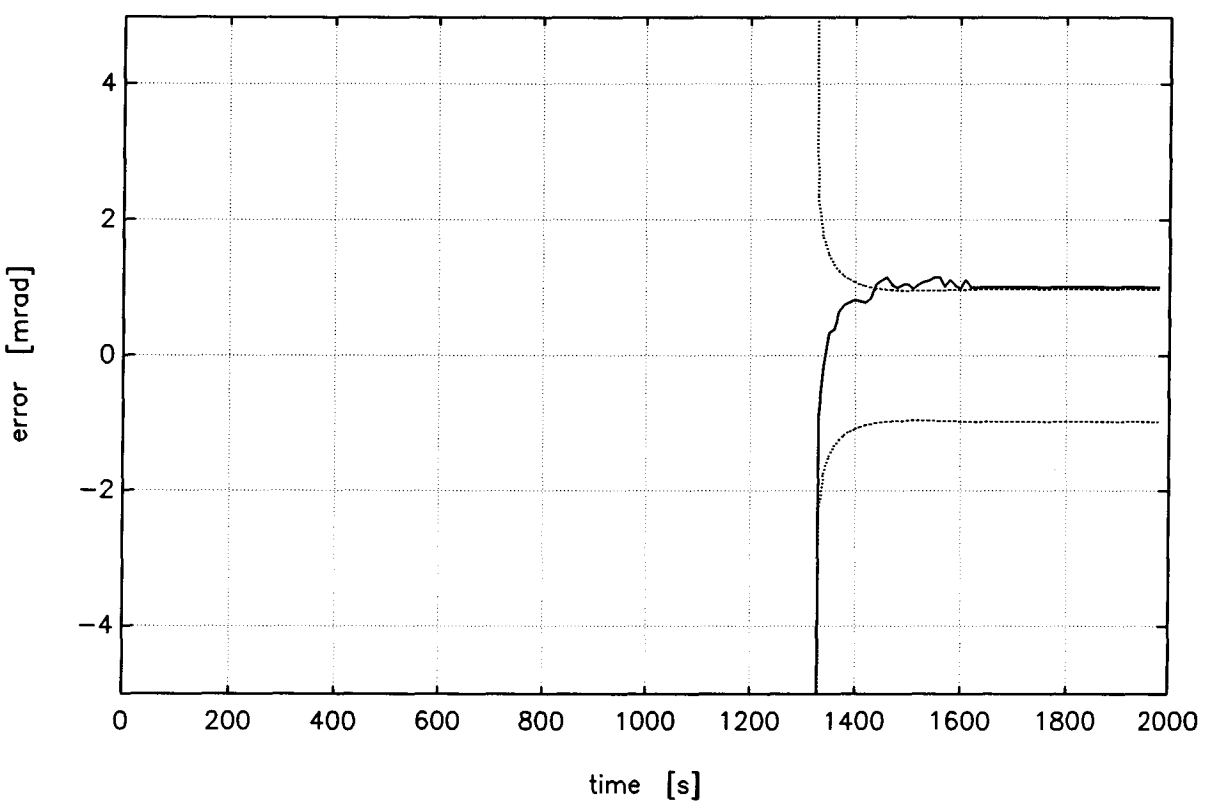


Figure 6-13 Error in Antenna Azimuth Misalignment Estimate for Simulation Run 4e

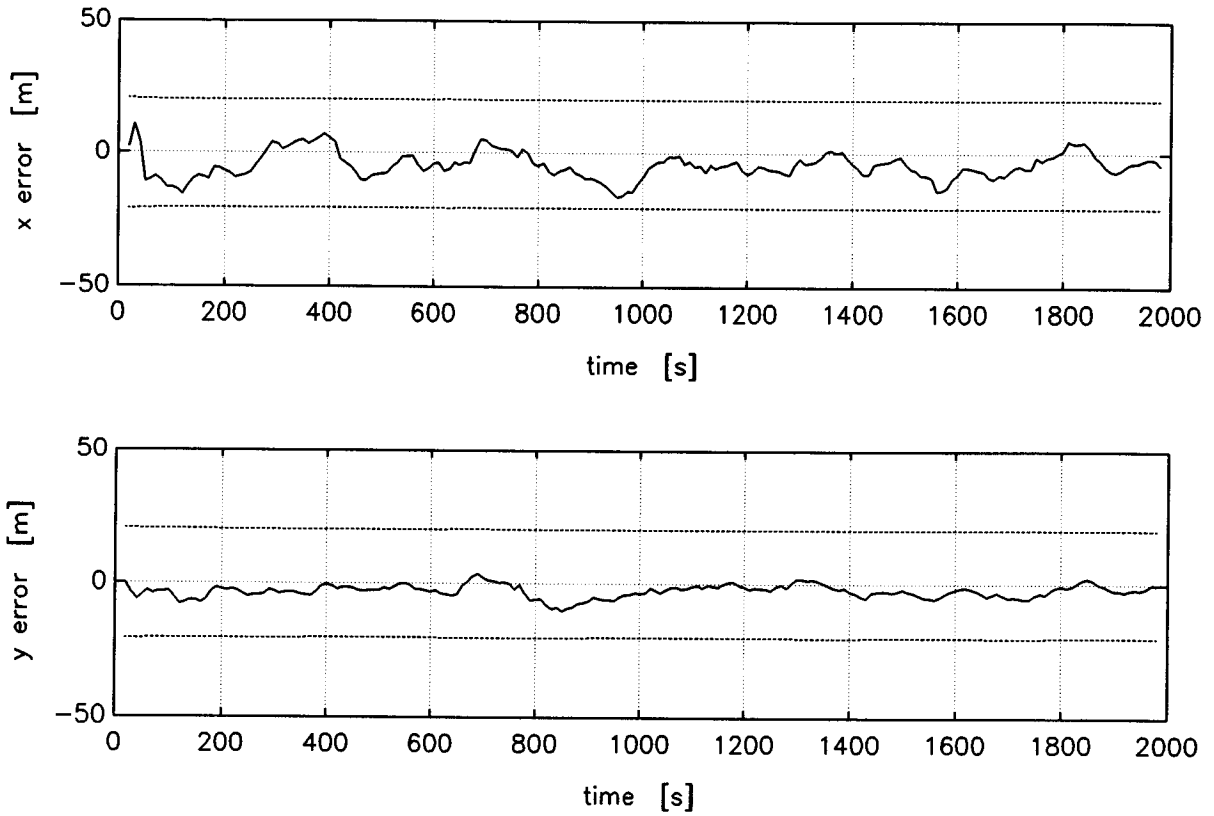


Figure 6-14 Error in Corrected IMU Position for Simulation Run 4e

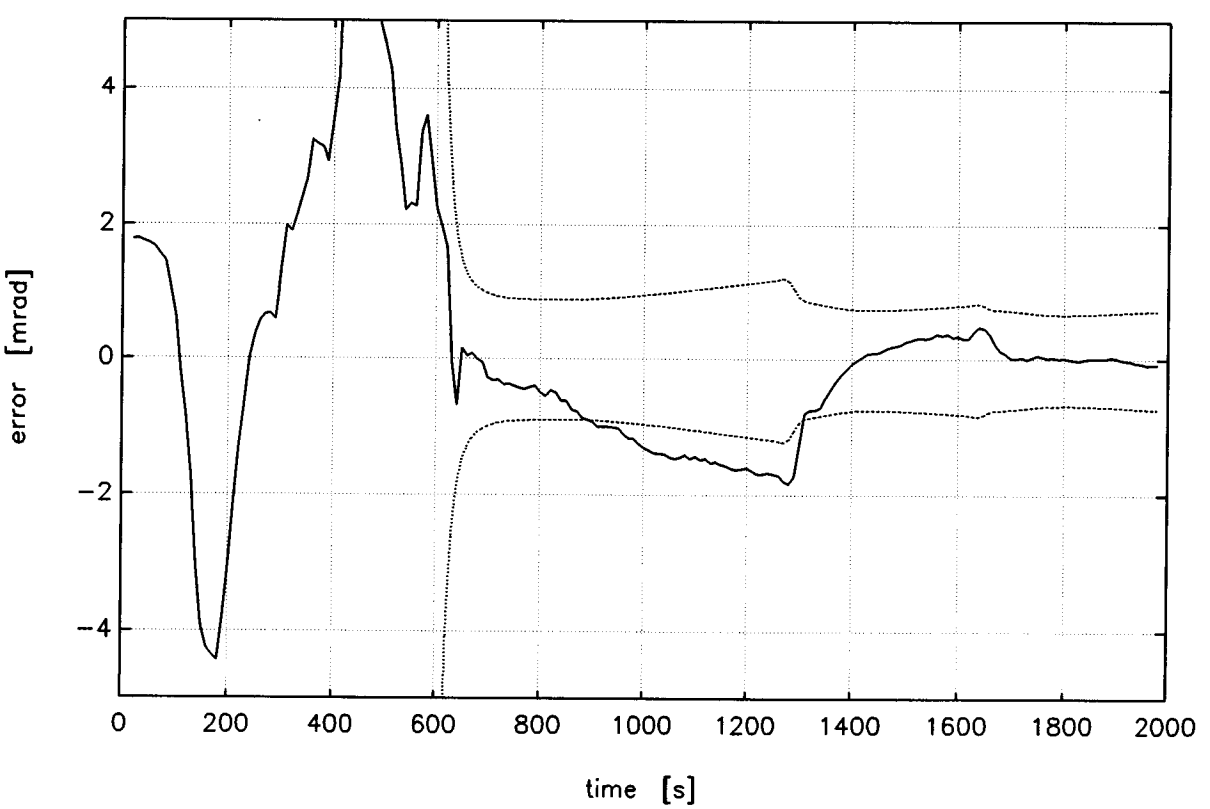


Figure 6-15 Error in Corrected z IMU Platform Alignment for Simulation Run 4e

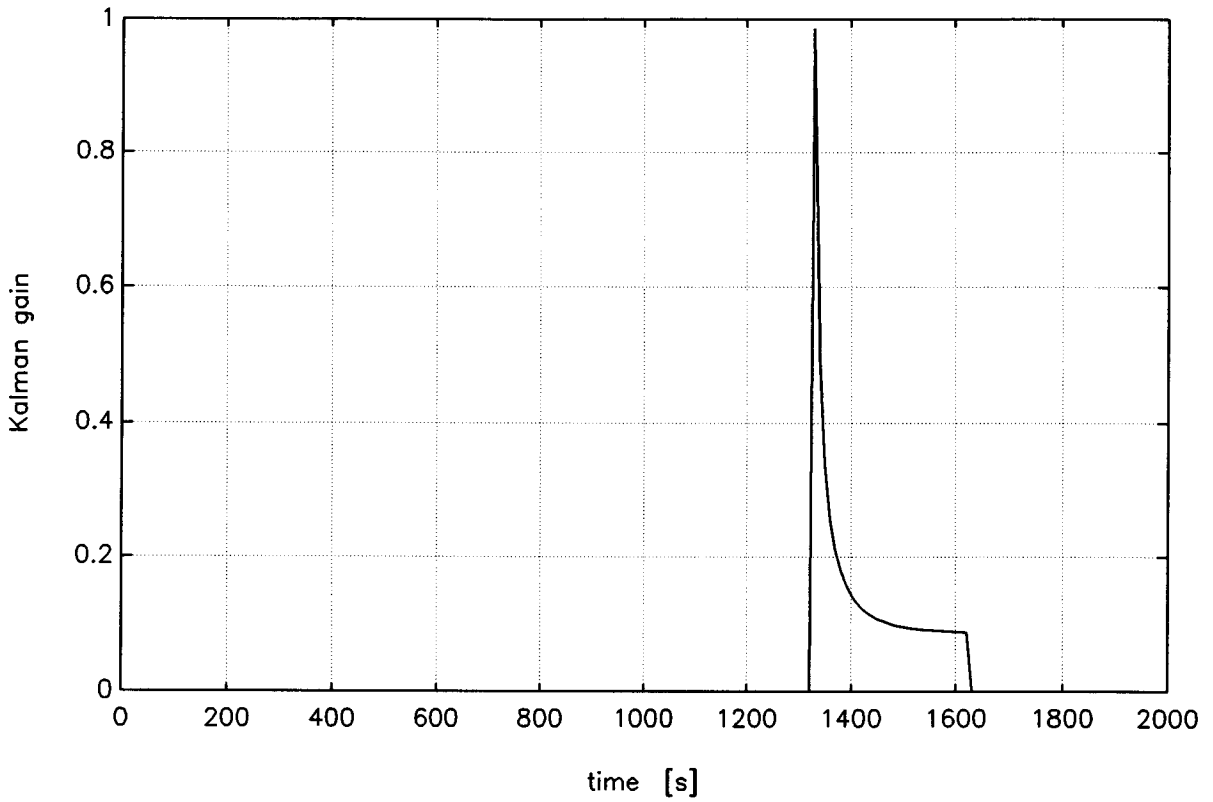


Figure 6-16 Antenna Azimuth Matching Kalman Gain for Simulation Run 4e

Since it is only the azimuth matching measurements and the antenna azimuth misalignment state that have been modified to accommodate the separate INS/GPS system, filter operation during the manoeuvre portion of the flight profile is unaffected. In fact, even during calibration, the effect of the antenna azimuth measurements on the navigation error and sensor error states of the calibration Kalman filter is very weak. This can be confirmed by comparing the plotted results of a simulation run with calibration enabled to the same run with calibration disabled. Such plots are virtually identical. Thus, it can be concluded that the filter modifications are not adversely affecting the transfer-of-alignment part of the filter. Furthermore, even when the INS/GPS part of the filter is not separated, the azimuth matching measurements significantly

affect only the position error estimates and then only at close range to the target and only after estimation of the antenna azimuth misalignment is essentially completed. It is, therefore, quite reasonable to consider the antenna azimuth misalignment estimation part of the Kalman filter in isolation from the rest. This is essentially what has been done to design the calibration Kalman filter for a separated INS/GPS system. The resulting performance appears to be quite acceptable.

SECTION SEVEN

CONCLUSIONS AND RECOMMENDATIONS

This thesis has presented the theoretical background and description of a method for estimating SAR antenna azimuth misalignment in-flight by means of a Kalman filter. The method, which was developed for the Canadian Department of National Defence Spotlight SAR program, offers a simple, quick and accurate operational procedure to replace cumbersome laser based procedures.

7.1 Kalman Filter Design

A 25-state antenna boresight calibration Kalman filter was formulated. It is based on the DREO designed SARMCS Kalman filter. The SARMCS filter uses GPS-indicated position to damp the Schuler oscillations in a medium accuracy INS. Simultaneously, it transfers the resulting stable platform alignment to an IMU navigator that provides position, velocity and heading information at the SAR antenna. The antenna boresight calibration Kalman filter adds a new antenna azimuth matching measurement and a random constant state representing antenna azimuth misalignment to the SARMCS filter.

The existing SARMCS Kalman filter was chosen to form the basis for the antenna boresight calibration Kalman filter because antenna boresight calibration is intended to become an operational mode of the SARMCS and because performance of the SARMCS filter is well proven in actual flight trials. It was shown that accurate position and heading error estimates are prerequisites to accurate antenna azimuth calibration. In this respect, desirable

characteristics of the current SARMCS Kalman filter are its effective control of platform azimuth misalignment and its bounding of position errors by GPS position measurements.

The choice of the random constant model for the antenna azimuth misalignment state was based on the assumption that the mechanical misalignment is fixed. It was argued that a constant is justified for the calibration filter because the calibration time is short. A more complicated model, such as a random walk process or an exponentially correlated process with a long time constant, is unnecessary.

The calibration Kalman filter directly observes antenna azimuth misalignment using azimuth matching measurements. These measurements are defined as the difference between the TAU-determined antenna azimuth and the antenna azimuth computed from IMU navigator information and the known target location. In forming the azimuth matching measurements, it is important not to introduce errors by mixing geodetic datums. In particular, since the GPS-determined position coordinates refer to the WGS 84 geodetic datum, the target position coordinates and the earth model constants should also refer to the WGS 84 system.

7.2 Test Methods

An extensive computer simulation software package, consisting of approximately 9000 lines of C++ code and 2000 lines of MATLAB M code, was developed to test the antenna boresight calibration Kalman filter. It is divided into a synthesis package, a processing package and an evaluation package. The synthesis package features the ability to quickly define and generate an arbitrary flight trajectory and to synthesize the corresponding sensor data with major sensor error sources included. The processing package (listed in Appendix C) implements the IMU

strapdown navigator algorithm and the calibration Kalman filter. It inputs sensor data and outputs time histories of various navigation and Kalman filter quantities for subsequent analysis. The evaluation package inputs truth data from the synthesis package and output data from the processing package and features the ability to automatically produce plots of selected navigation and Kalman filter quantities including the Kalman filter state estimation errors and predicted rms error bounds.

The synthesis package was used for flight profile and aircraft/target geometry determination because it allowed many flight profiles to be quickly and conveniently generated. Simulations using this synthesized data also provided a "best case" performance baseline. The filter performance is not truly optimal with respect to the synthesized sensor data because the filter models include process and measurement noise sources not in the synthesized data (gravity anomalies, for example). However, the statistical behaviour of the errors which are included in the synthesized sensor data precisely matches the error models used in the Kalman filter. In this respect, the performance is optimal and represents a baseline for subsequent performance evaluations.

Error models in the synthesis package are limited to the major sensor error sources. More realistic synthesized sensor data provided by DREO was used for the Kalman filter performance evaluation. Real sensor data was not available for the final evaluation but SARMCS experience has shown that sensor data generated using the DREO synthesis package compares favourably with real data for testing purposes.

7.3 Flight Profile and Aircraft/Target Geometry

Recommendations regarding appropriate flight profiles and aircraft/target geometry for antenna boresight calibration have been presented based on computer simulations and operational considerations. The recommended flight profiles include manoeuvres designed to allow the Kalman filter to observe and estimate heading errors. In particular, it was found that while a single manoeuvre is sufficient for observation of the inertial platform azimuth misalignment, a second manoeuvre is required to ensure observation of the azimuth gyro bias. A particularly attractive combination of manoeuvres consists of two 180 degree turns in opposite directions separated by approximately 600 seconds of straight and level flight. This combination minimizes both manoeuvre time and cumulative errors and was found to work well in the simulations.

The described antenna boresight calibration method requires a radar target situated at a well surveyed, fixed location on the ground. Antenna azimuth misalignment estimation is most conveniently performed while flying directly toward the target. It was found that a calibration time of 300 to 500 seconds is adequate given azimuth matching measurements at 10 second intervals (30 to 50 measurements). Furthermore, it was found that calibration should be performed at maximum TAU range from the target, where bearing error is minimum, and immediately following the manoeuvres, where platform azimuth misalignment is minimum.

Limited error models in the data synthesis package result in near optimal Kalman filter performance when processing this data. The antenna azimuth calibration accuracy was found to be about 2 arc minutes rms using the recommended flight profile and aircraft/target geometry. This assumes 30 antenna azimuth matching measurements at 10 second intervals and a measurement noise of 2 milliradians rms. The calibration performance was found to be

insensitive to changes in the flight profile heading, ground speed, altitude and latitude, within reasonable operating limits. These results form the "best case" performance baseline for subsequent performance evaluations.

7.4 Antenna Boresight Calibration Performance

Evaluation of the Kalman filter calibration performance is based on simulations using synthesized sensor data provided by DREO. The calibration Kalman filter is suboptimal since it is based on the suboptimal SARMCS filter. However, extensive flight trials have validated the SARMCS design and provide grounds for confidence in the calibration Kalman filter performance evaluation results. In the simulations, errors in the Kalman filter estimated states were consistent with predicted error bounds suggesting that the design is reasonable.

The evaluation flight profile includes two 180 degree turns in opposite directions separated by 600 seconds and followed by 300 seconds of flight directly toward a target from an initial range of 50 kilometres. Antenna azimuth misalignment estimation was performed during this last 300 seconds with TAU measurements at 10 second intervals. The total of manoeuvre time plus calibration time is less than 20 minutes.

Calibration performance was found to be poorer than the previously achieved 2 arc minutes rms. Major factors contributing to the poorer performance were found to be increased process noise spectral densities due to turbulence and vibration, increased position matching measurement noise due to INS output quantization, and decreased target range. Increased process noise alone brought the filter predicted antenna azimuth misalignment error up to 2.6 arc minutes rms and the addition of INS quantization noise further increased the predicted error to 2.9 arc minutes

rms. Decreasing the initial target range from 100 kilometres to 50 kilometres resulted in another jump in the predicted antenna azimuth misalignment error up to 3.3 arc minutes rms.

For implementation of the calibration Kalman filter in the current SARMCS, it is necessary to use a ten second Kalman filter update interval. When simulations were performed using ten second update intervals (previous results were based on one second updates), the filter predicted antenna azimuth misalignment error increased from 3.3 to 3.8 arc minutes rms. The SARMCS Kalman filter actually obtains the transition matrix $\mathbf{\Phi}_j$ at 2 second subintervals and forms $\mathbf{\Phi}_k$ for the entire interval using the property $\mathbf{\Phi}_k = \mathbf{\Phi}_{j+4}\mathbf{\Phi}_{j+3}\mathbf{\Phi}_{j+2}\mathbf{\Phi}_{j+1}\mathbf{\Phi}_j$. This may slightly improve the results obtained with the longer filter update interval.

DREO is also interested in replacing the INS and GPS sensors used in the SARMCS with an integrated INS/GPS system. A cursory examination of the problems associated with a separated INS/GPS Kalman filter and simulations using a somewhat ad hoc calibration Kalman filter design suggest that a calibration accuracy of better than 5 arc minutes rms is still feasible.

In summary, extensive computer simulations have demonstrated the feasibility of the described antenna boresight calibration approach for use in the Canadian Department of National Defence Spotlight SAR program. Required calibration accuracy is 5 arc minutes rms or better. The performance evaluation results suggest that the Spotlight SAR antenna azimuth misalignment can be estimated to an accuracy of better than 4 arc minutes rms within a 20 minute period. It remains, however, to verify these results using real sensor data obtained during actual flight trials.

REFERENCES

Benson, D. O., Jr. (1975): "A Comparison of Two Approaches to Pure-Inertial and Doppler-Inertial Error Analysis," IEEE Trans. Aerospace and Electronic Systems, Vol. AES-11, No. 4, July, pp. 447-455.

Britting, K. R. (1971): *Inertial Navigation Systems Analysis*, Wiley-Interscience, New York.

Brown, R. G. and P. Y. C. Hwang (1992): *Introduction to Random Signals and Applied Kalman Filtering*, 2nd ed., Wiley, New York.

Carlson, N. A. (1988): "Federated Filter for Fault-Tolerant Integrated Navigation Systems," Proceedings of IEEE Position, Location and Navigation Symposium, PLANS-88, pp. 110-119.

Defense Mapping Agency (1987): "Department of Defense World Geodetic System 1984—Its Definition and Relationships with Local Geodetic Systems," DMA Technical Report 8350.2, Defense Mapping Agency, Washington, D.C.

DiFilippo, D. J. (July 10, 1992): Letter to the author including unpublished notes on inertial navigation error analysis.

DiFilippo, D. J. (Dec. 23, 1992): "Design Notes on SARMCS Kalman Filter," unpublished.

DiFilippo, D. J., G. E. Haslam and W. S. Widnall (1988): "Evaluation of a Kalman Filter for SAR Motion Compensation," Proceedings of IEEE Position, Location and Navigation Symposium, PLANS-88, pp. 259-268.

Dragomir, V. C., D. N. Ghiţău, M. S. Mihăilescu and M. G. Rotaru (1982): *Theory of the Earth's Shape*, Elsevier Scientific Publishing Co., New York.

Ewing, C. E. and M. M. Mitchell (1970): *Introduction to Geodesy*, American Elsevier Publishing Co., New York.

Farrell, J. L. (1976): *Integrated Aircraft Navigation*, Academic Press, New York.

Gelb, A. ed. (1974): *Applied Optimal Estimation*, MIT Press, Cambridge, Mass.

Haslam, G. E., M. R. Vant and D. DiFilippo (1988): "A Synthetic Aperture Radar Mode for the AN/APS-506 Maritime Search Radar," IEEE National Radar Conference, pp. 106-110.

Hepburn, J. S. A., D. B. Reid, W. S. Widnall, D. F. Liang and G. E. Haslam (1984): "Motion Compensation for High Resolution Spotlight SAR," Proceedings of IEEE Position, Location and Navigation Symposium, PLANS-84, pp. 59-65.

Heiskanen, W. A. and H. Moritz (1967): *Physical Geodesy*, W. H. Freeman and Co., San Francisco, Calif.

Huddle, J. R. (1983): "Inertial Navigation System Error-Model Considerations in Kalman Filtering Applications," in *Control and Dynamic Systems*, Vol. 20 (C. T. Leondes, ed.), Academic Press, New York.

MATLAB, The Mathworks, Inc., Cochituate Place, 24 Prime Parkway, Natick, MA 01760.

Milliken, R. J. and C. J. Zoller (1978): "Principle of Operation of NAVSTAR and System Characteristics," Navigation: Journal of the Institute of Navigation, Vol. 25, No. 2, pp. 95-106.

Von Mises, R. (1959): *Theory of Flight*, Dover Publications, New York.

Poor, W. A. (1989): "A Geometric Description of Wander Azimuth Frames," Navigation: Journal of the Institute of Navigation, Vol. 36, No. 3, pp. 303-318.

Press, W. H., B. P. Flannery, S. A. Teukolsky and W. T. Vetterling (1986): *Numerical Recipes: The Art of Scientific Computing,* Cambridge University Press, New York.

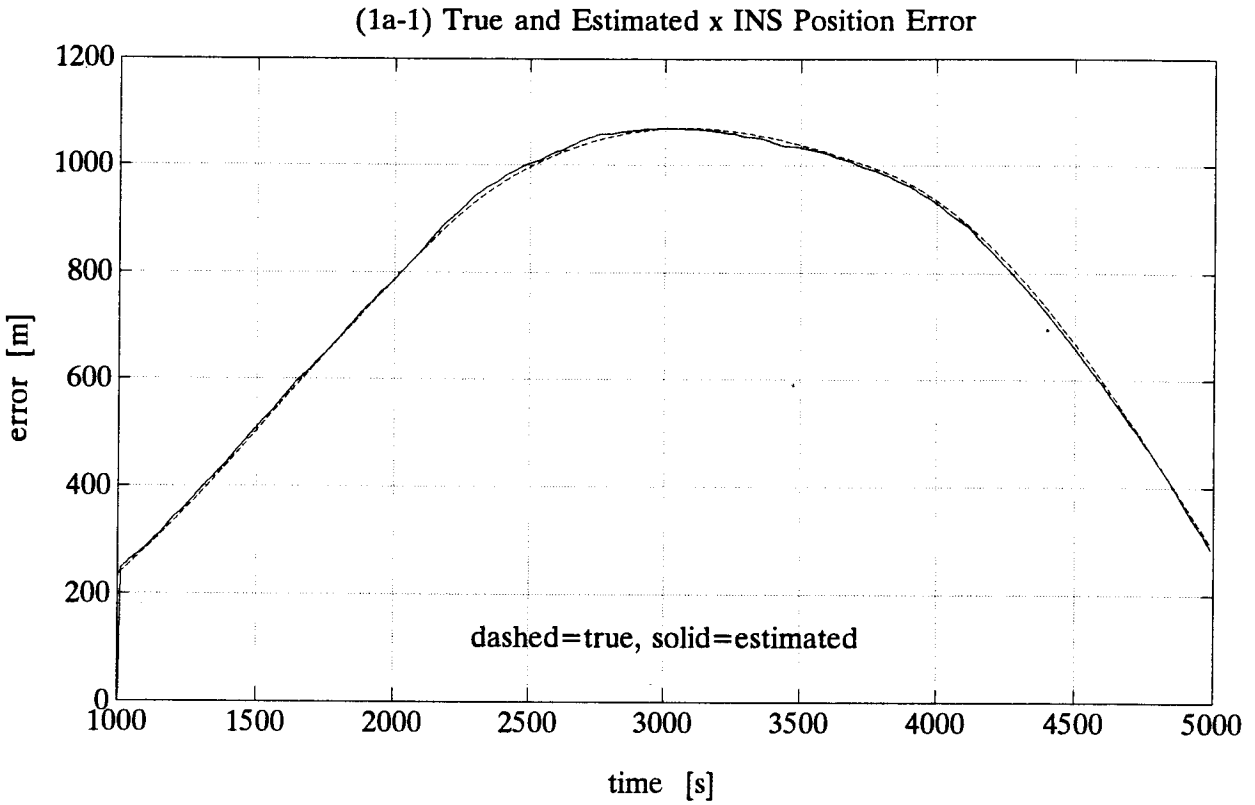
Schlee, F. H., N. F. Toda, M. A. Islam and C. J. Standish (1988): "Use of an External Cascaded Kalman Filter to Improve the Performance of a Global Positioning System (GPS) Inertial Navigator," Proceedings of National Aerospace and Electronics Conference, NAECON, pp. 345-350.

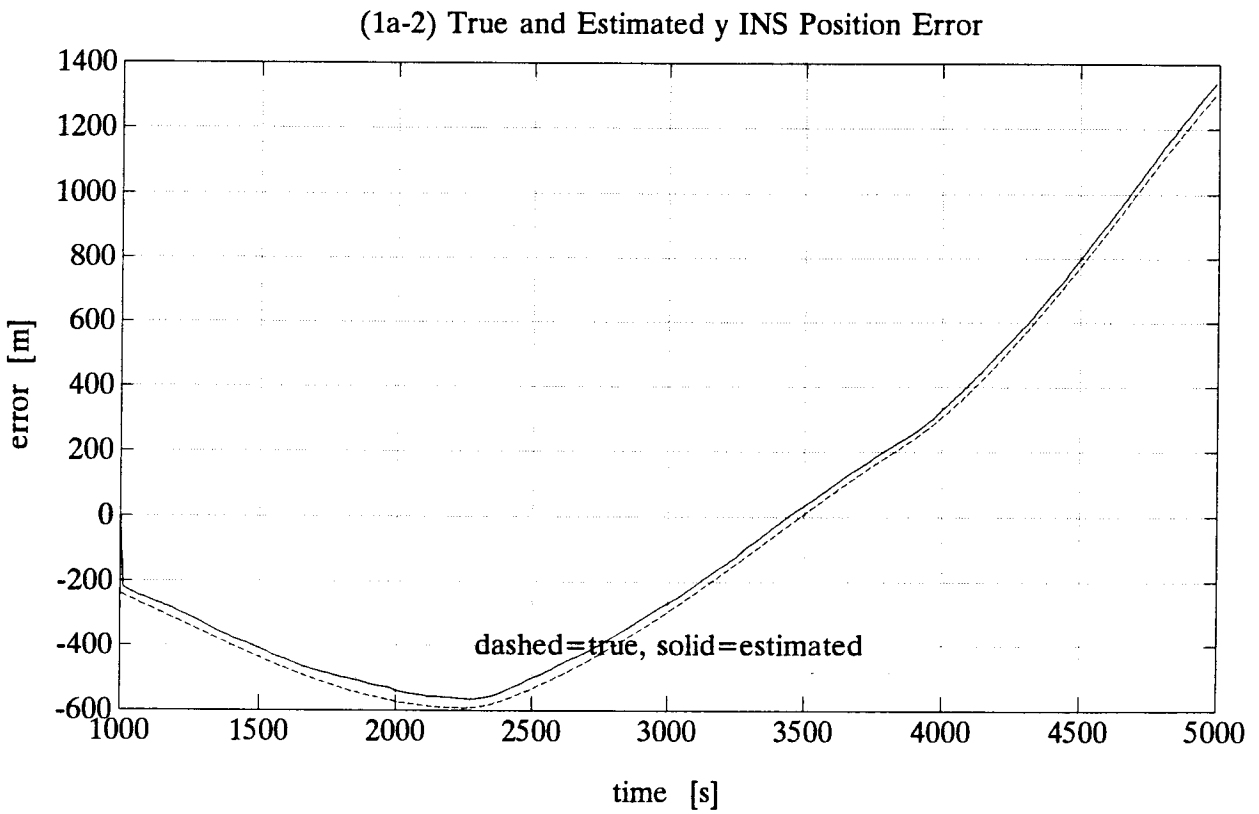
Zachmann, G. W. (1988): "GPS Accuracy for Civil Marine Navigation," presented at National Marine Electronics Assoc., Boston, Mass.

APPENDIX A

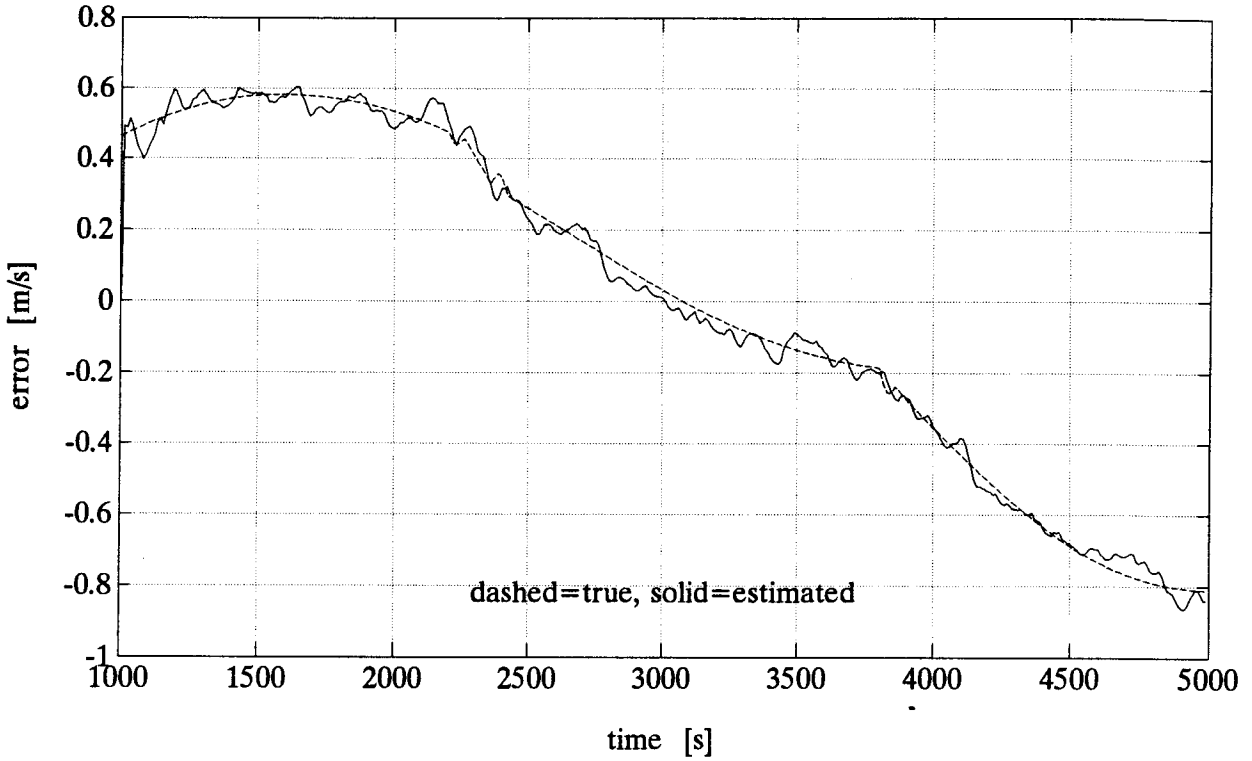
FLIGHT PROFILE SIMULATION PLOTS

This appendix contains plots from the flight profile and aircraft/target geometry simulation runs (Section 5, simulation runs 1a, 2a and 3a). The plots were produced using the computer simulation evaluation package described in Section 4.

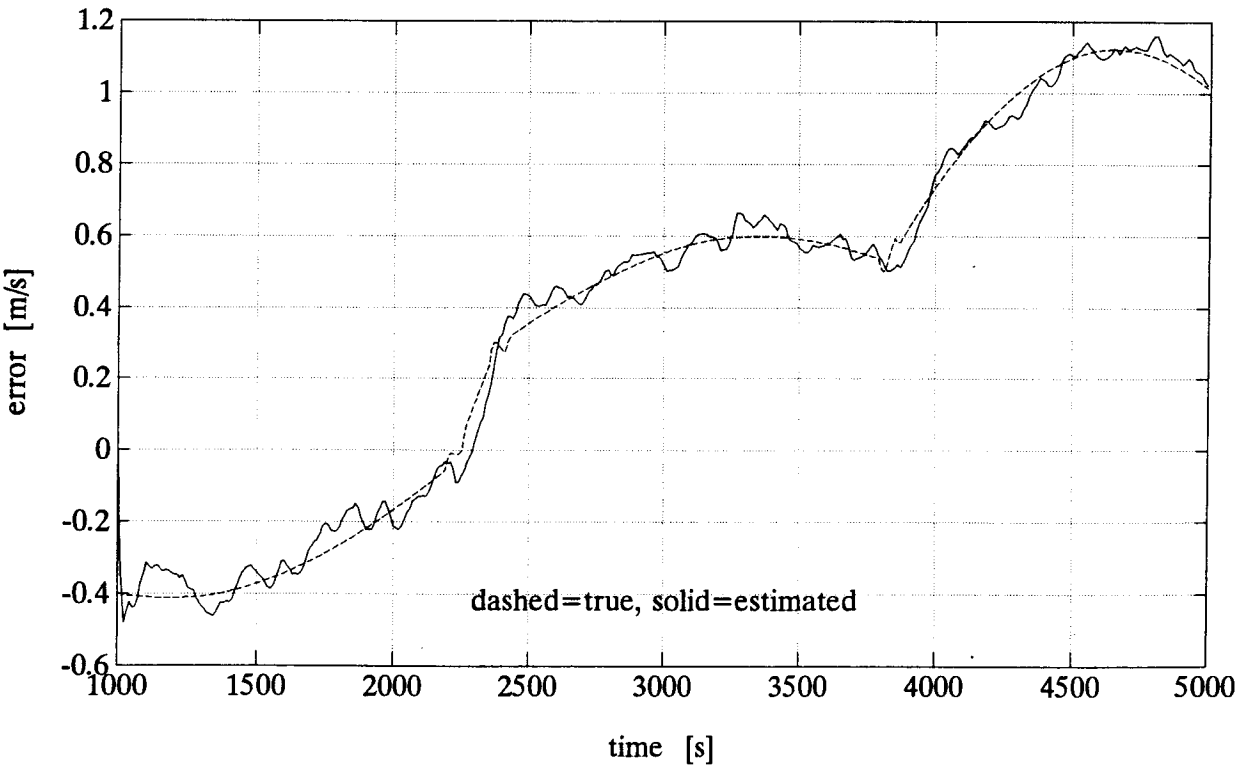

(1a-1) True and Estimated x INS Position Error
1200
1000
800
600
400
200
0
error [m]
dashed=true, solid=estimated
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


(1a-2) True and Estimated y INS Position Error
1400
1200
1000
800
600
400
200
0
-200
-400
-600
error [m]
dashed=true, solid=estimated
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

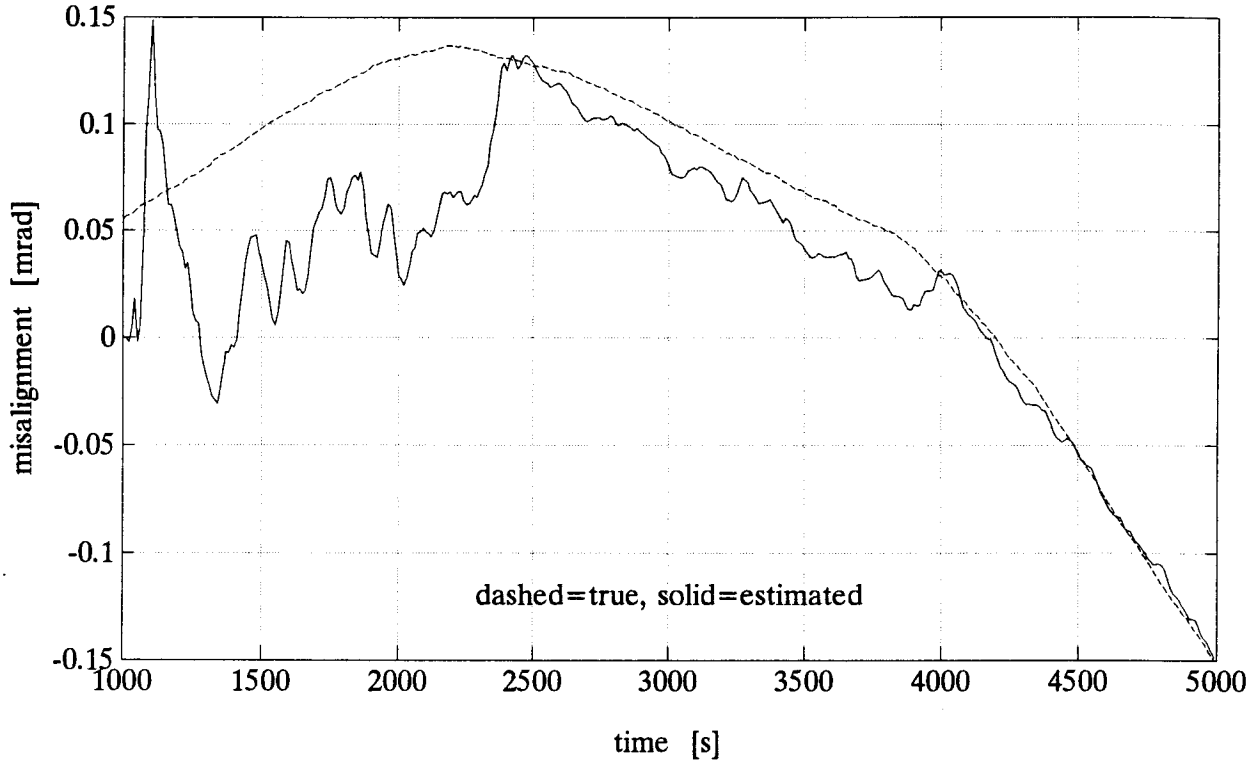
(1a-3) True and Estimated x INS Velocity Error



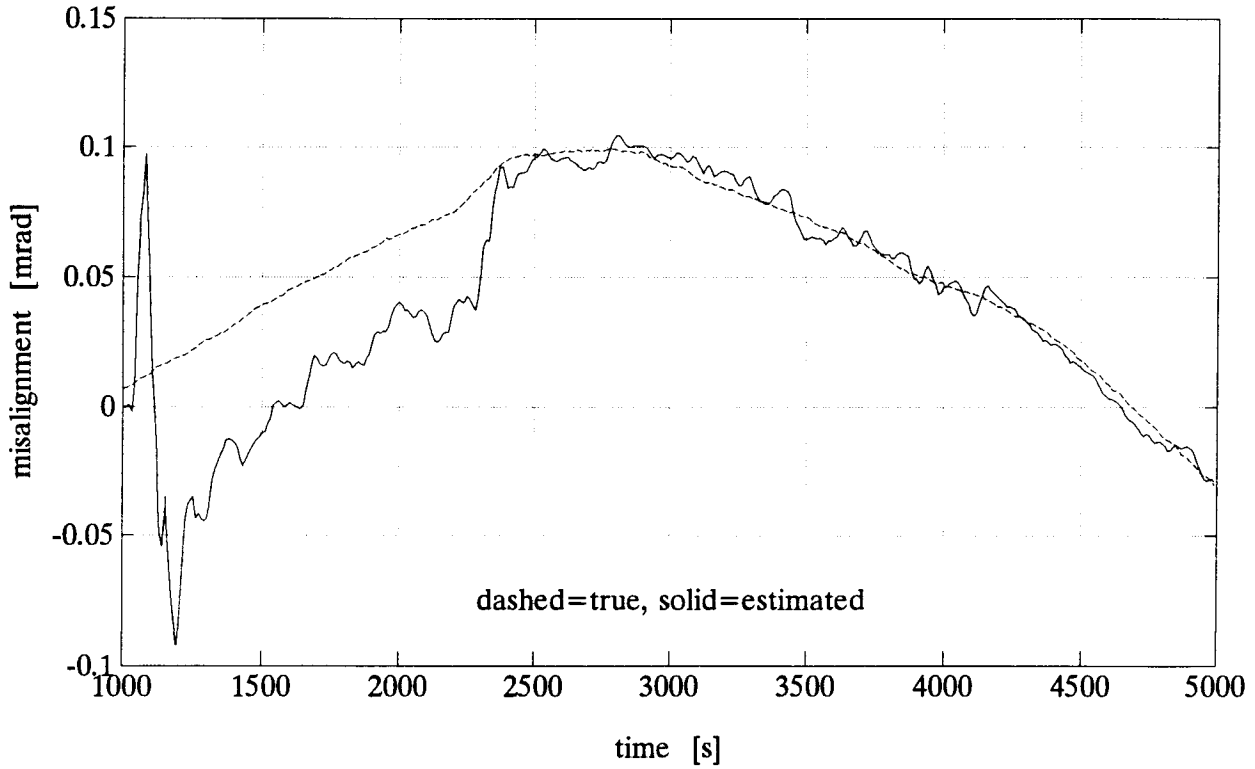
(1a-4) True and Estimated y INS Velocity Error



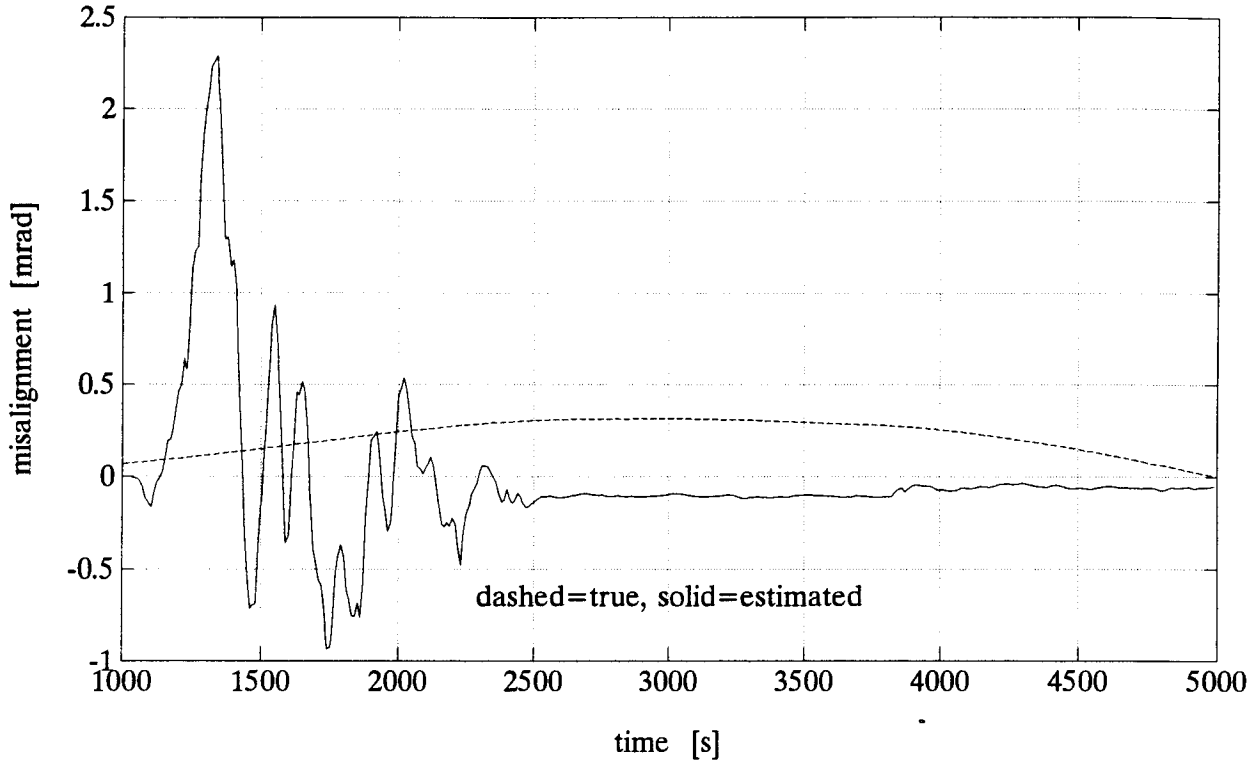
(1a-5) True and Estimated x INS Platform Misalignment



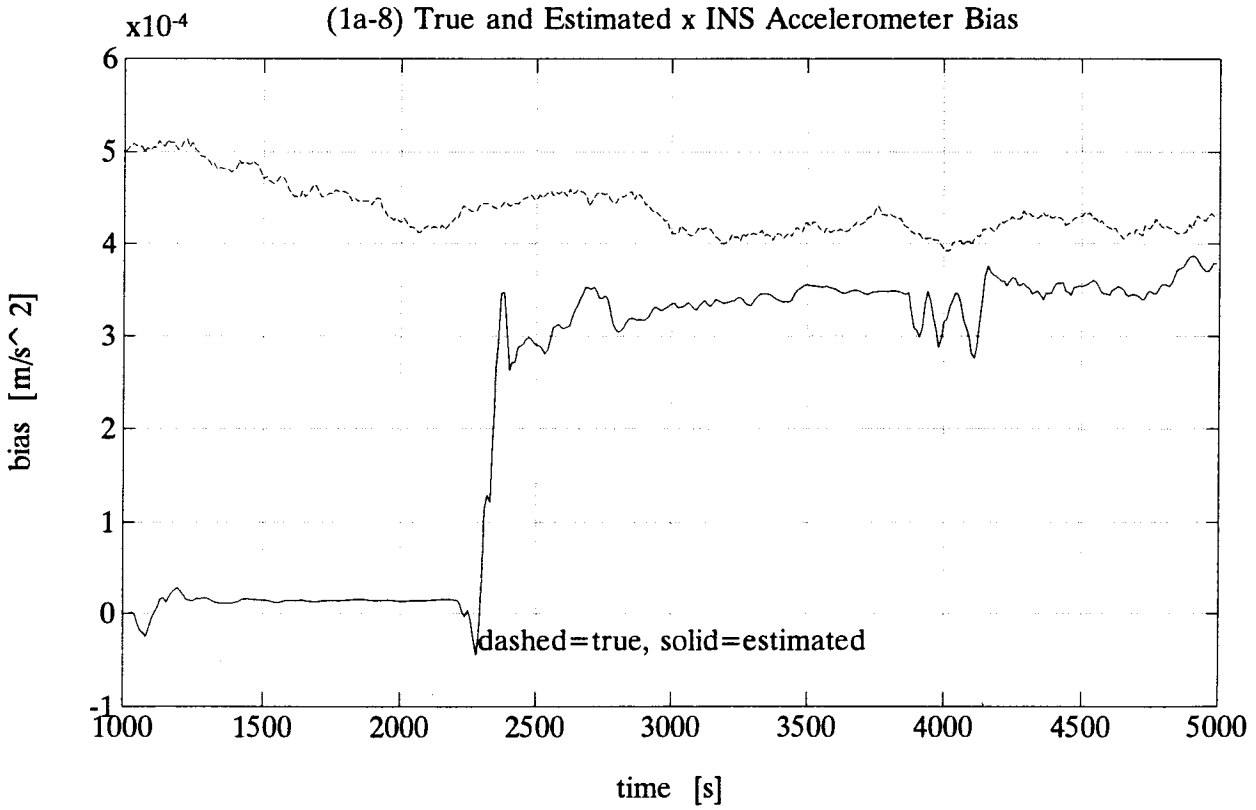
(1a-6) True and Estimated y INS Platform Misalignment

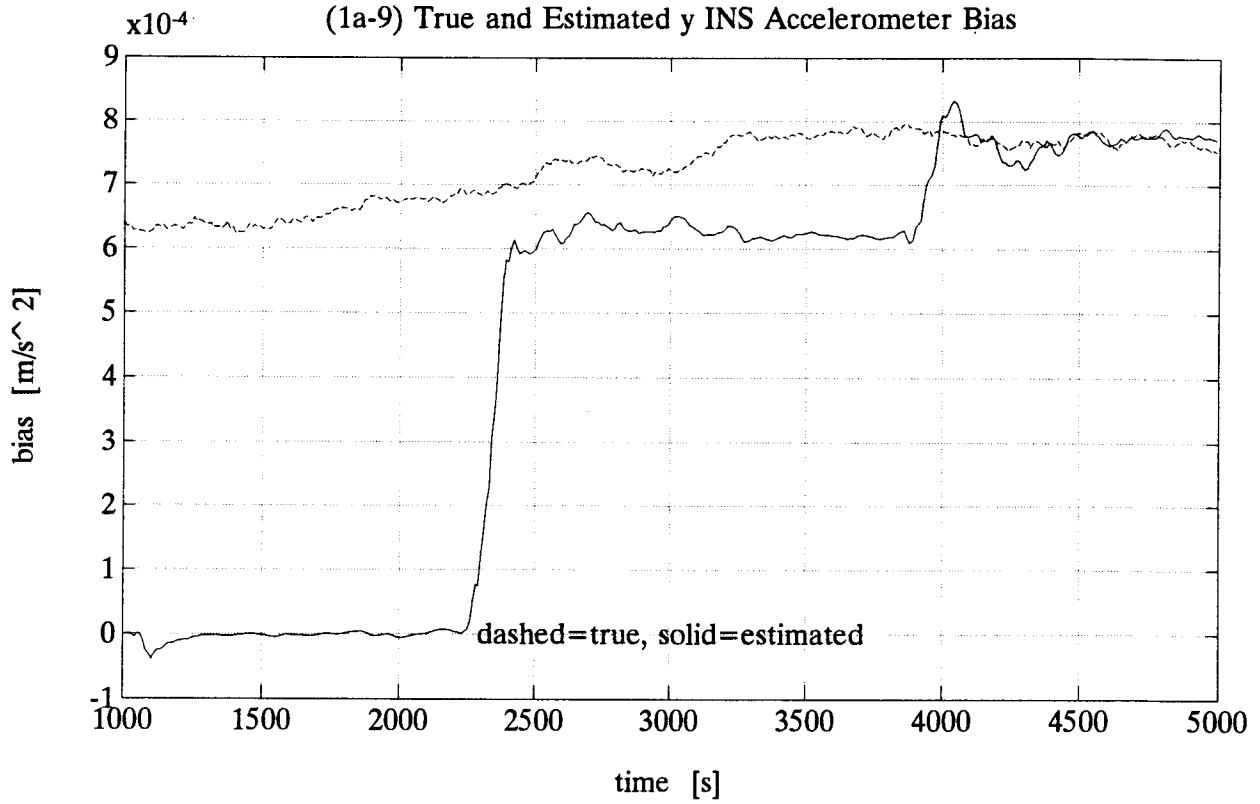


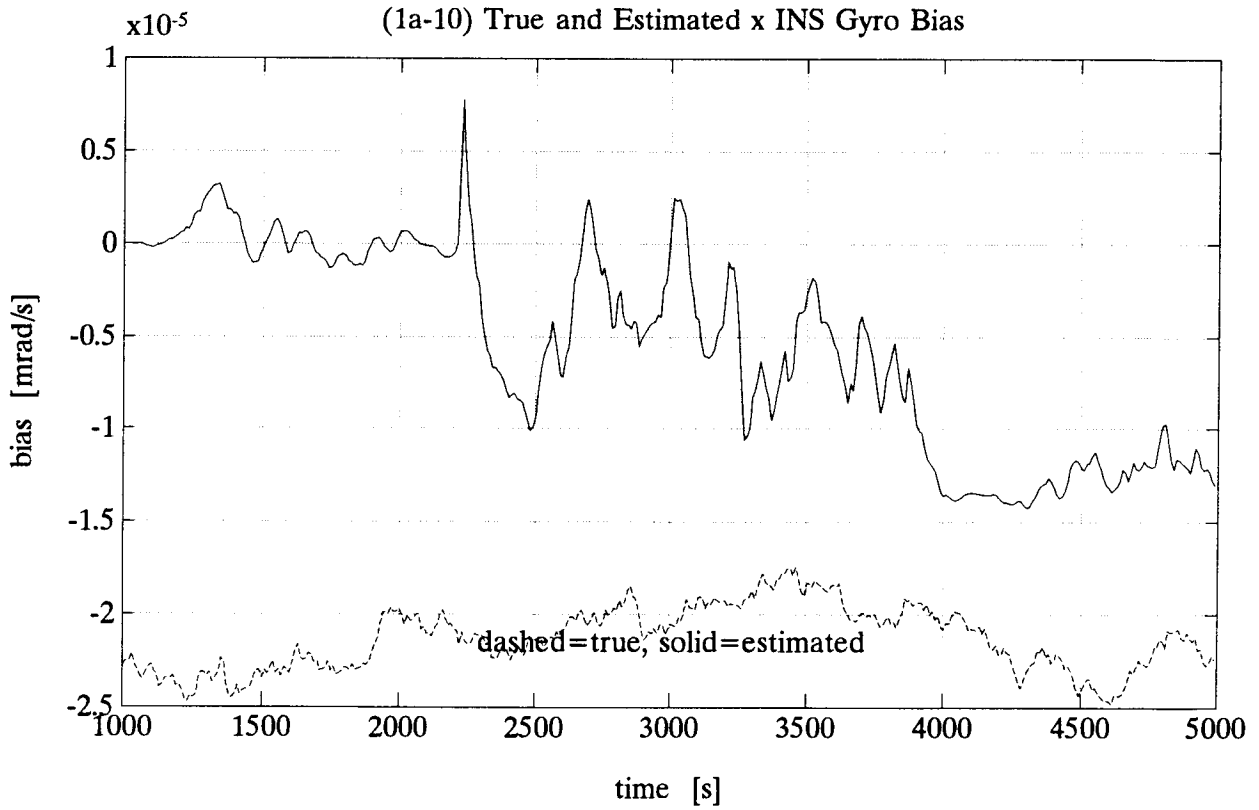
(1a-7) True and Estimated z INS Platform Misalignment

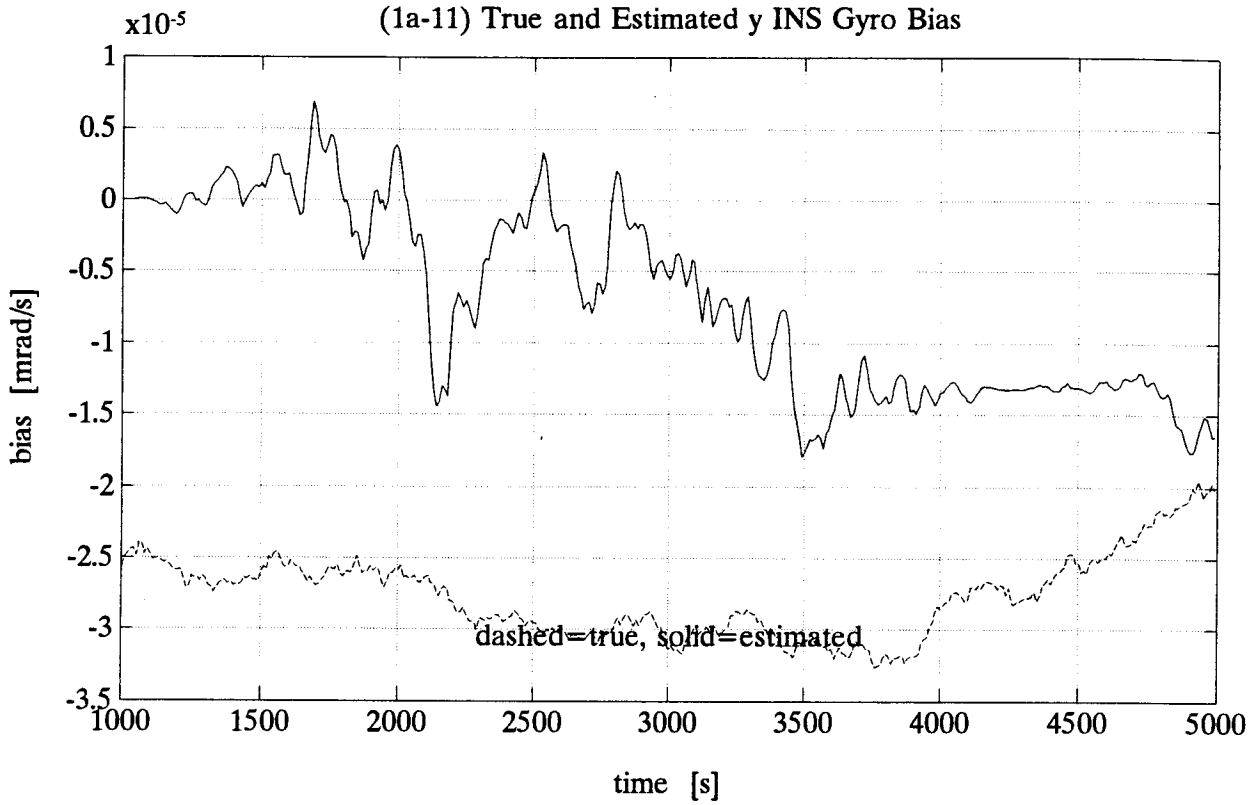


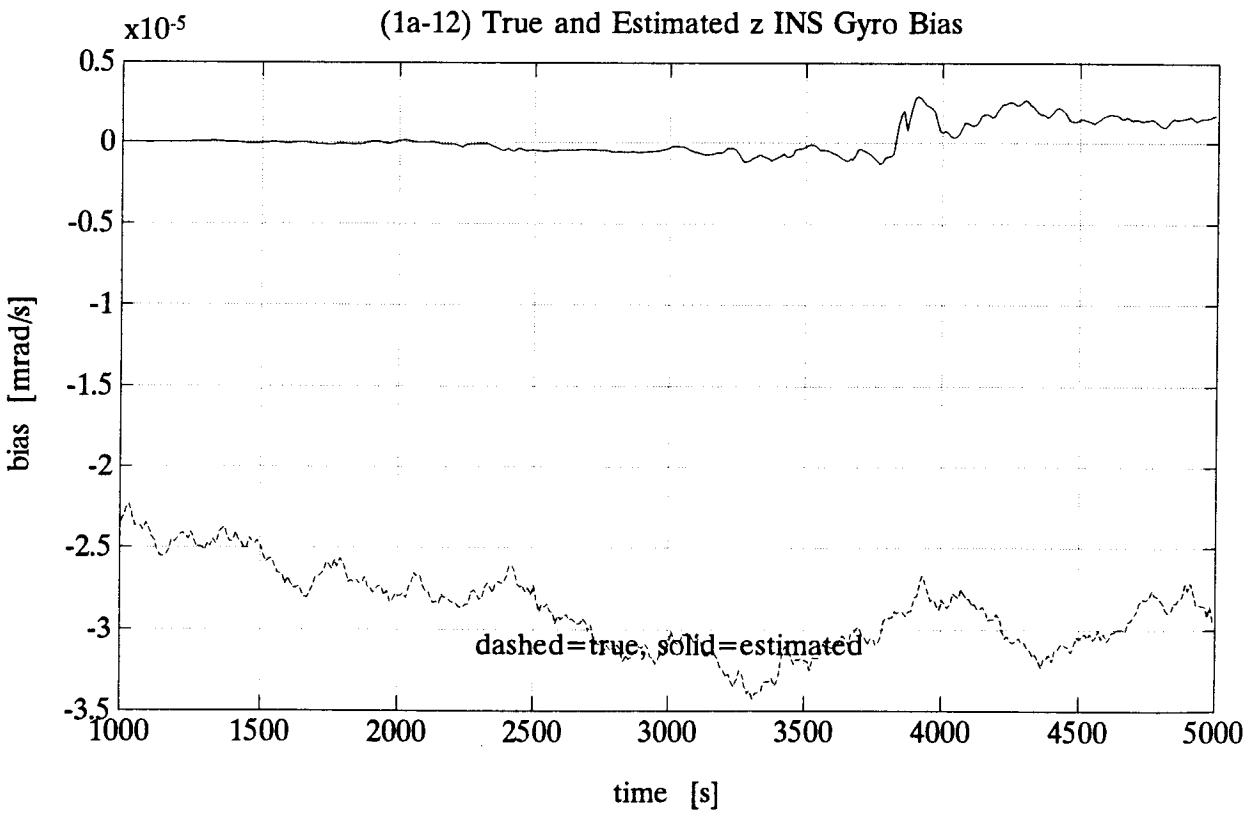
(1a-8) True and Estimated x INS Accelerometer Bias



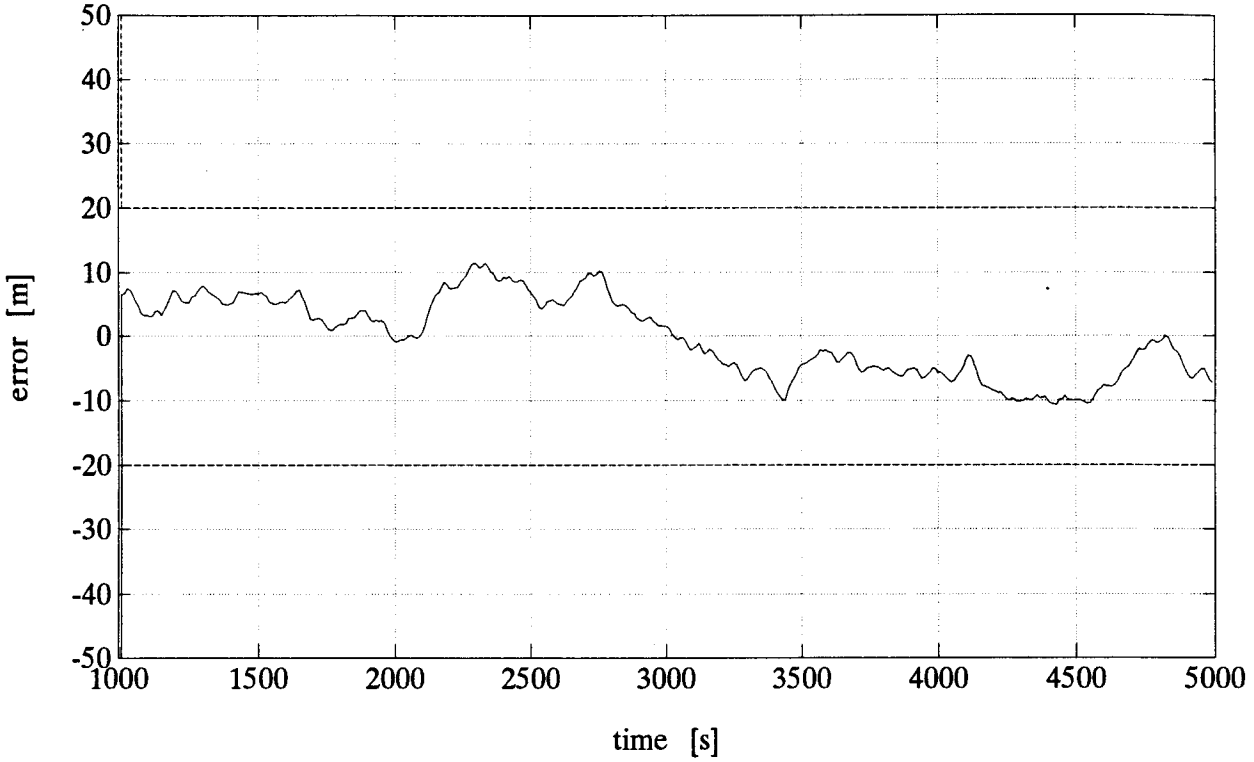

x10-4
(1a-9) True and Estimated y INS Accelerometer Bias
bias [m/s^ 2]
dashed=true, solid=estimated
time [s]


x10-5
(1a-10) True and Estimated x INS Gyro Bias
bias [mrad/s]
dashed=true, solid=estimated
time [s]

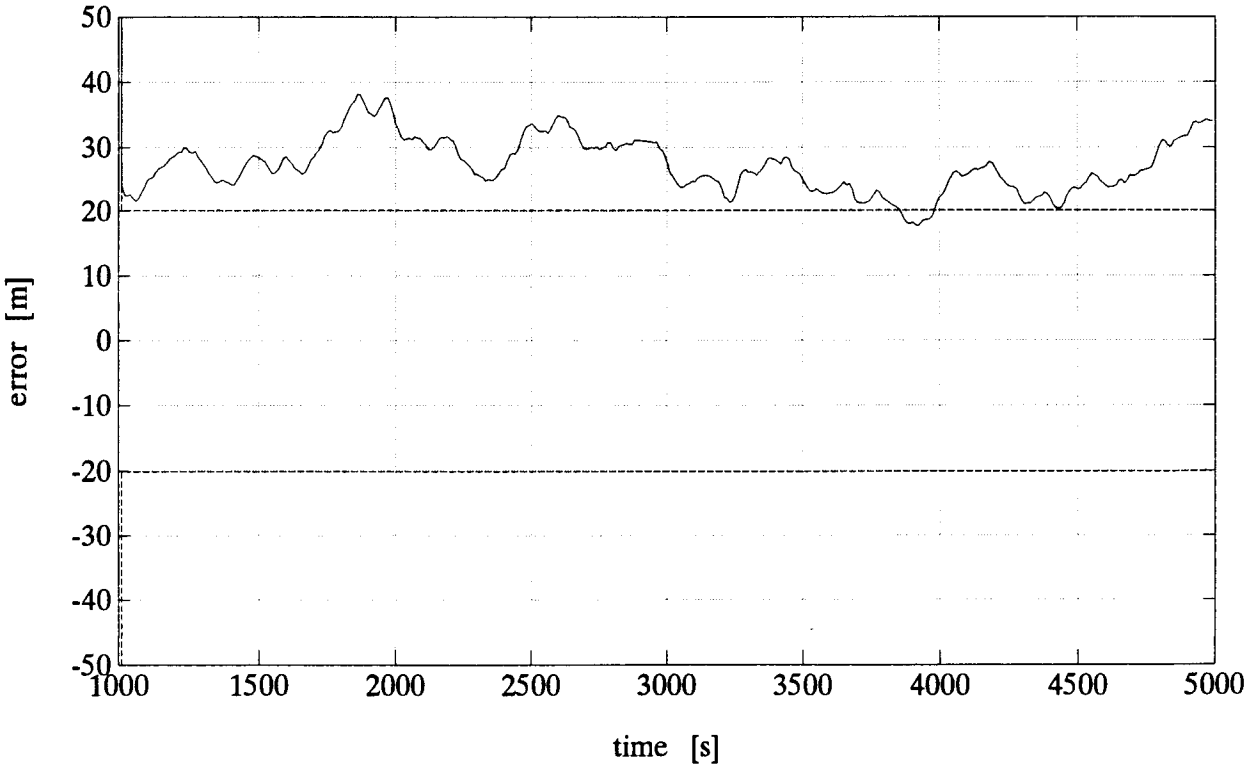

x10-5
(1a-11) True and Estimated y INS Gyro Bias
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
bias [mrad/s]
dashed=true, solid=estimated
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


x10-5
(1a-12) True and Estimated z INS Gyro Bias
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
bias [mrad/s]
dashed=true, solid=estimated
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

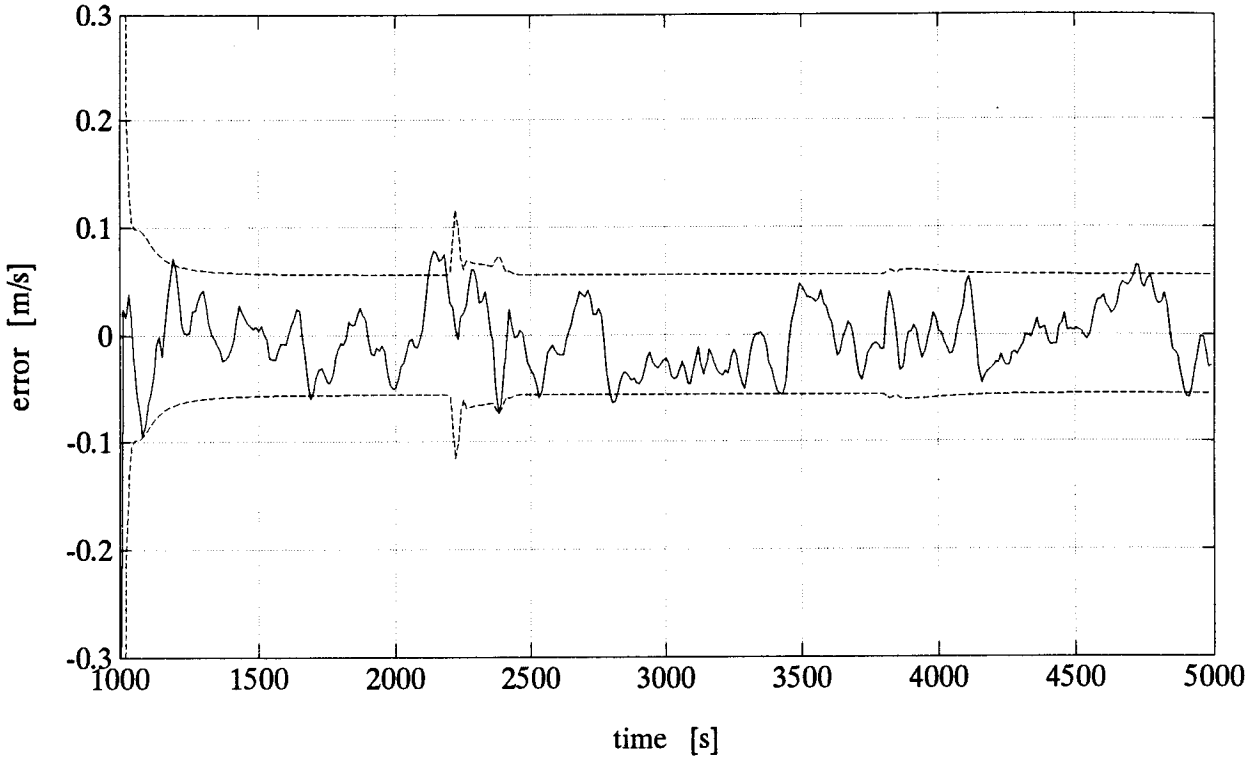
(1a-13) Error in x INS Position Error Estimate



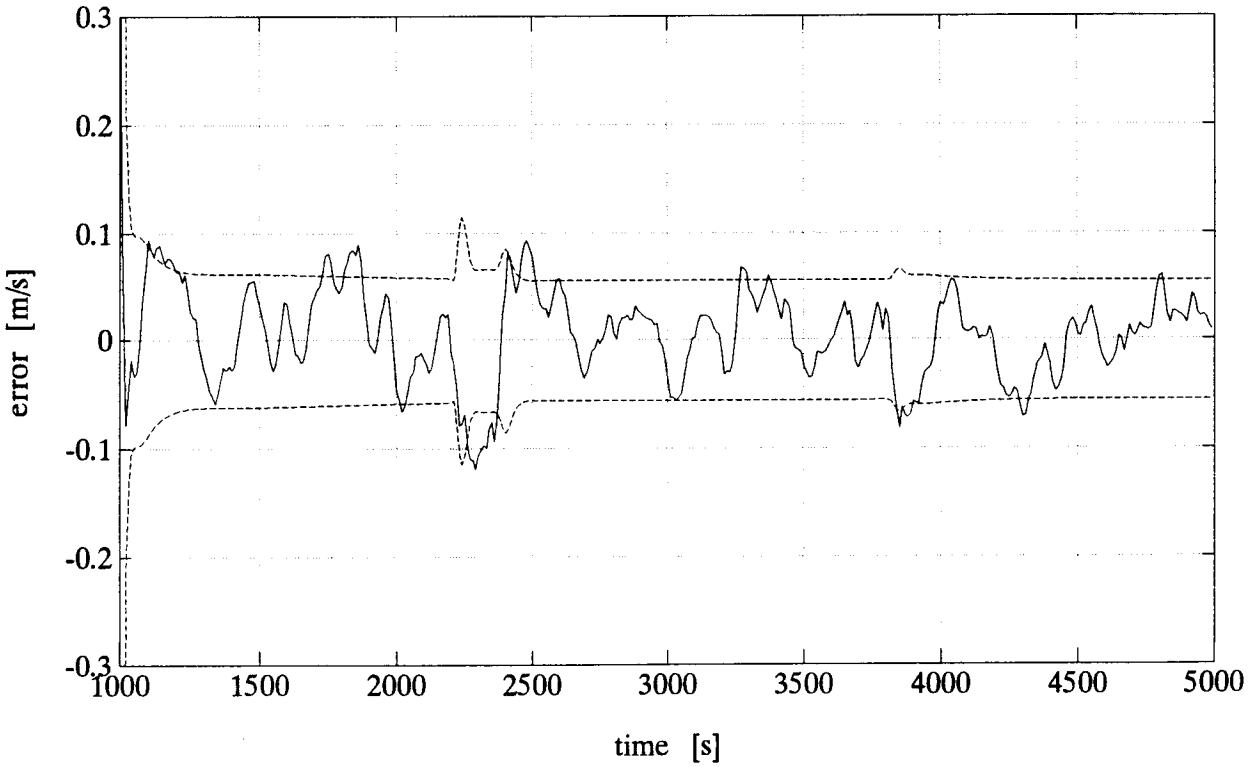
(1a-14) Error in y INS Position Error Estimate



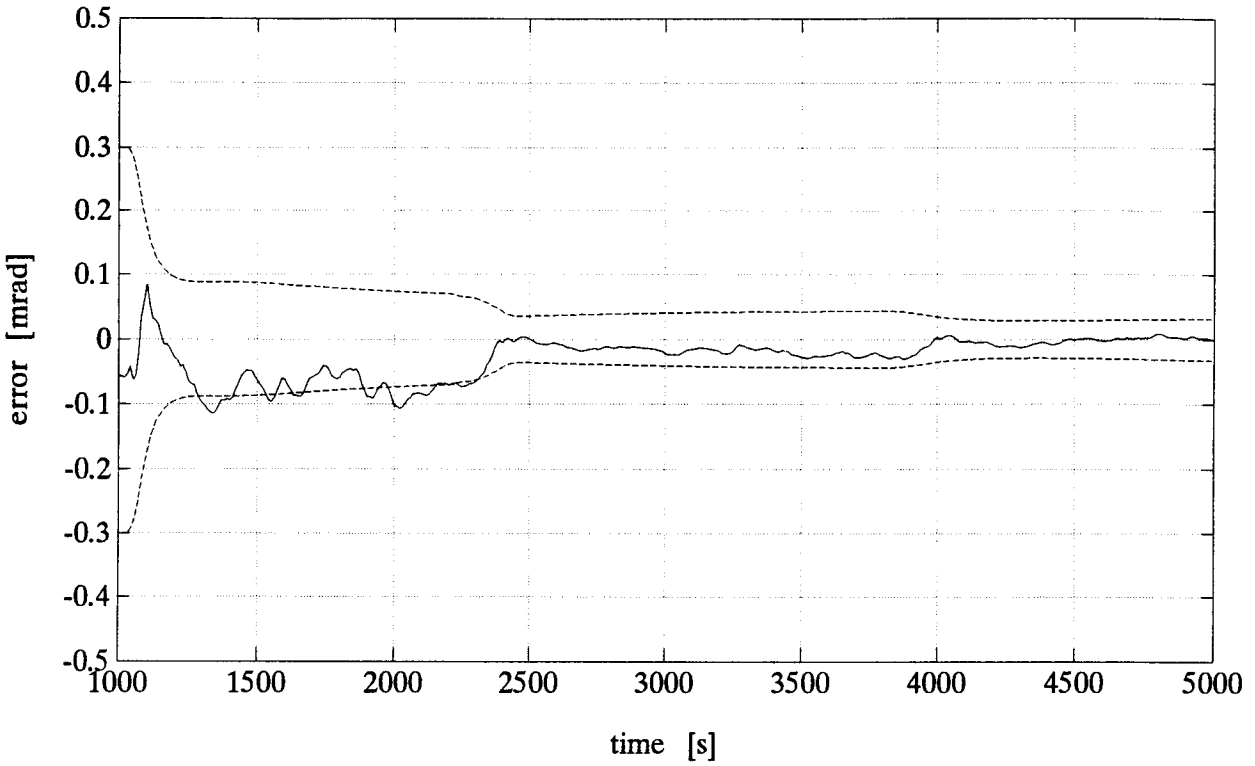
(1a-15) Error in x INS Velocity Error Estimate



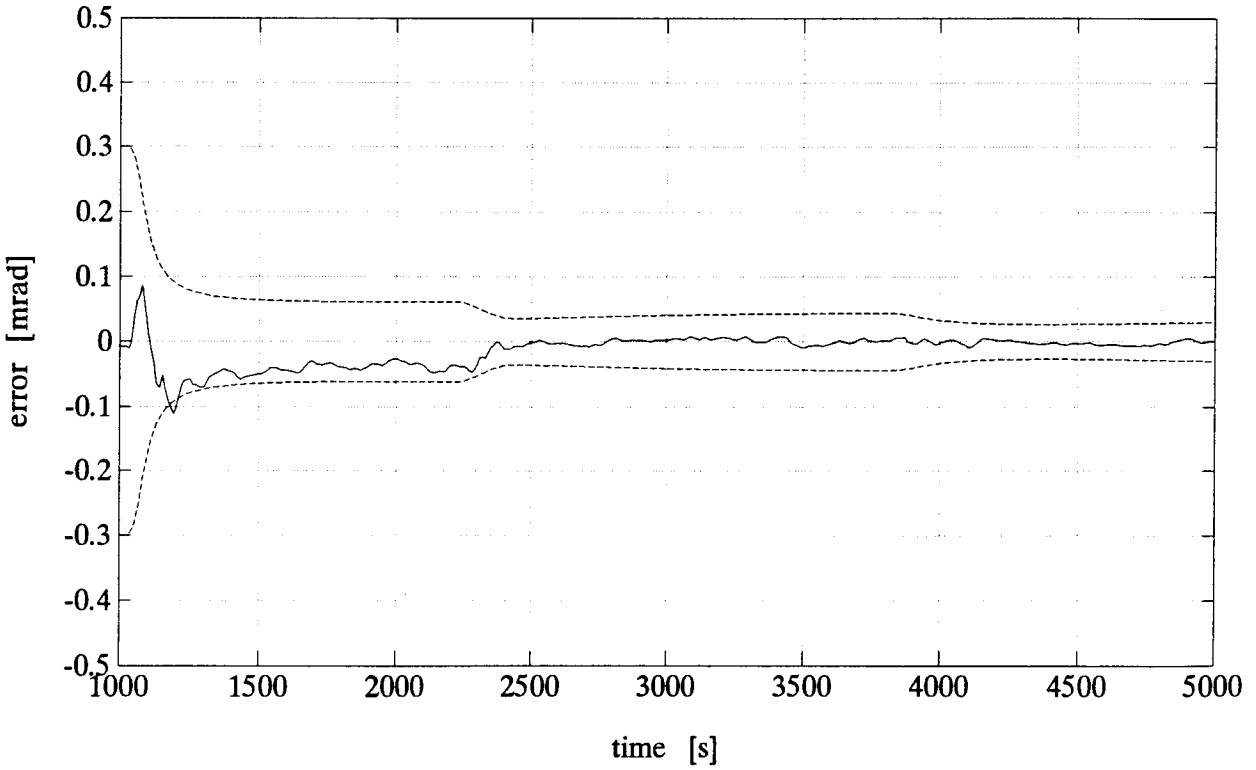
(1a-16) Error in y INS Velocity Error Estimate



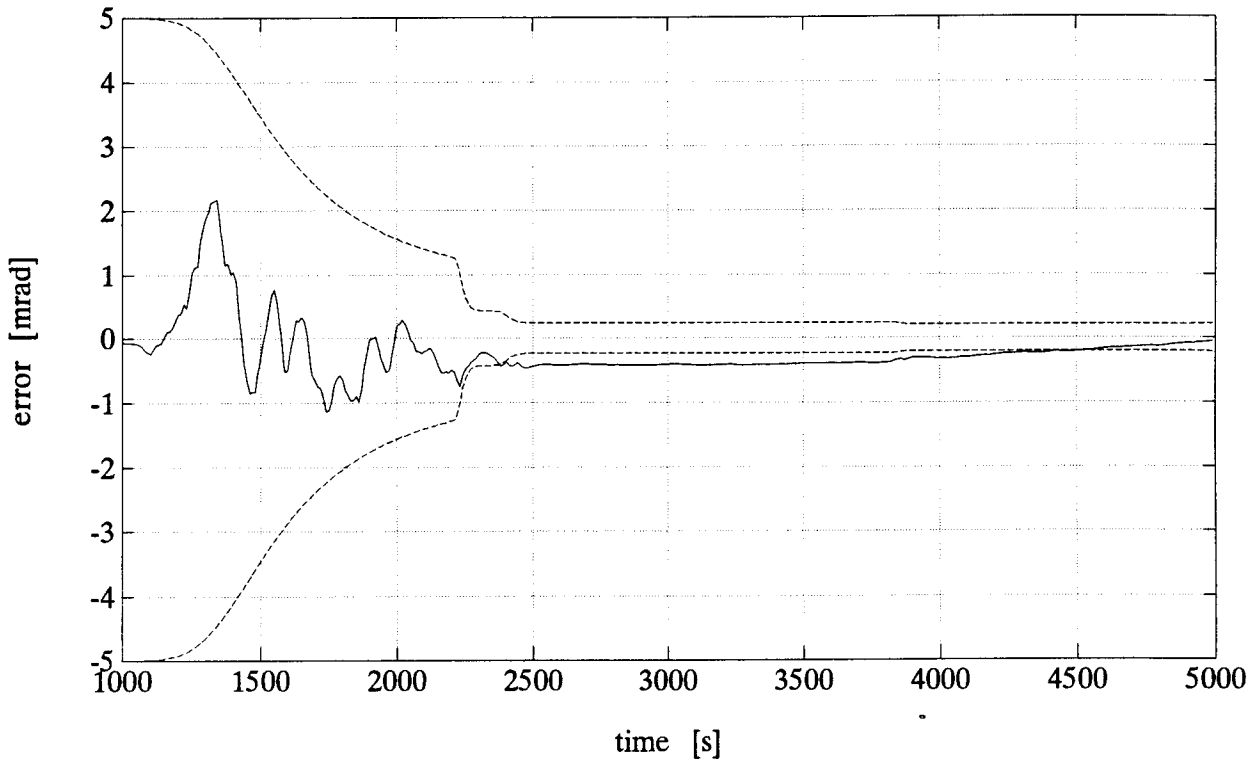
(1a-17) Error in x INS Platform Misalignment Estimate



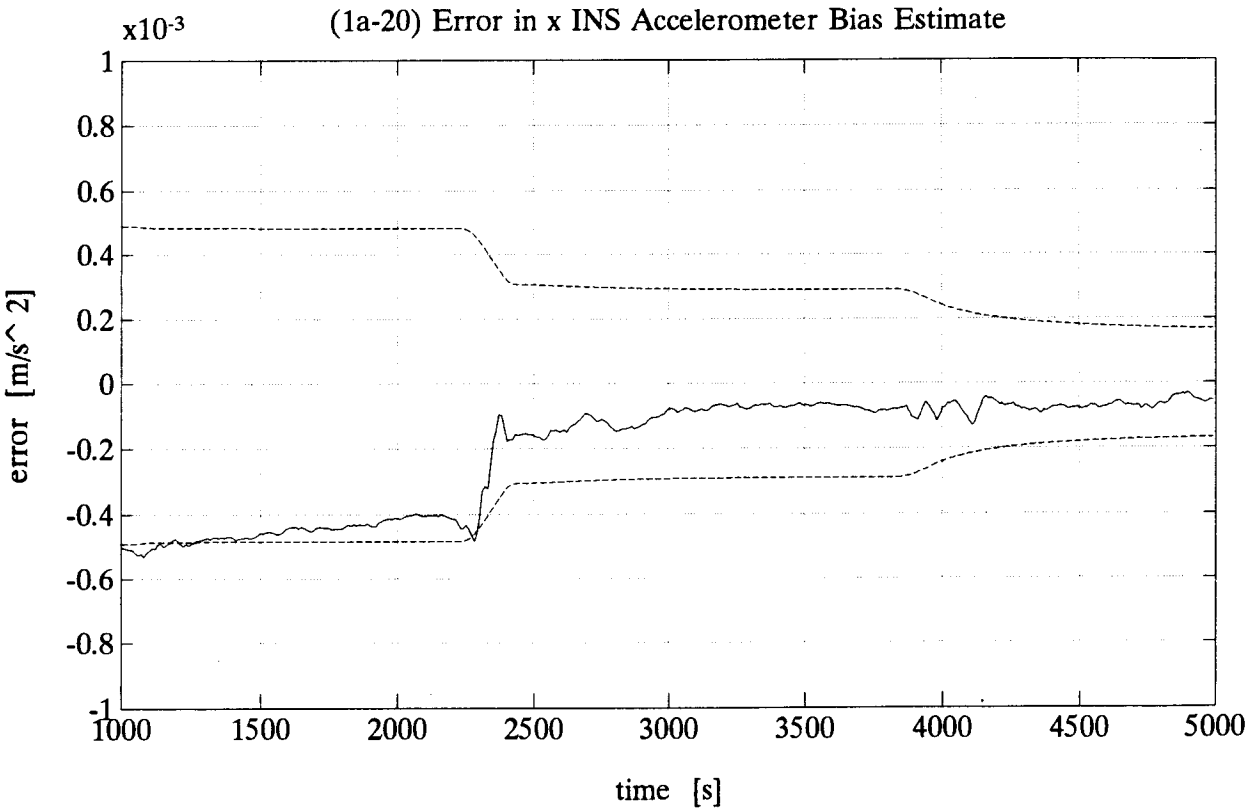
(1a-18) Error in y INS Platform Misalignment Estimate

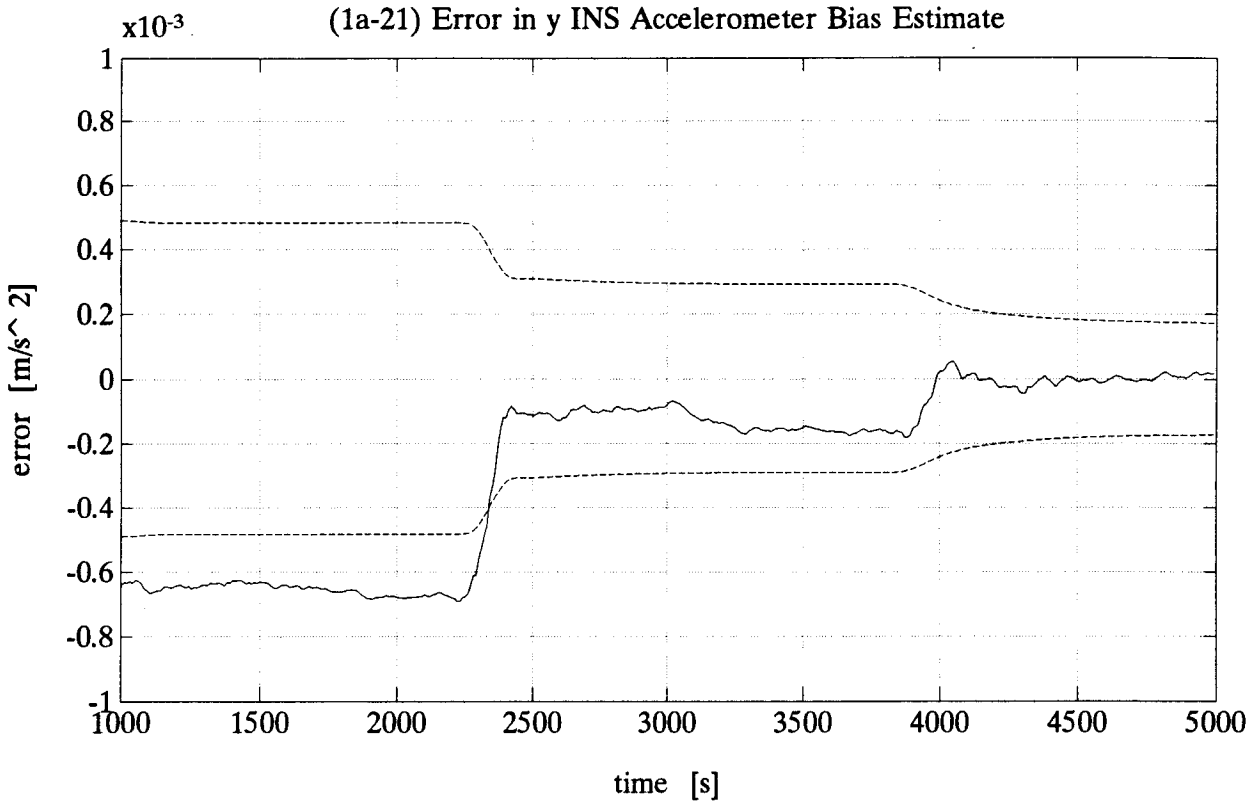


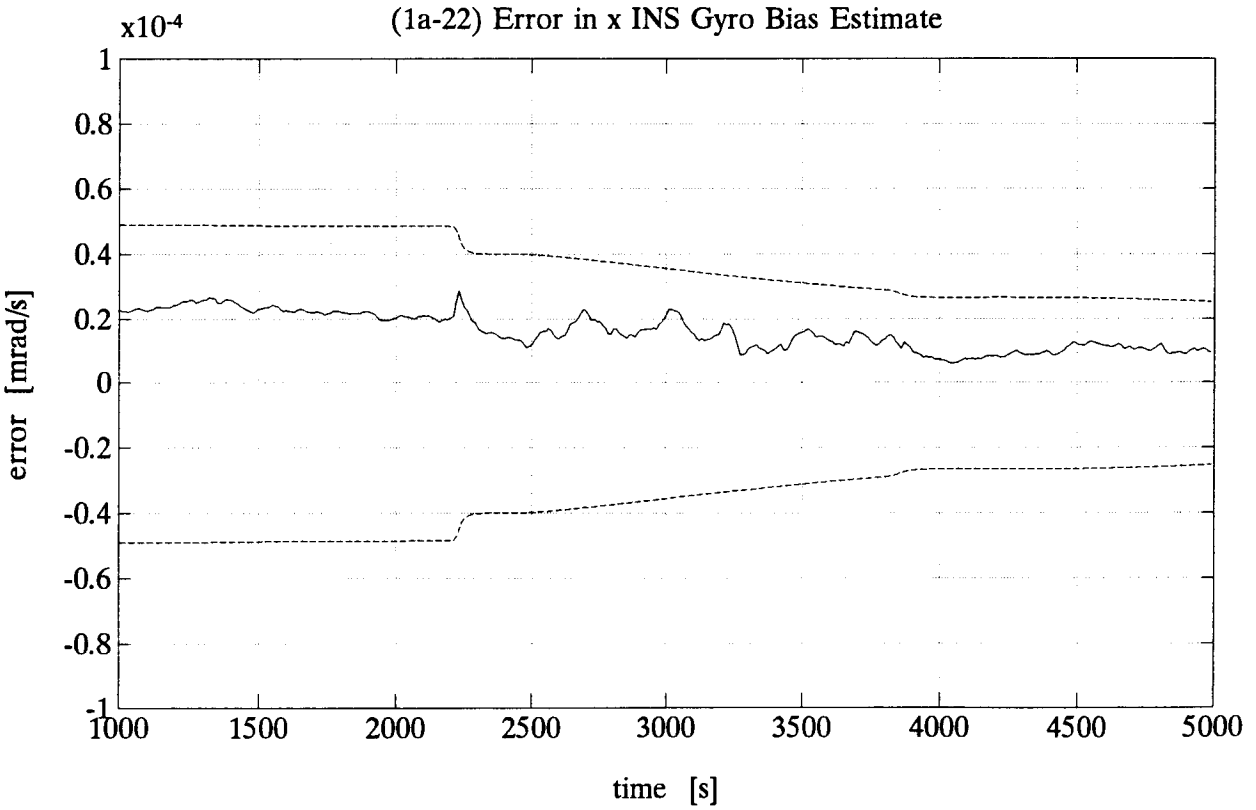
(1a-19) Error in z INS Platform Misalignment Estimate

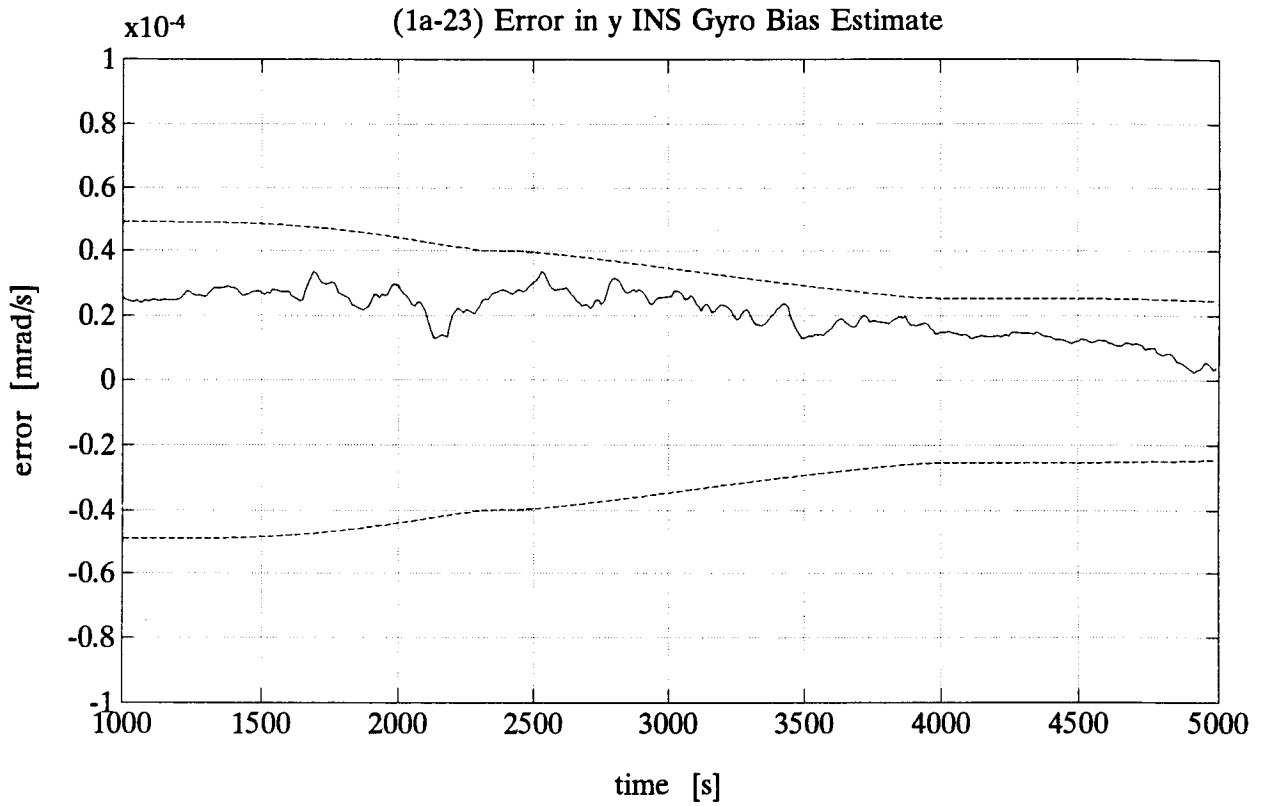


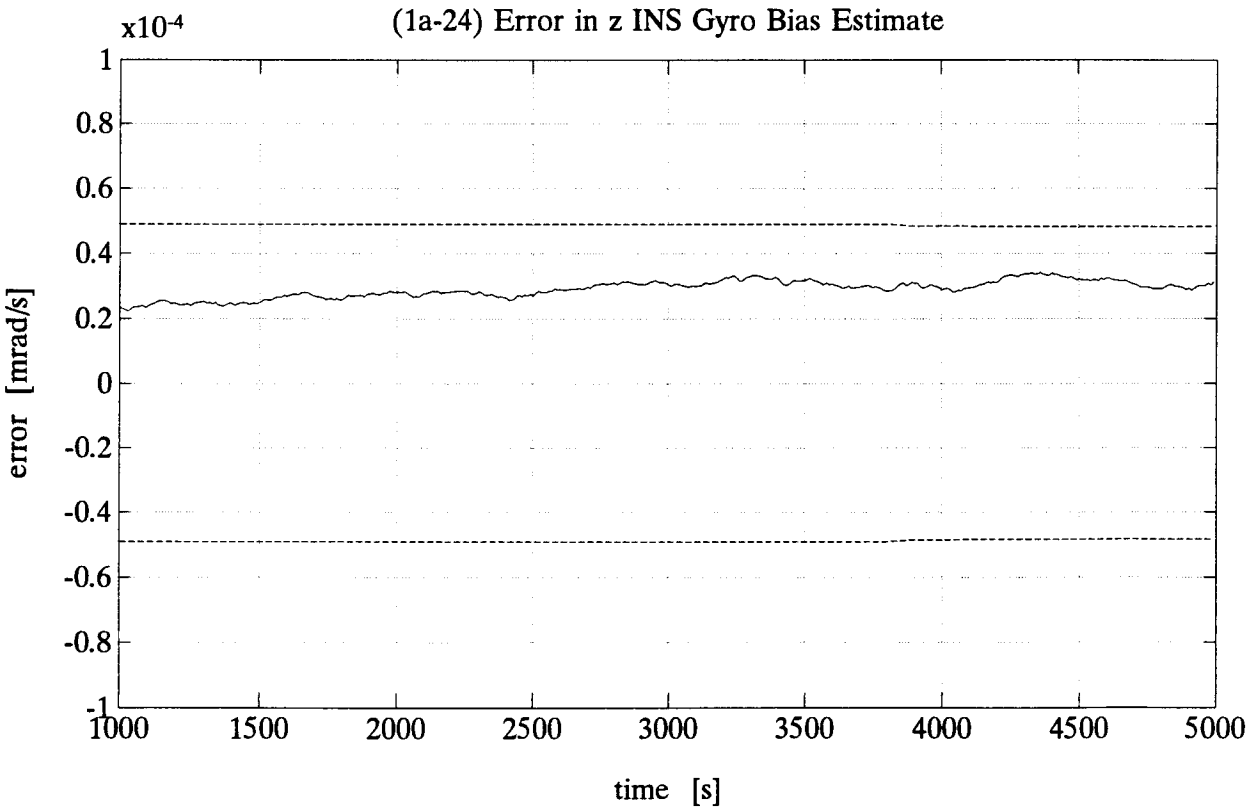
(1a-20) Error in x INS Accelerometer Bias Estimate



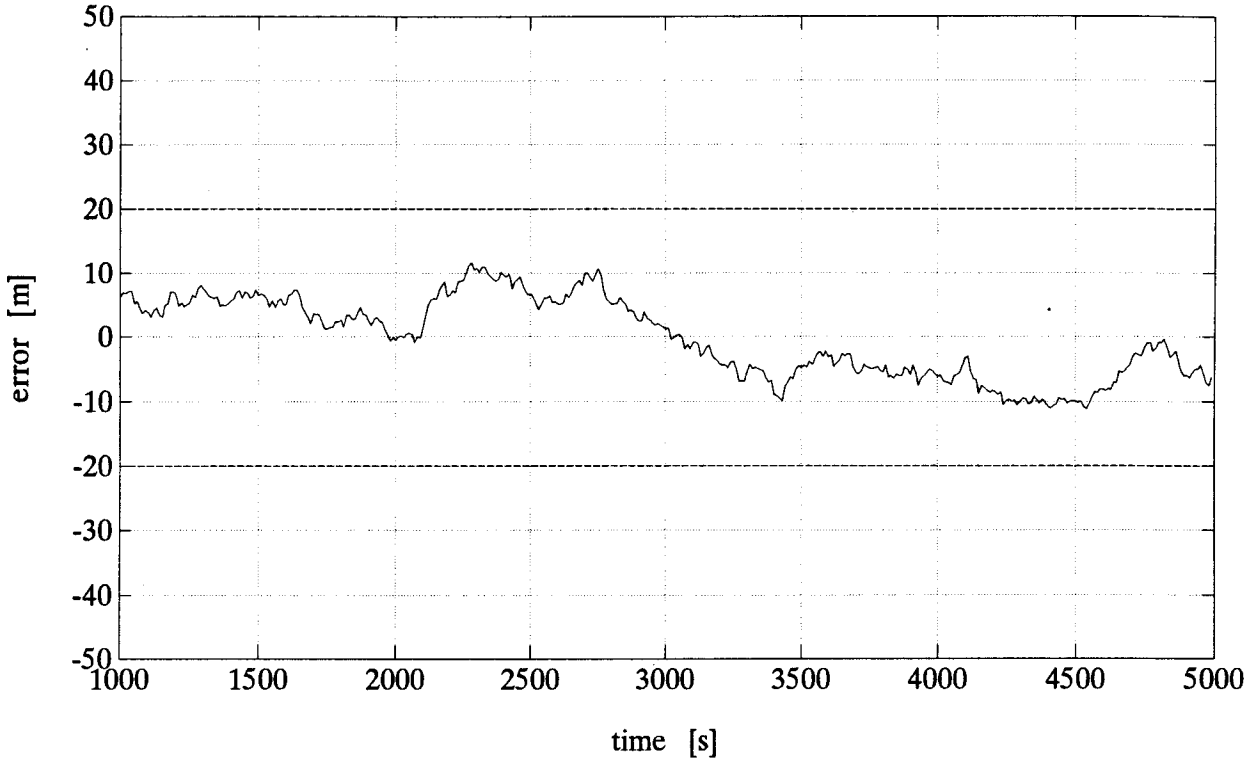

x10-3
(1a-21) Error in y INS Accelerometer Bias Estimate
error [m/s^ 2]
time [s]
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1000
1500
2000
2500
3000
3500
4000
4500
5000


x10-4
(1a-22) Error in x INS Gyro Bias Estimate
error [mrad/s]
time [s]
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1000
1500
2000
2500
3000
3500
4000
4500
5000

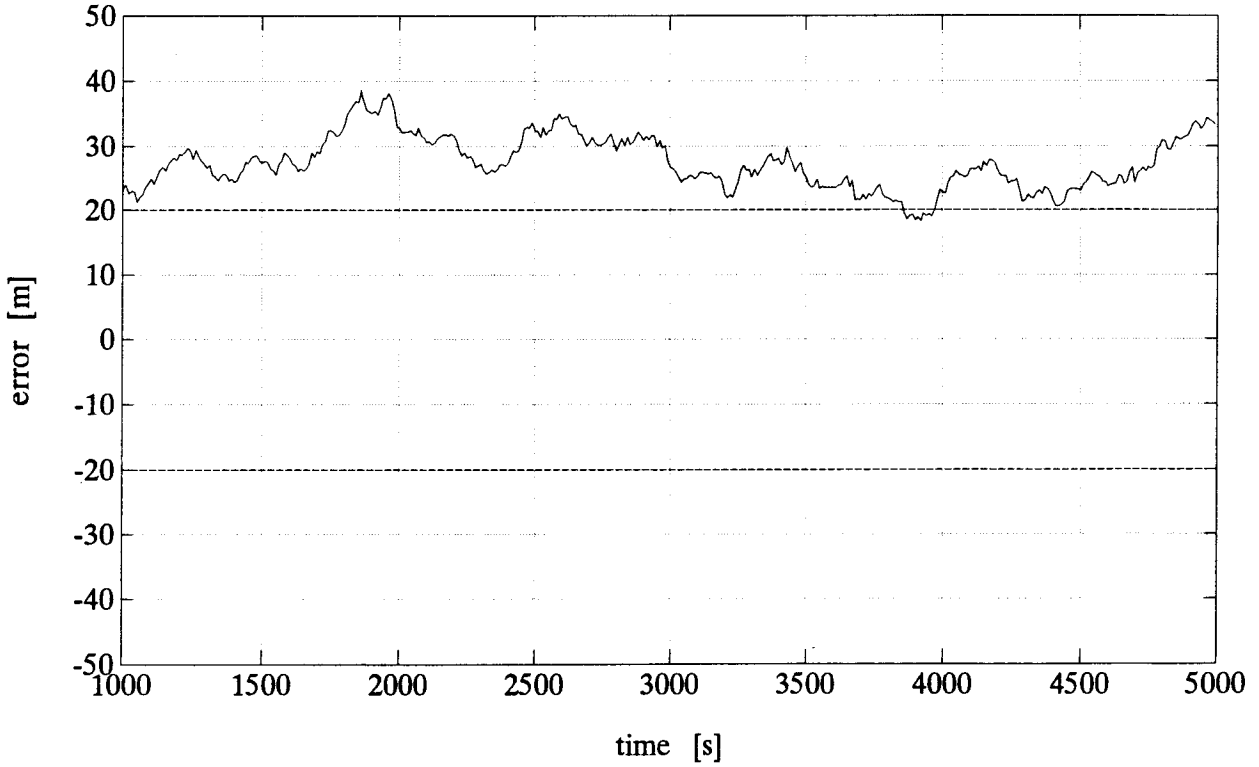

(1a-23) Error in y INS Gyro Bias Estimate
x10-4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


(1a-24) Error in z INS Gyro Bias Estimate
x10-4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

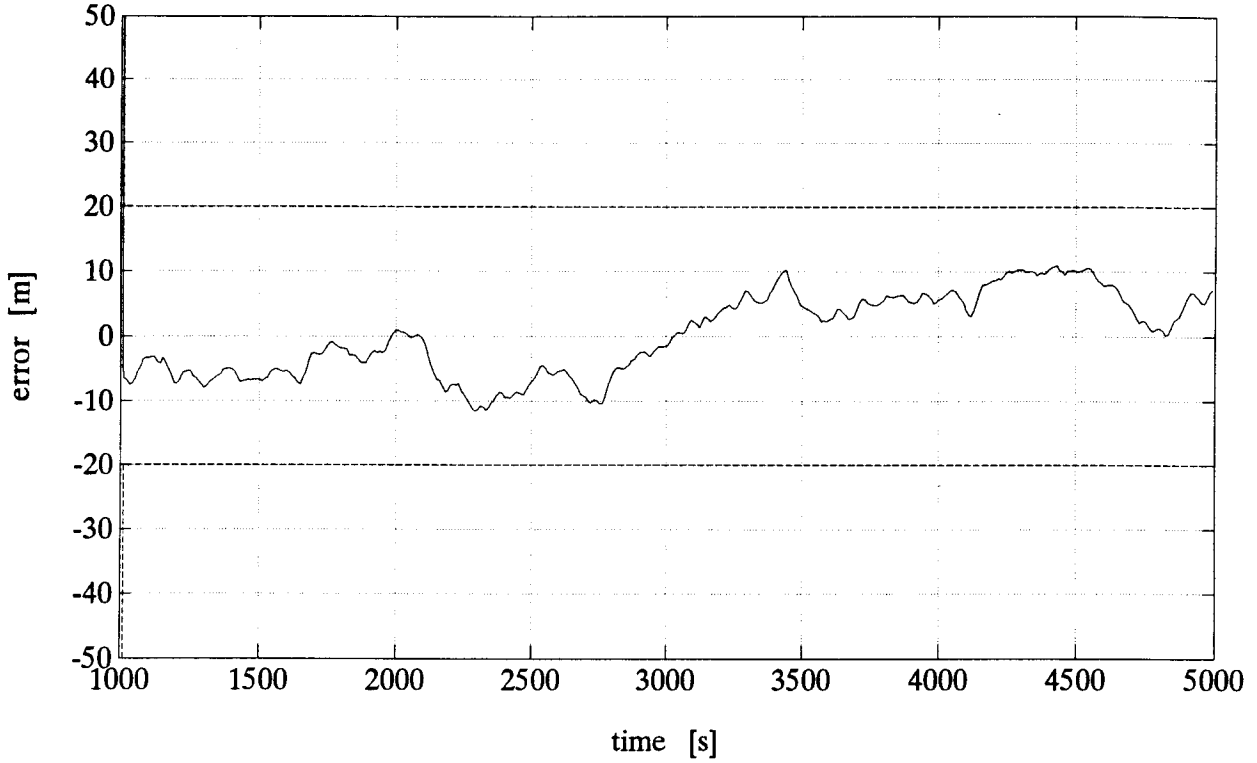
(1a-25) Error in x GPS Position Error Estimate



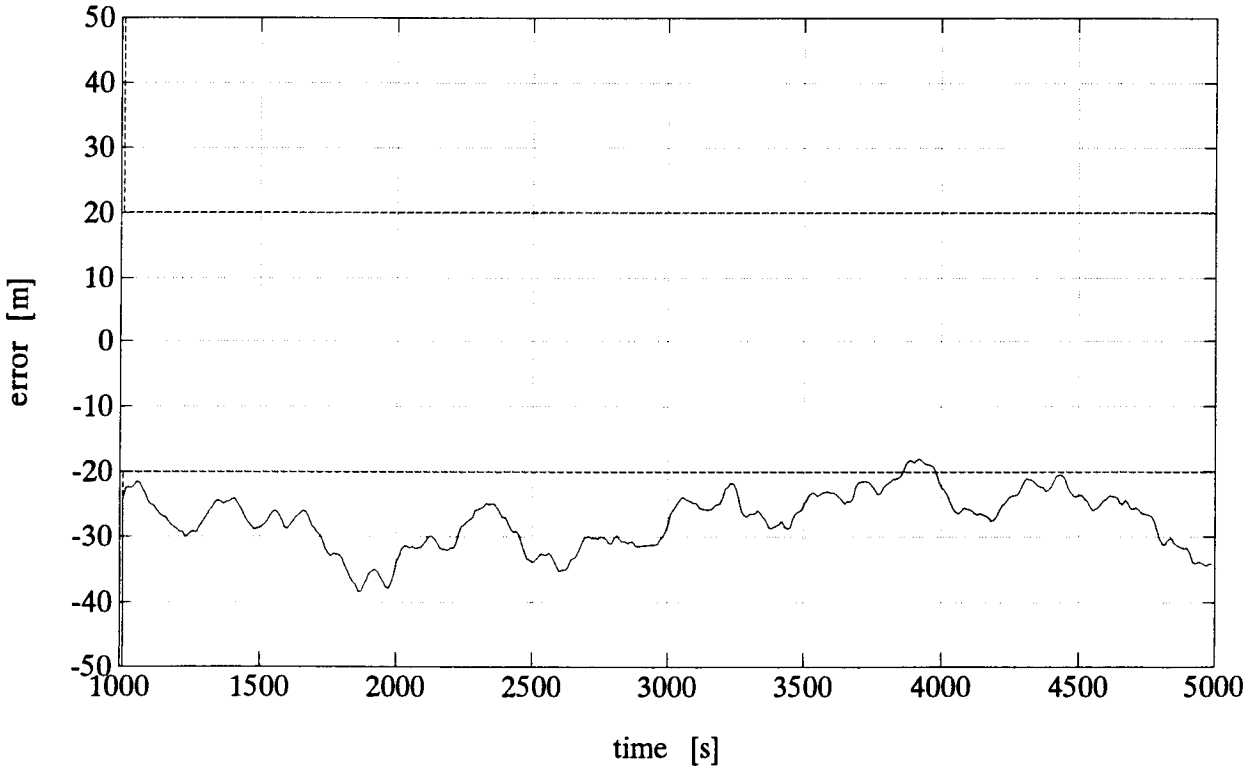
(1a-26) Error in y GPS Position Error Estimate



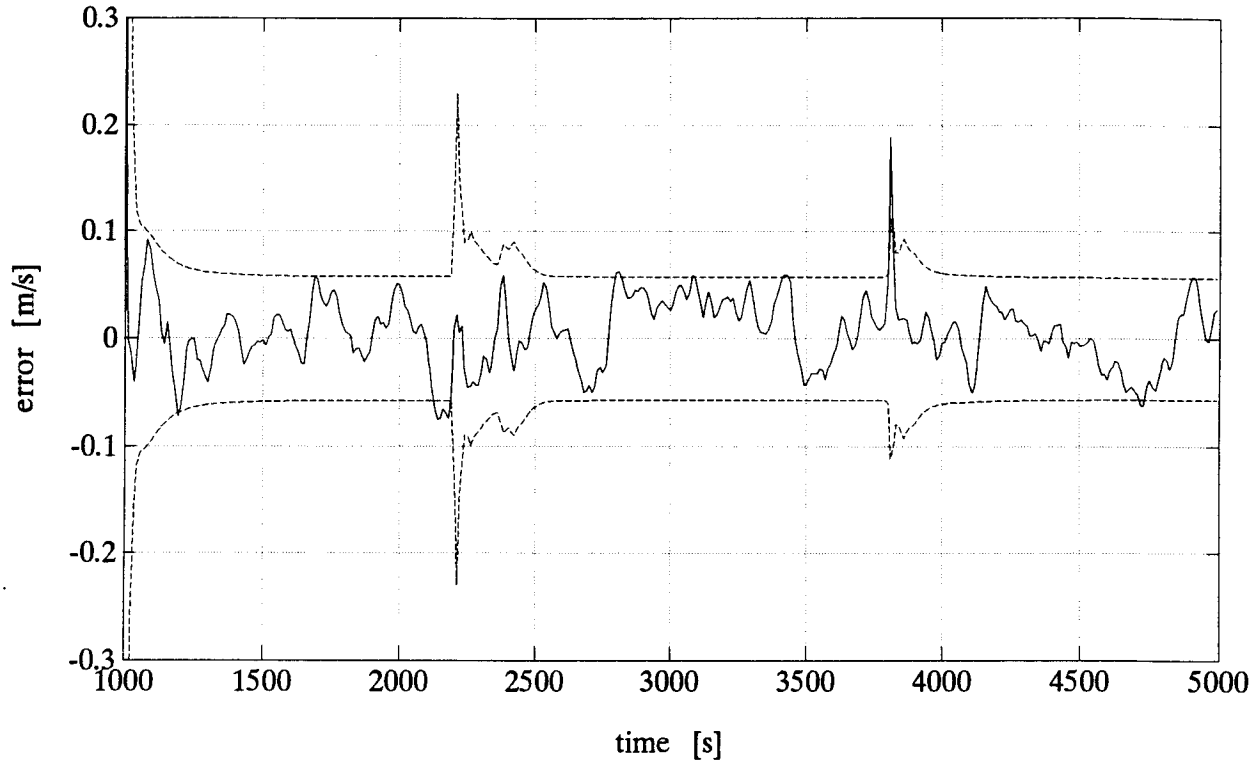
(1a-27) Error in Corrected x IMU Position



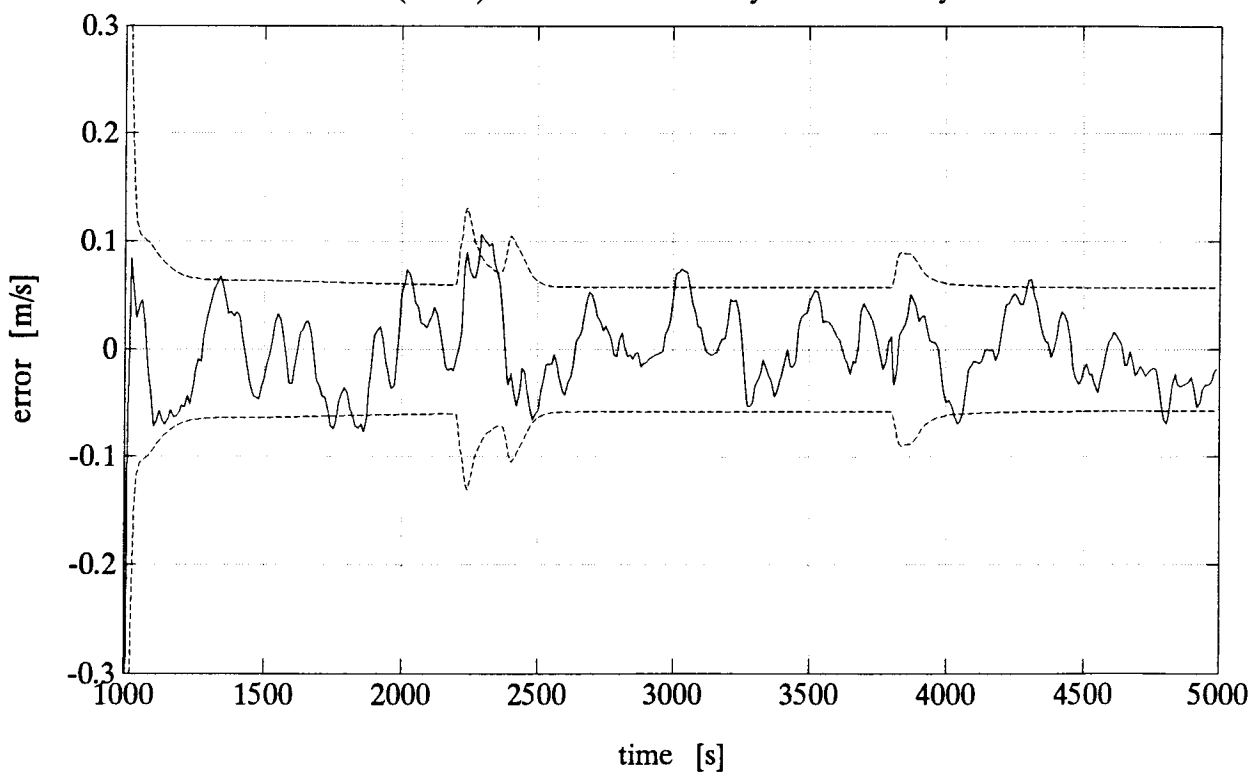
(1a-28) Error in Corrected y IMU Position



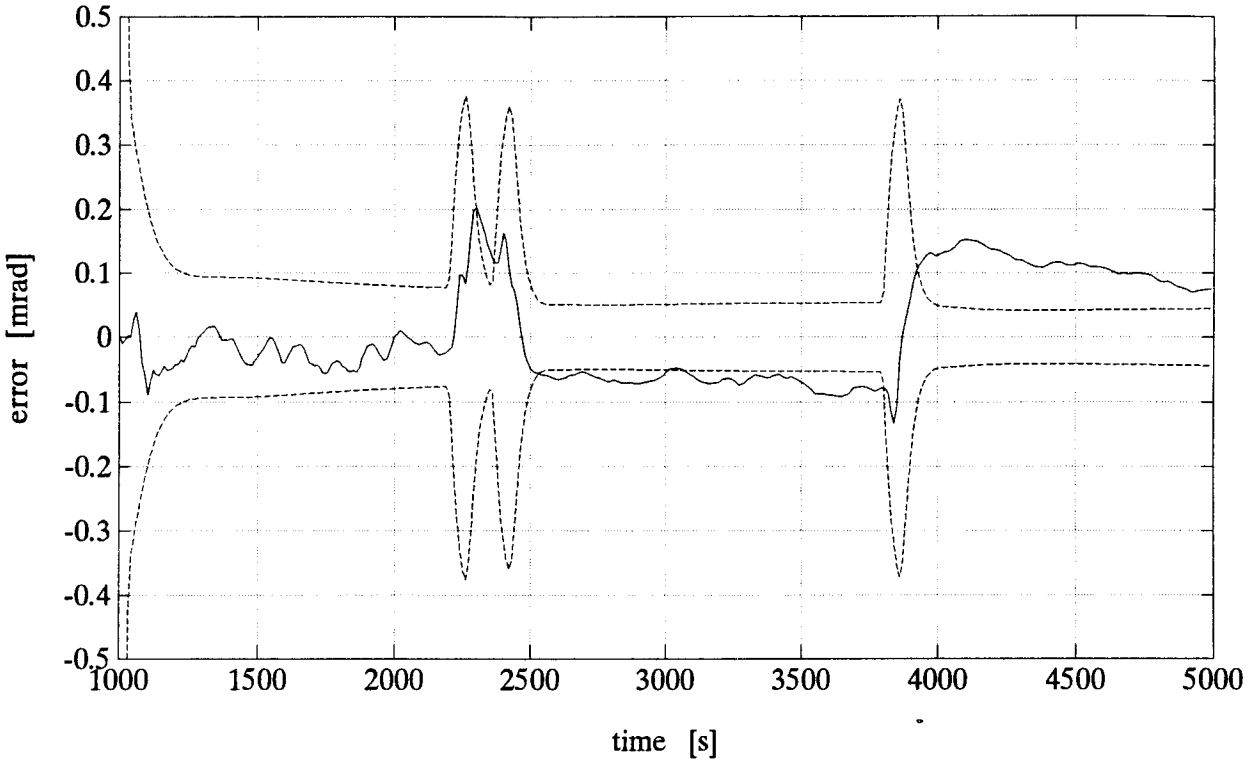
(1a-29) Error in Corrected x IMU Velocity



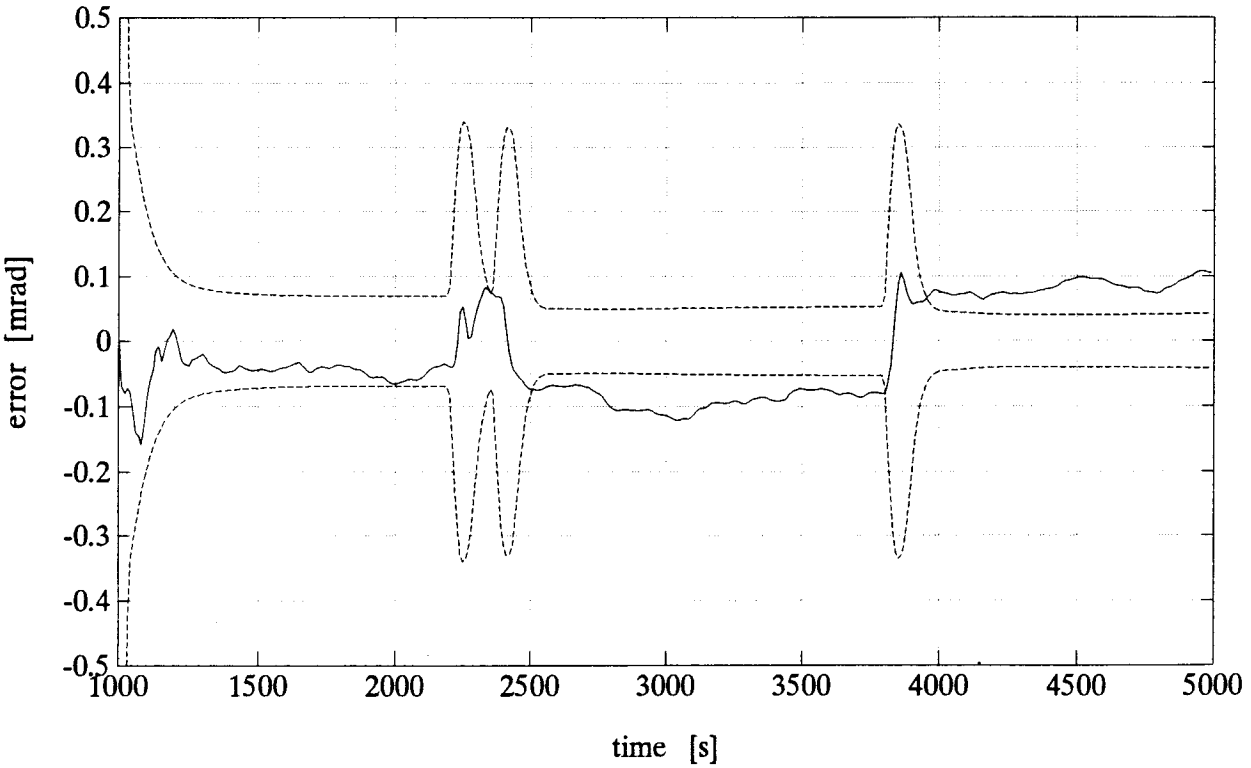
(1a-30) Error in Corrected y IMU Velocity



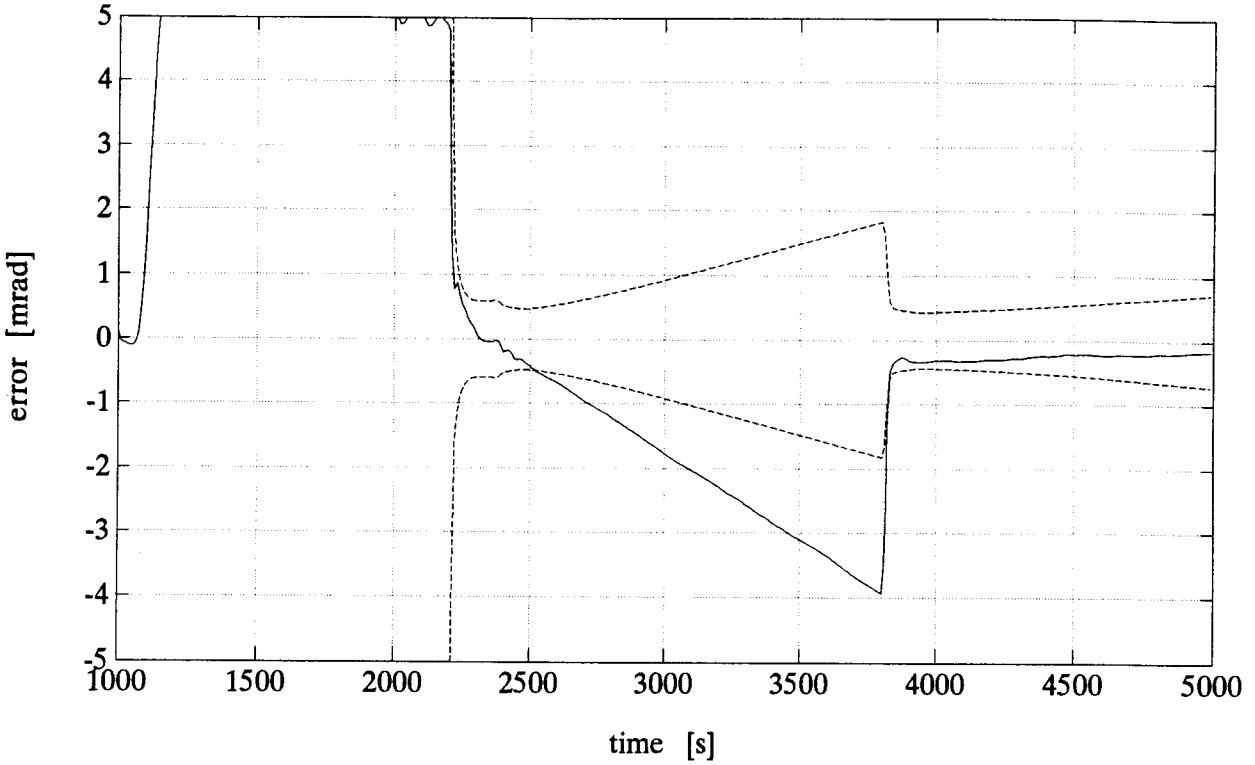
(1a-31) Error in Corrected x IMU Platform Alignment



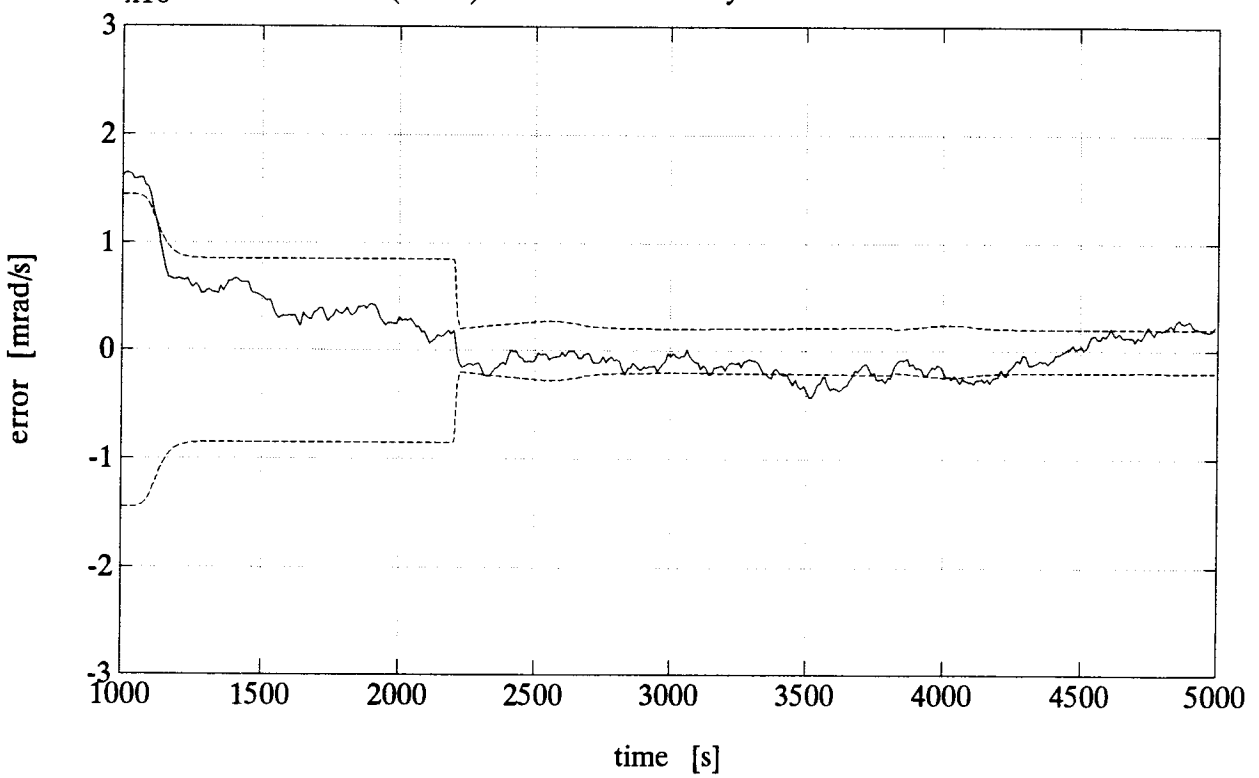
(1a-32) Error in Corrected y IMU Platform Alignment

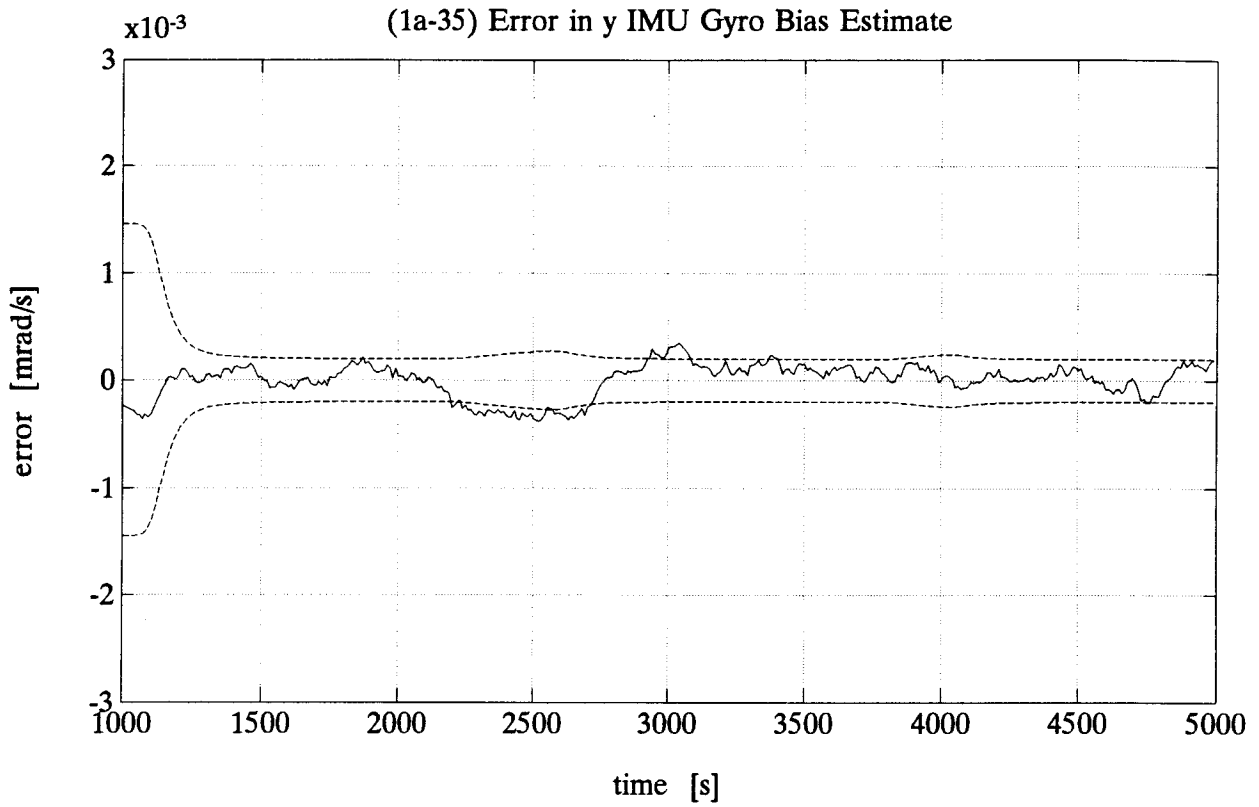


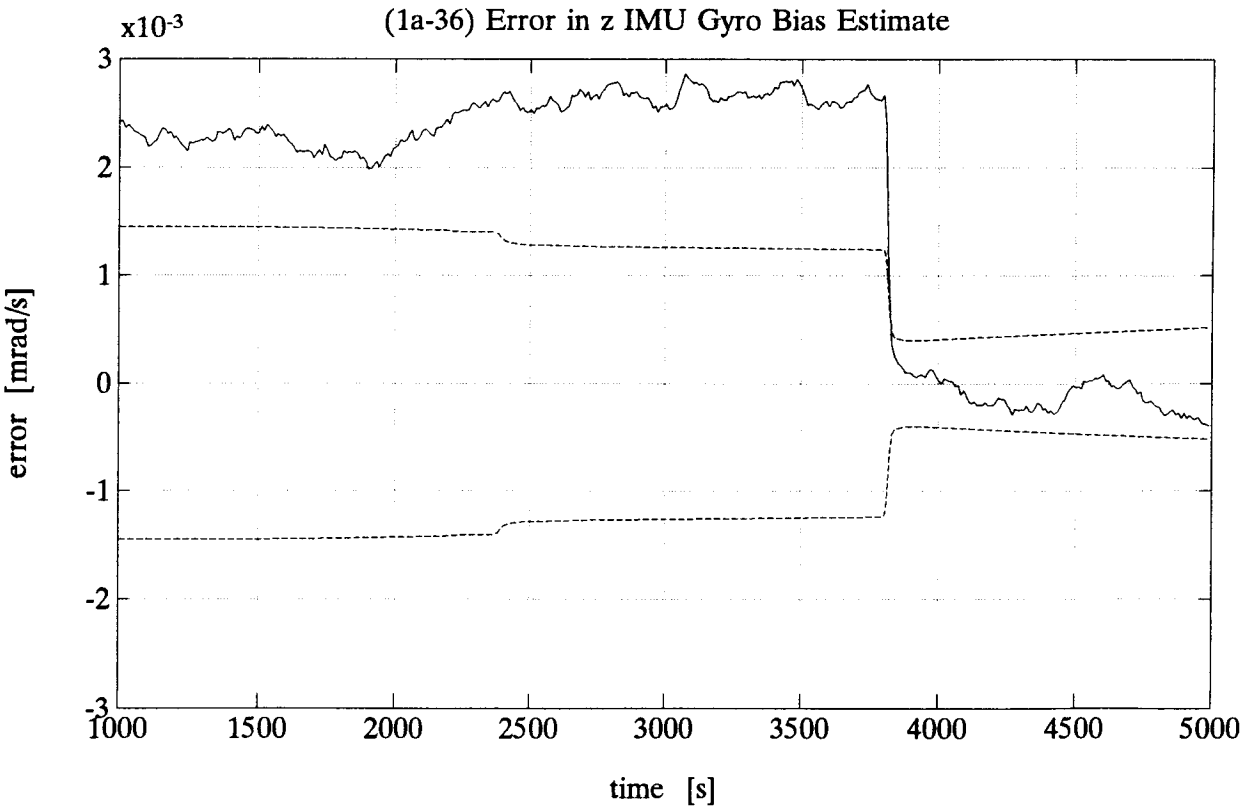
(1a-33) Error in Corrected z IMU Platform Alignment



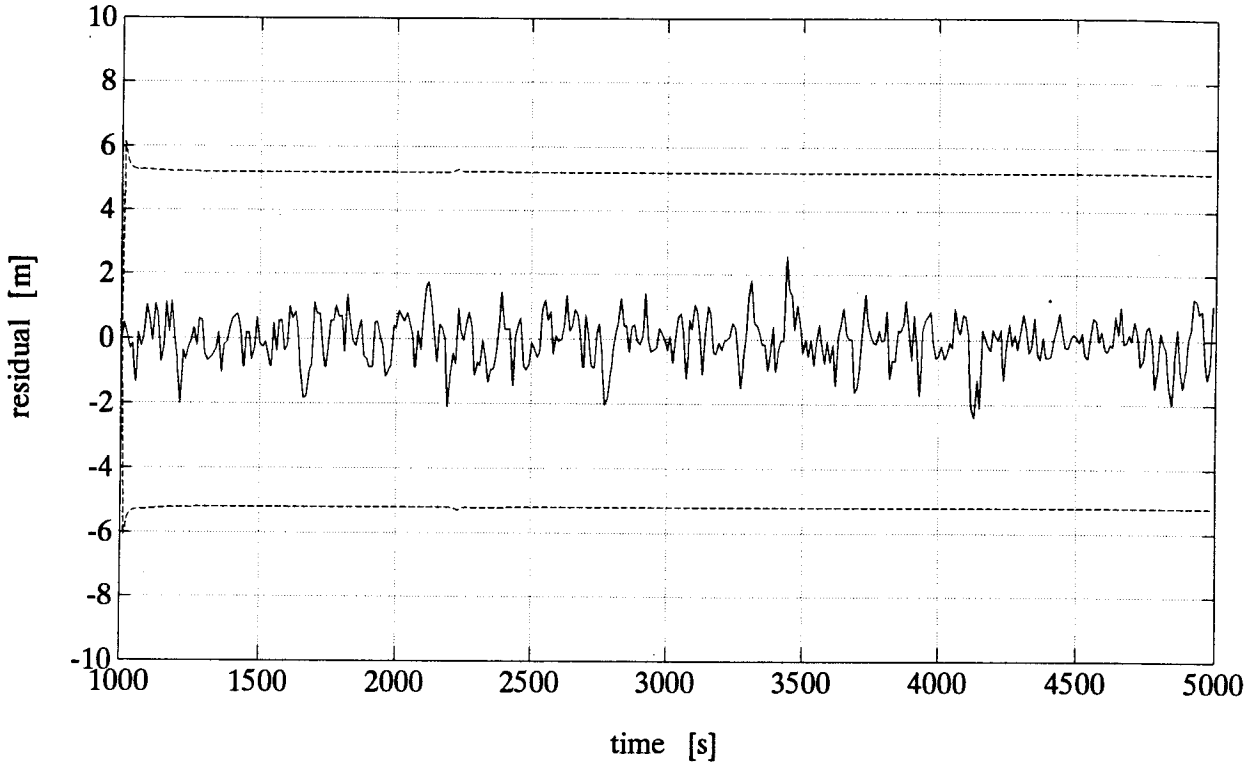
(1a-34) Error in x IMU Gyro Bias Estimate



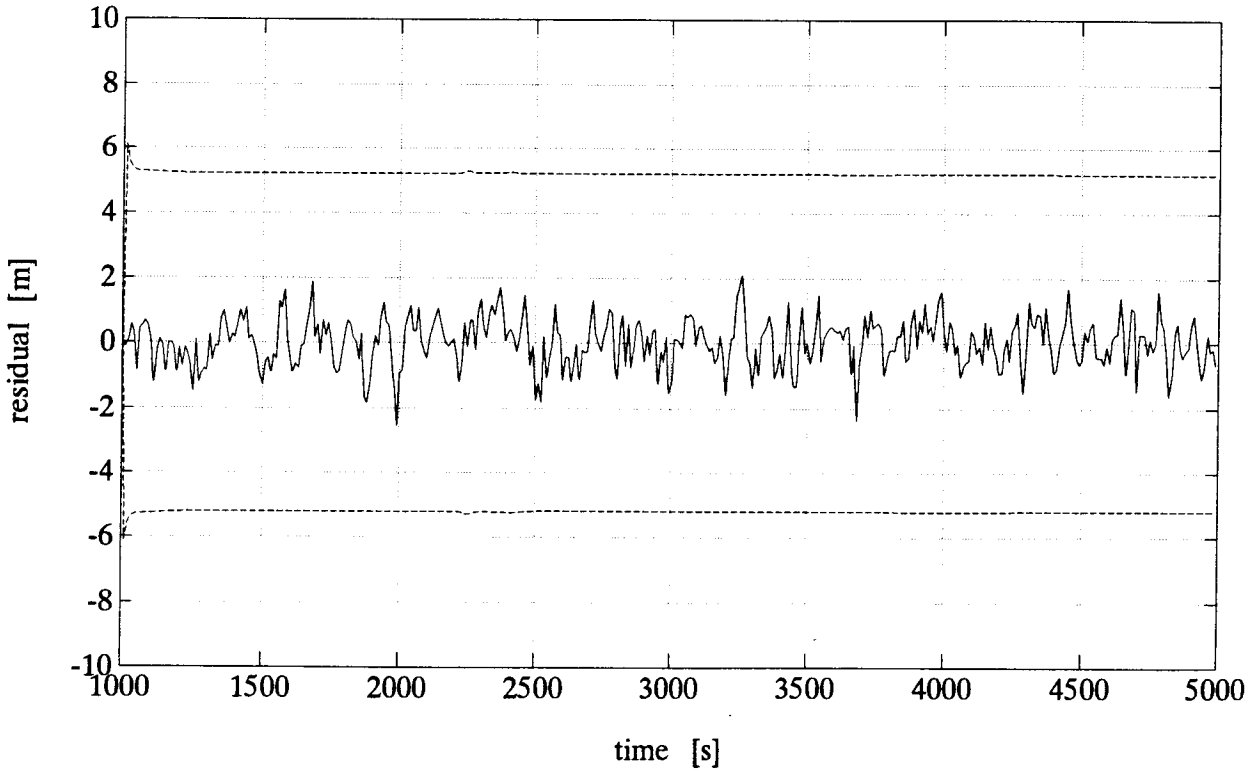

x10-3
(1a-35) Error in y IMU Gyro Bias Estimate
3
2
1
0
-1
-2
-3
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


x10-3
(1a-36) Error in z IMU Gyro Bias Estimate
3
2
1
0
-1
-2
-3
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

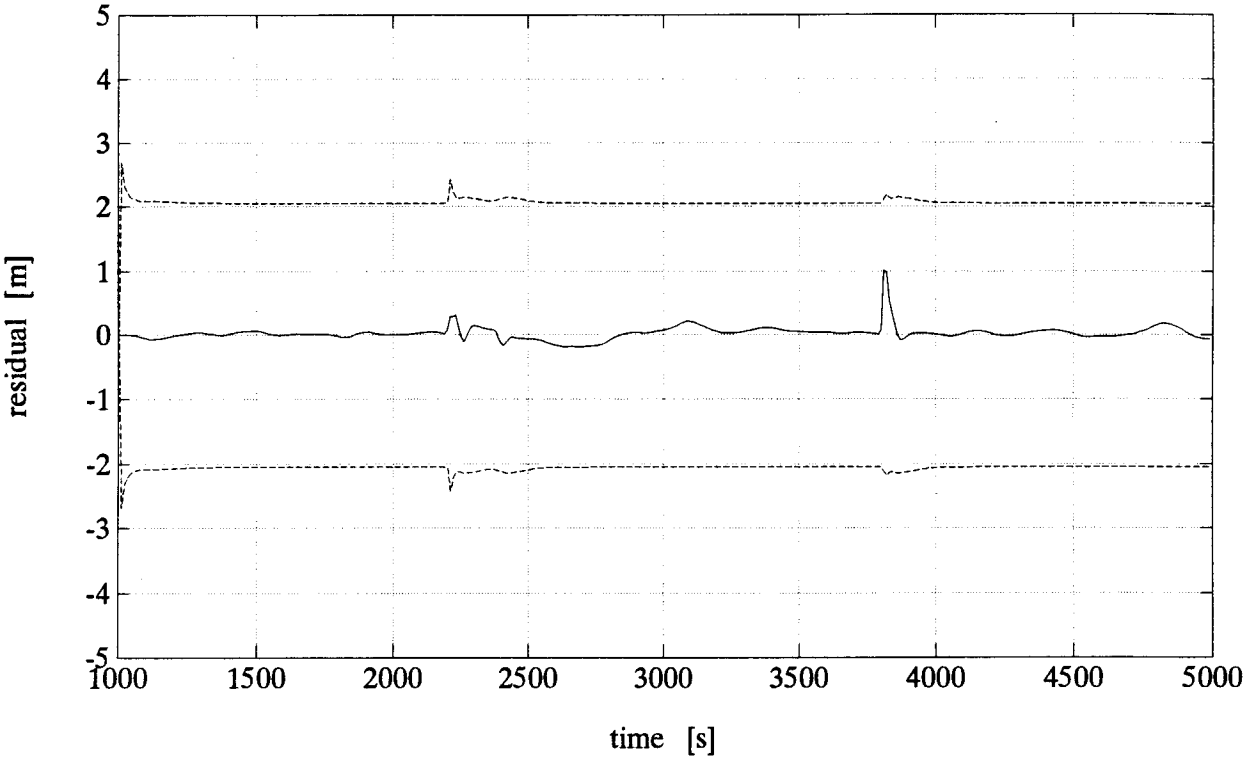
(1a-37) x INS/GPS Position Matching Measurement Residual



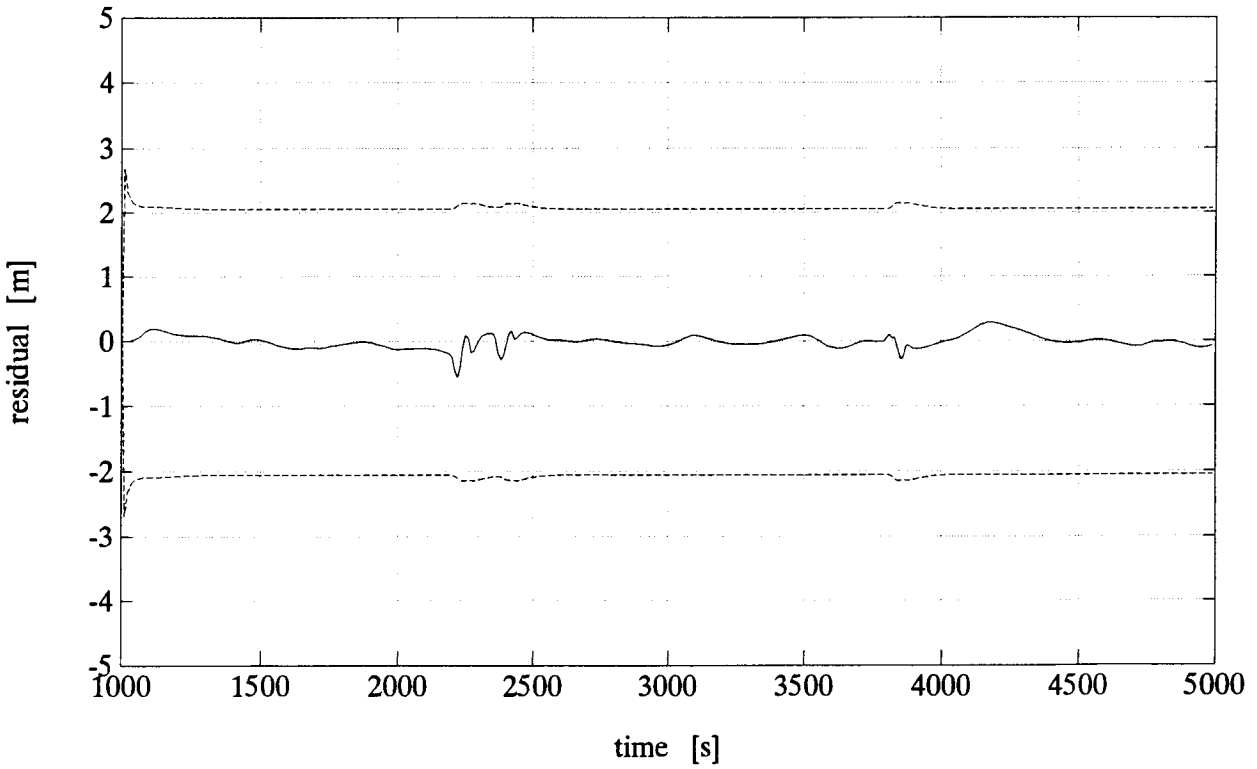
(1a-38) y INS/GPS Position Matching Measurement Residual



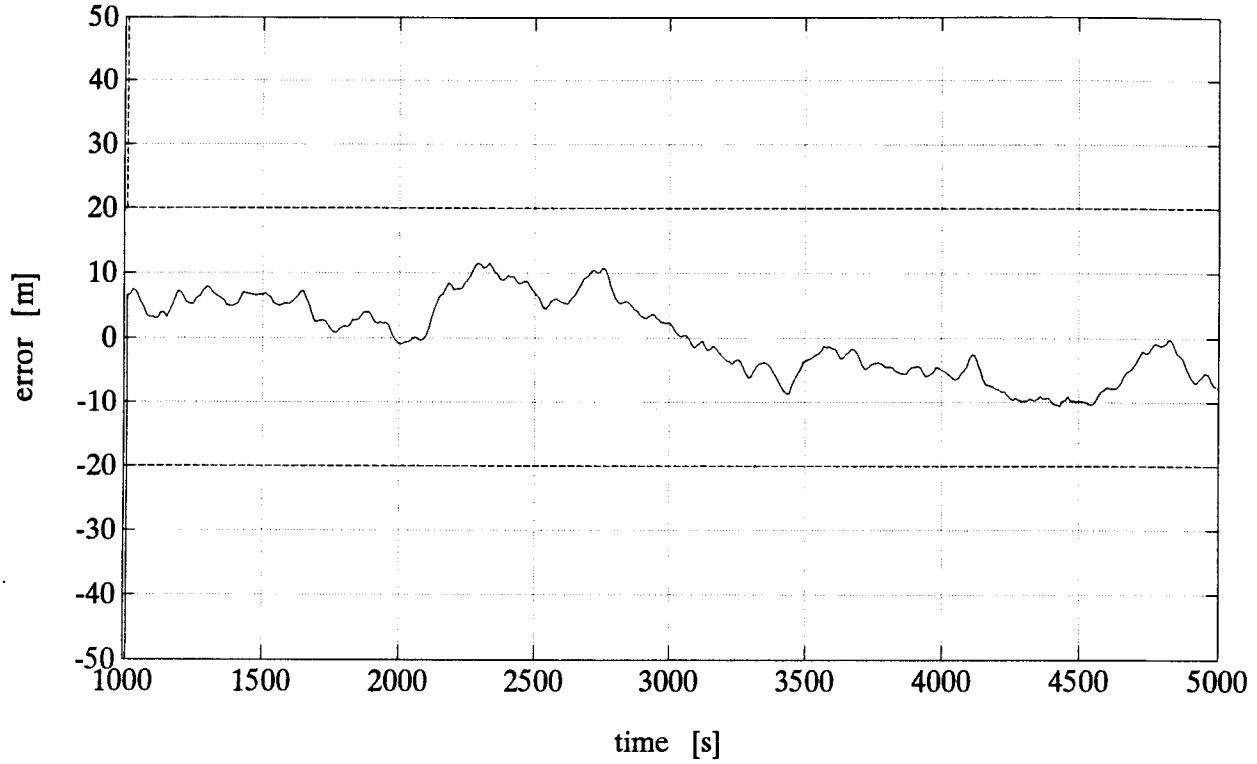
(1a-39) x IMU/INS Position Matching Measurement Residual



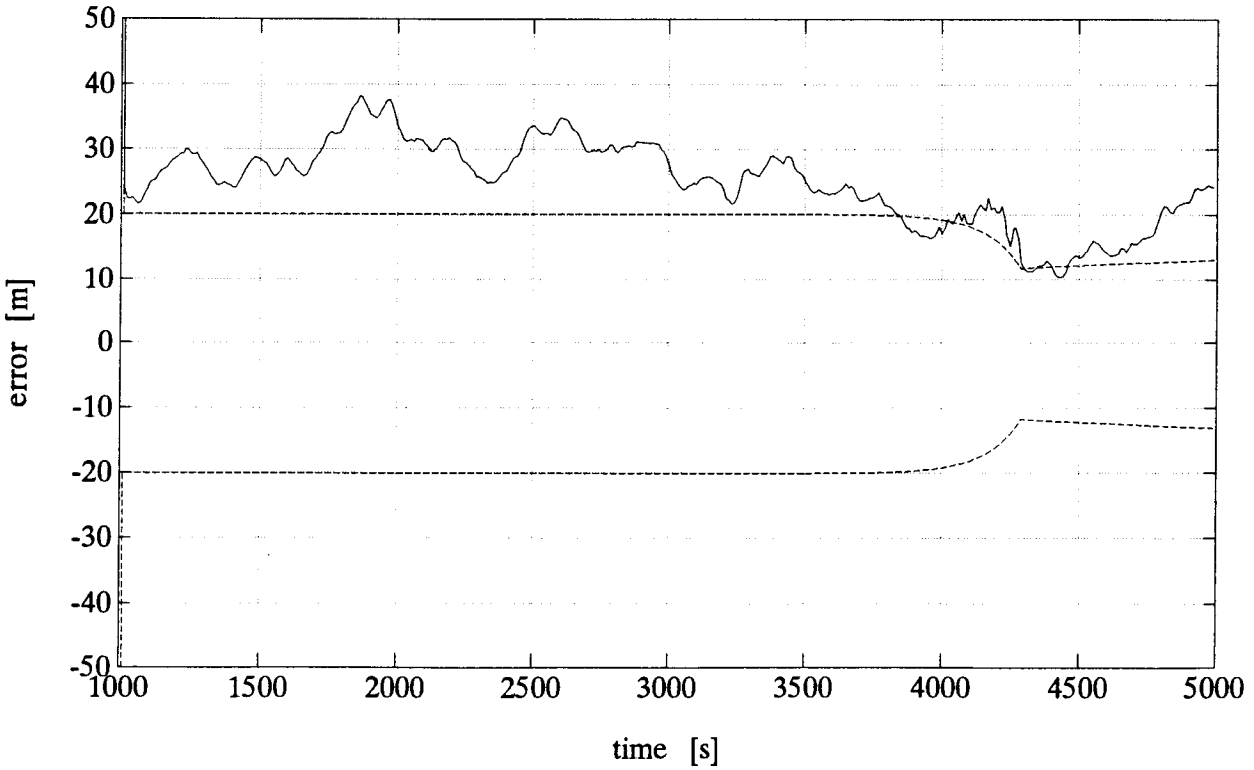
(1a-40) y IMU/INS Position Matching Measurement Residual



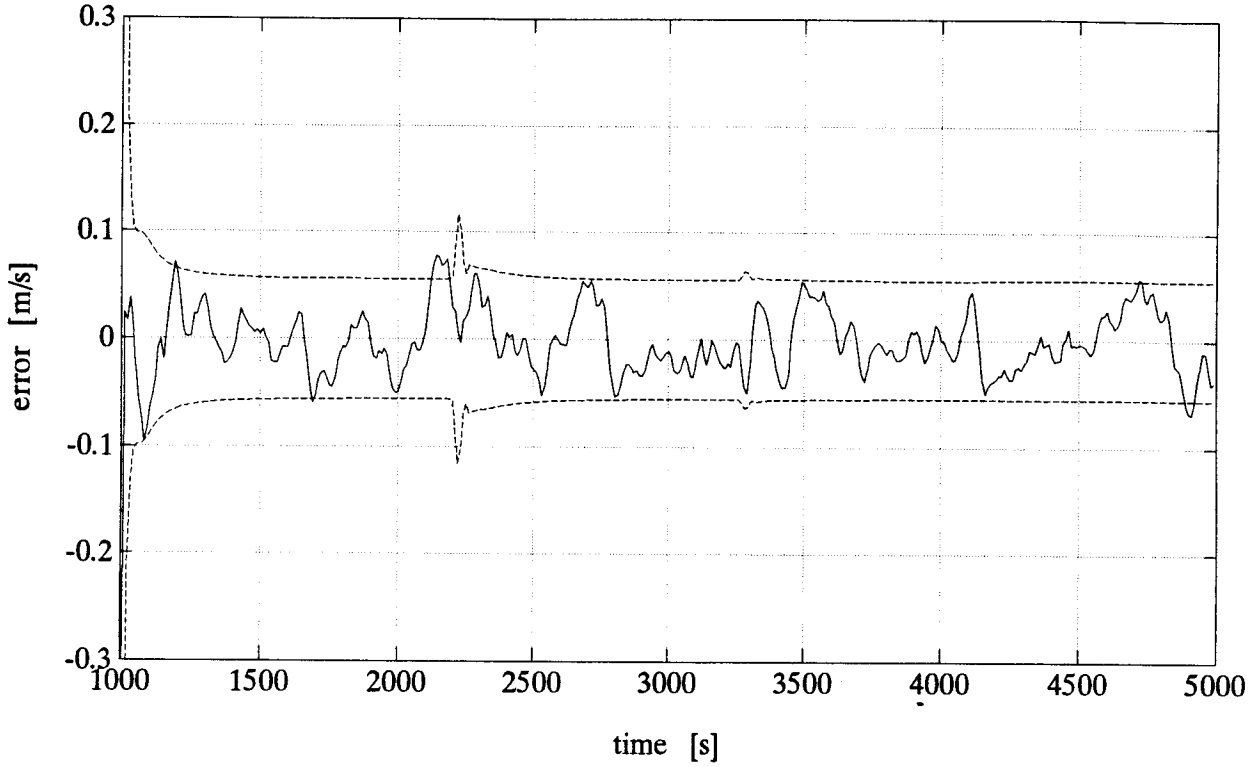
(2a-1) Error in x INS Position Error Estimate



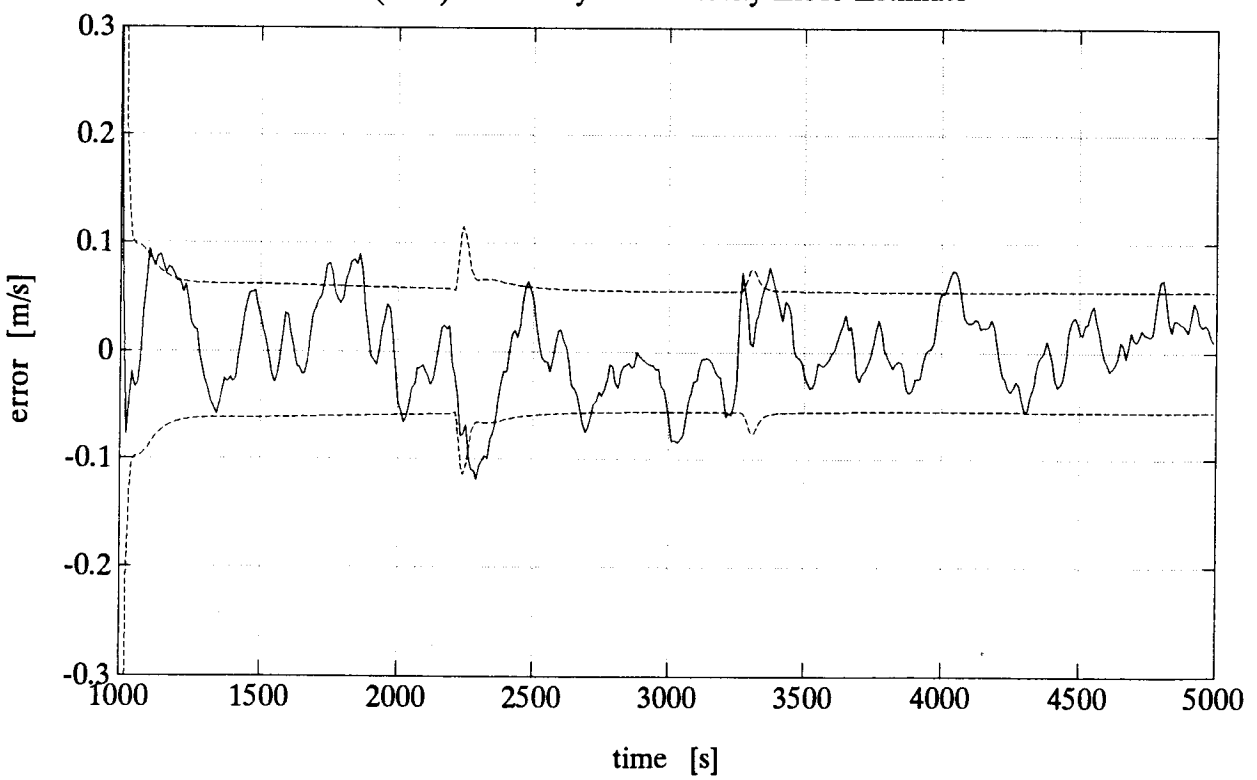
(2a-2) Error in y INS Position Error Estimate



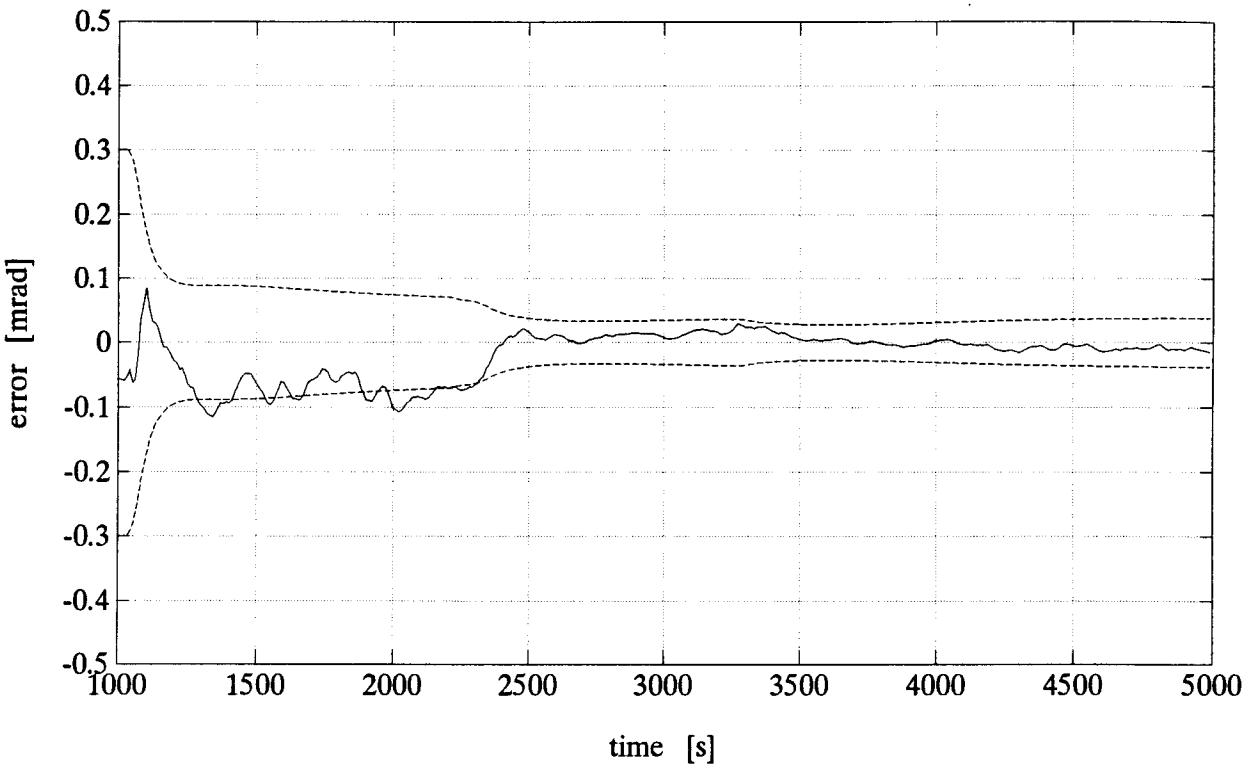
(2a-3) Error in x INS Velocity Error Estimate



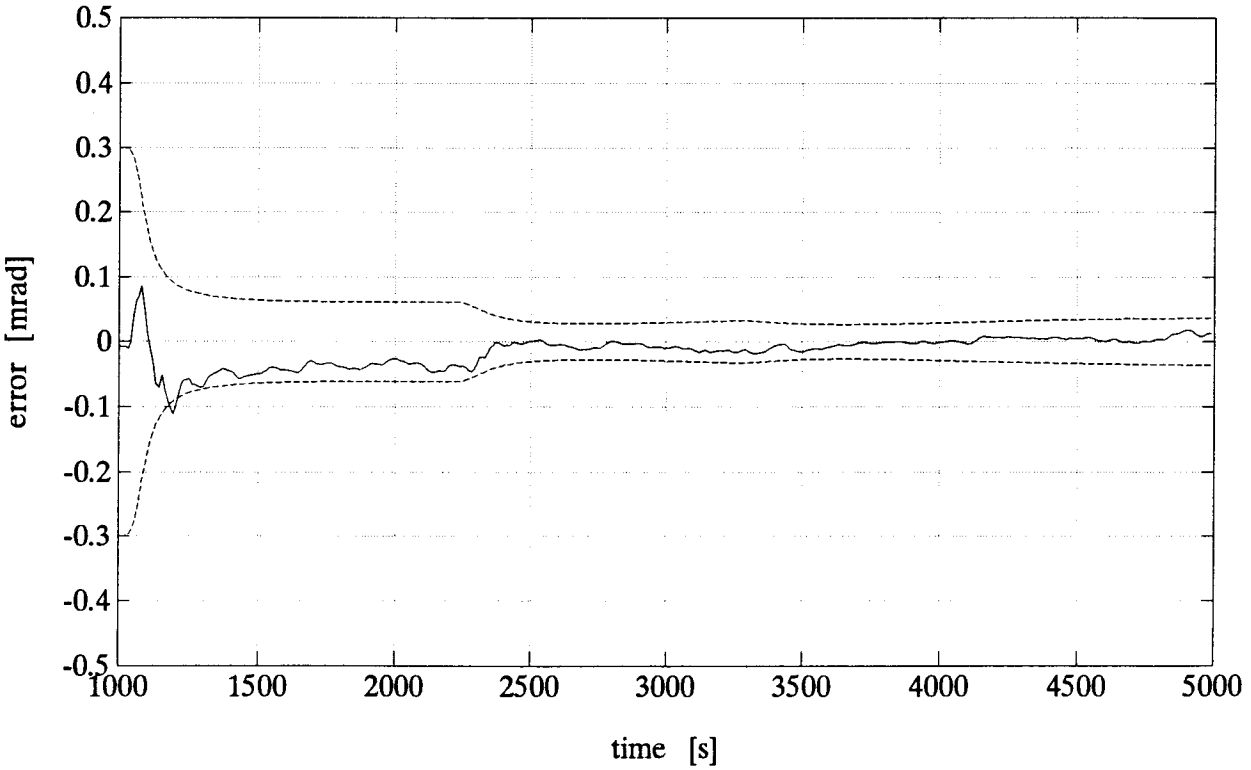
(2a-4) Error in y INS Velocity Error Estimate



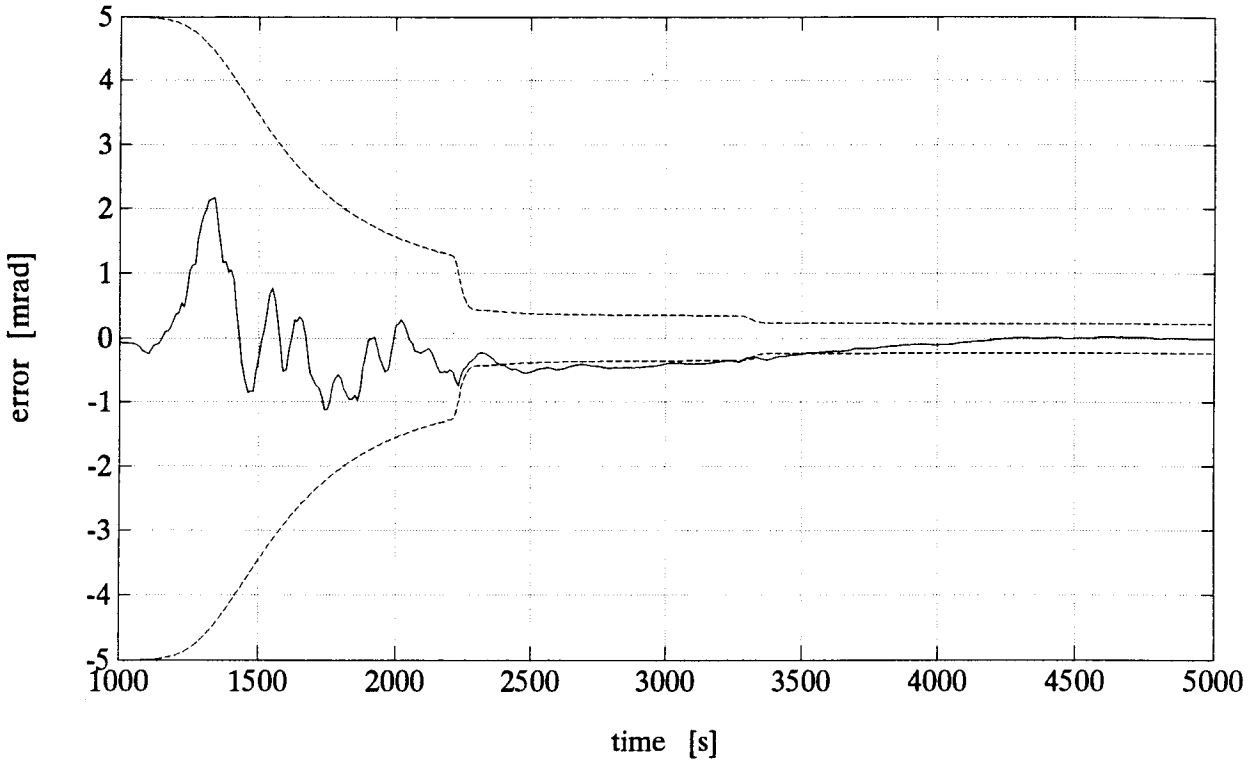
(2a-5) Error in x INS Platform Misalignment Estimate



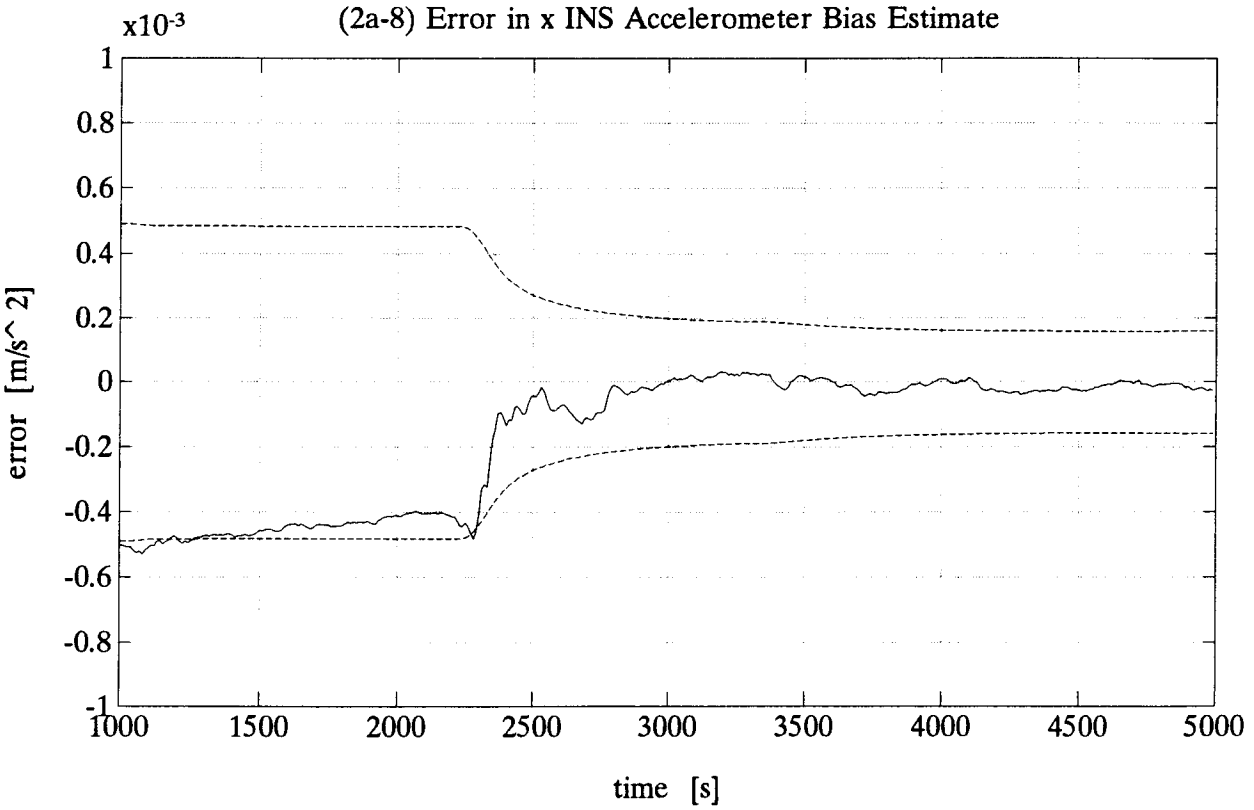
(2a-6) Error in y INS Platform Misalignment Estimate

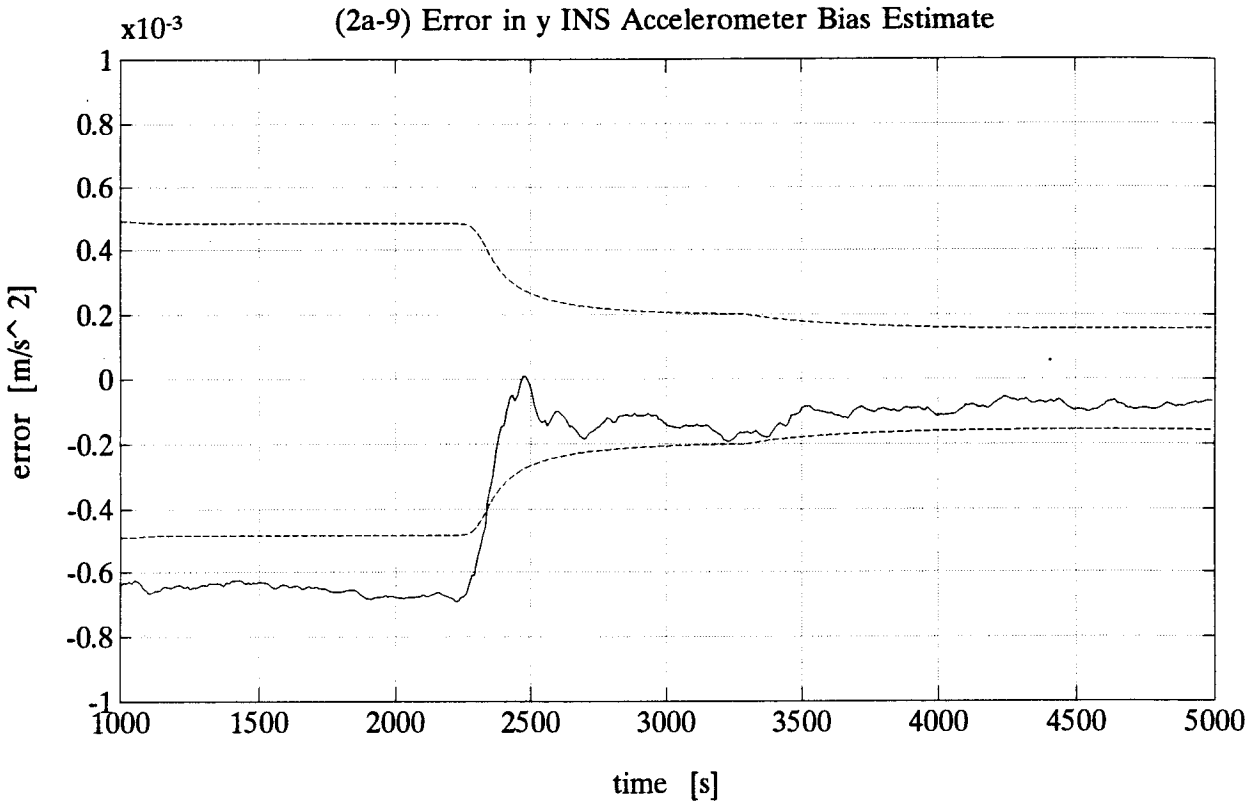


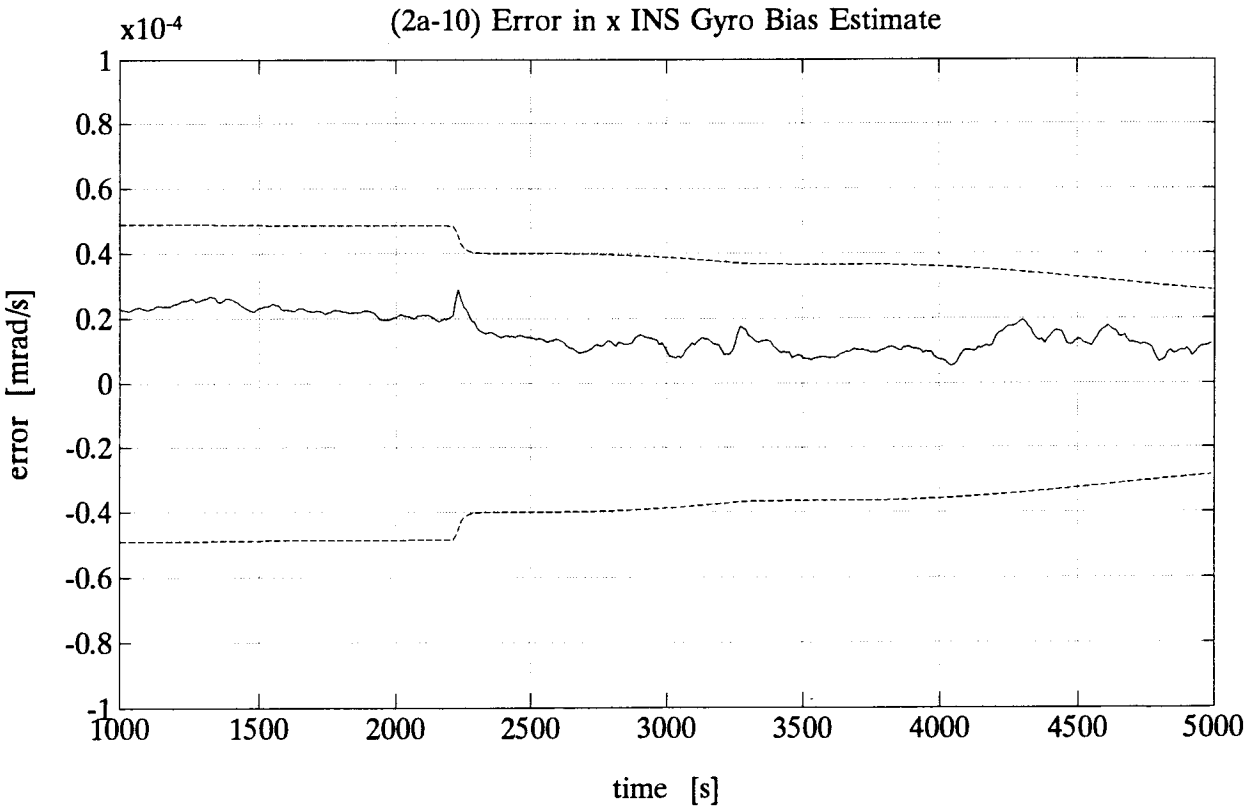
(2a-7) Error in z INS Platform Misalignment Estimate

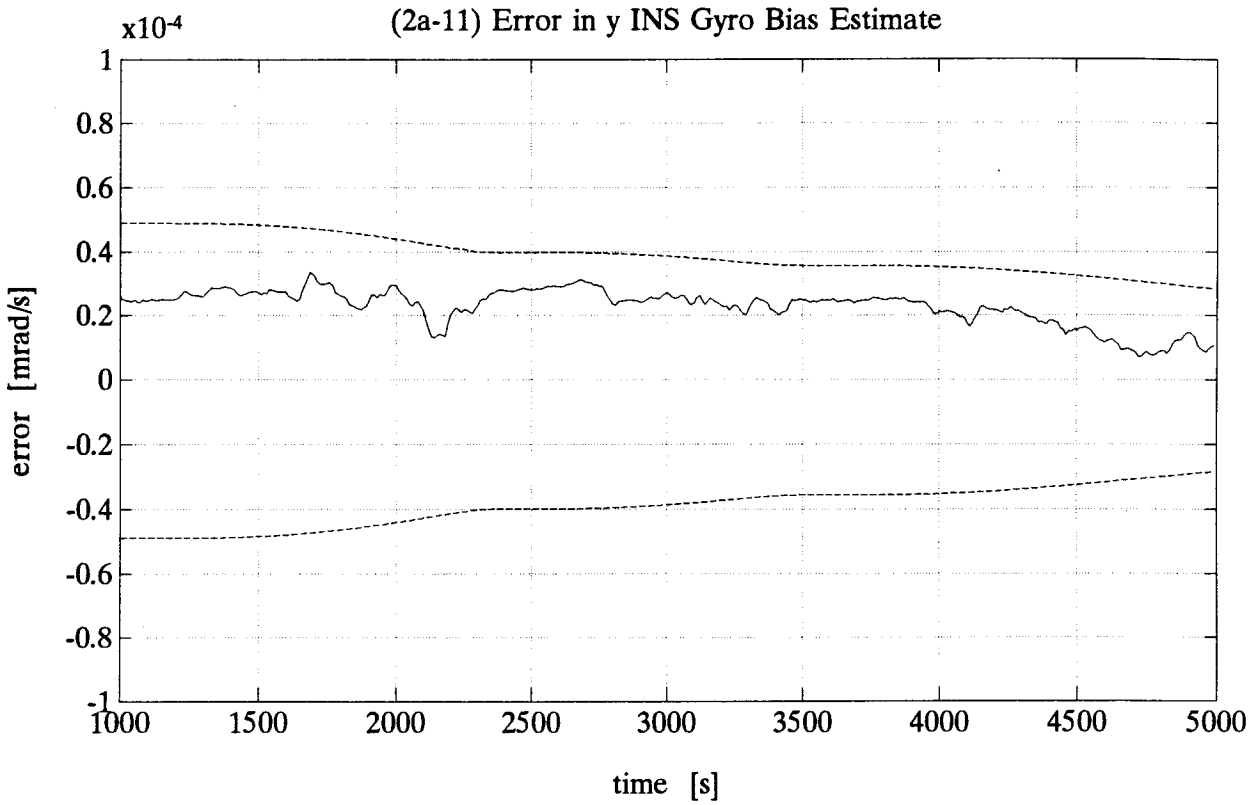


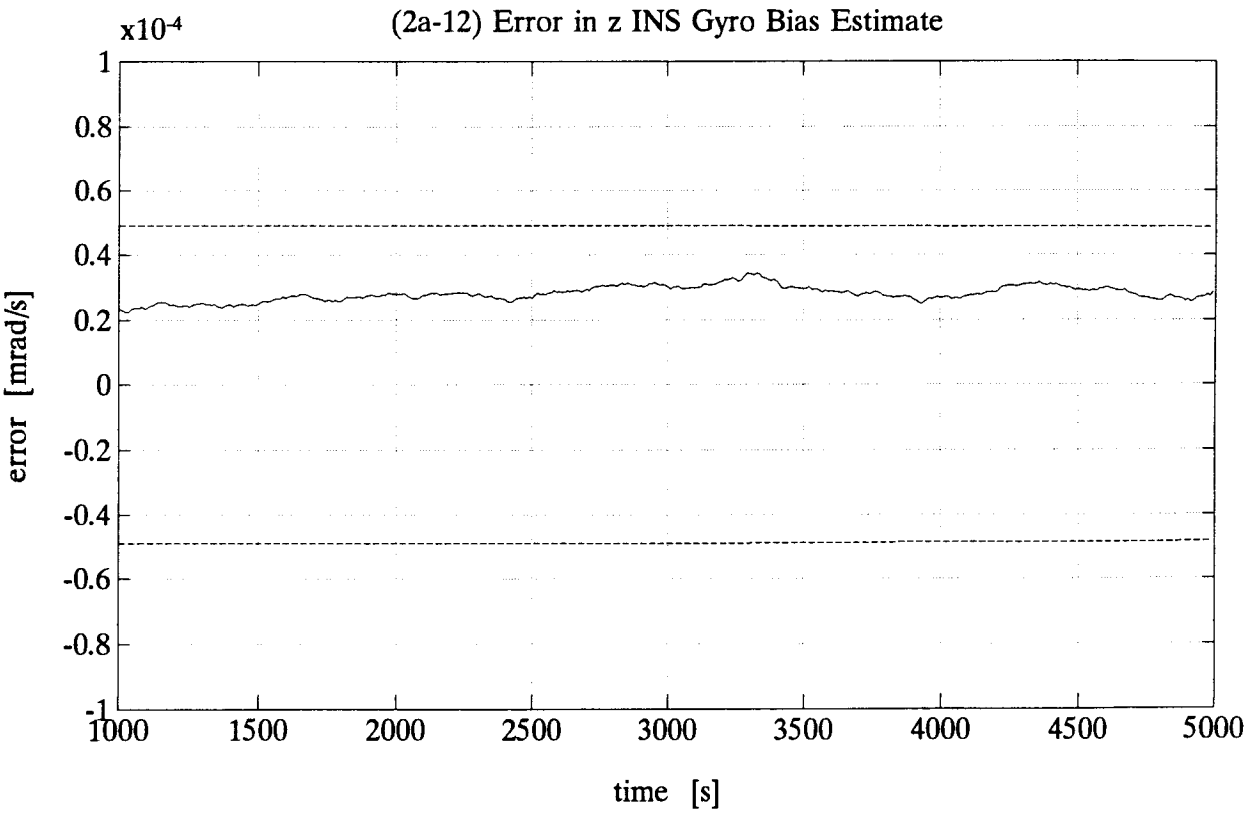
(2a-8) Error in x INS Accelerometer Bias Estimate



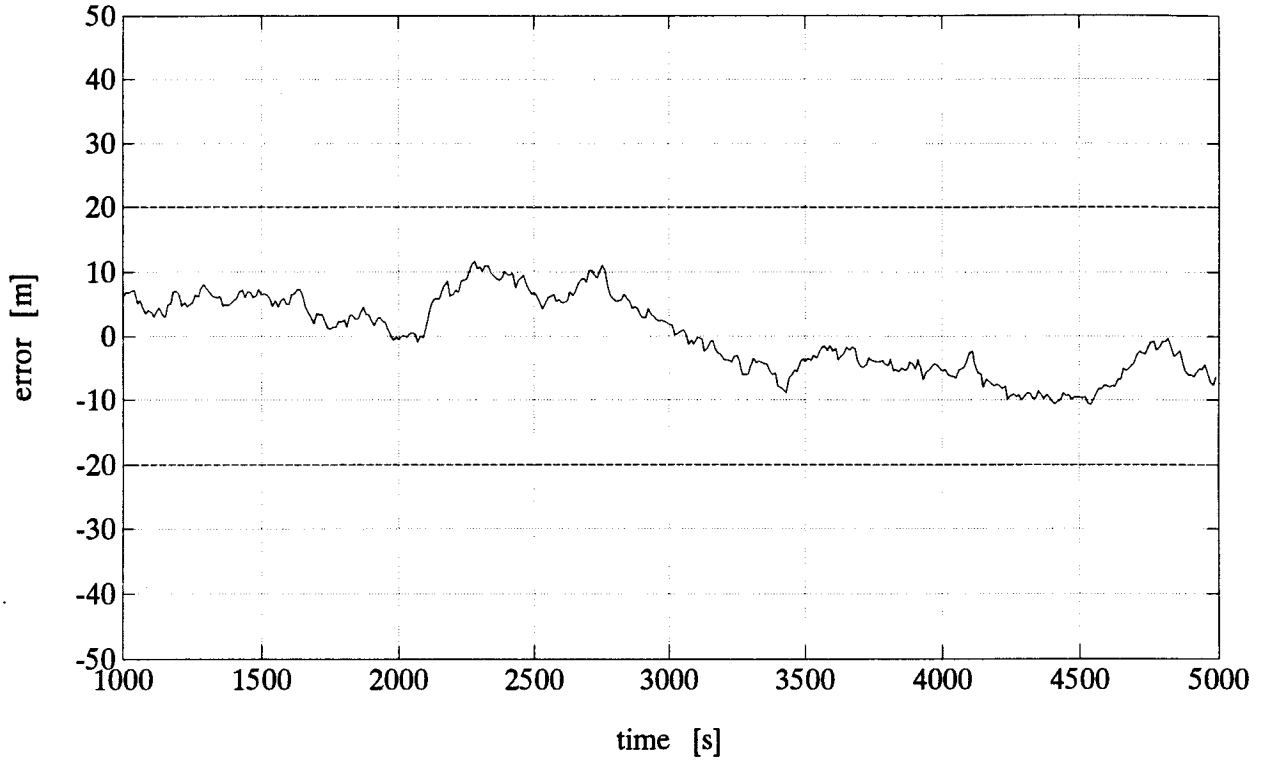

(2a-9) Error in y INS Accelerometer Bias Estimate
x10-3
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [m/s^ 2]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


(2a-10) Error in x INS Gyro Bias Estimate
x10-4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

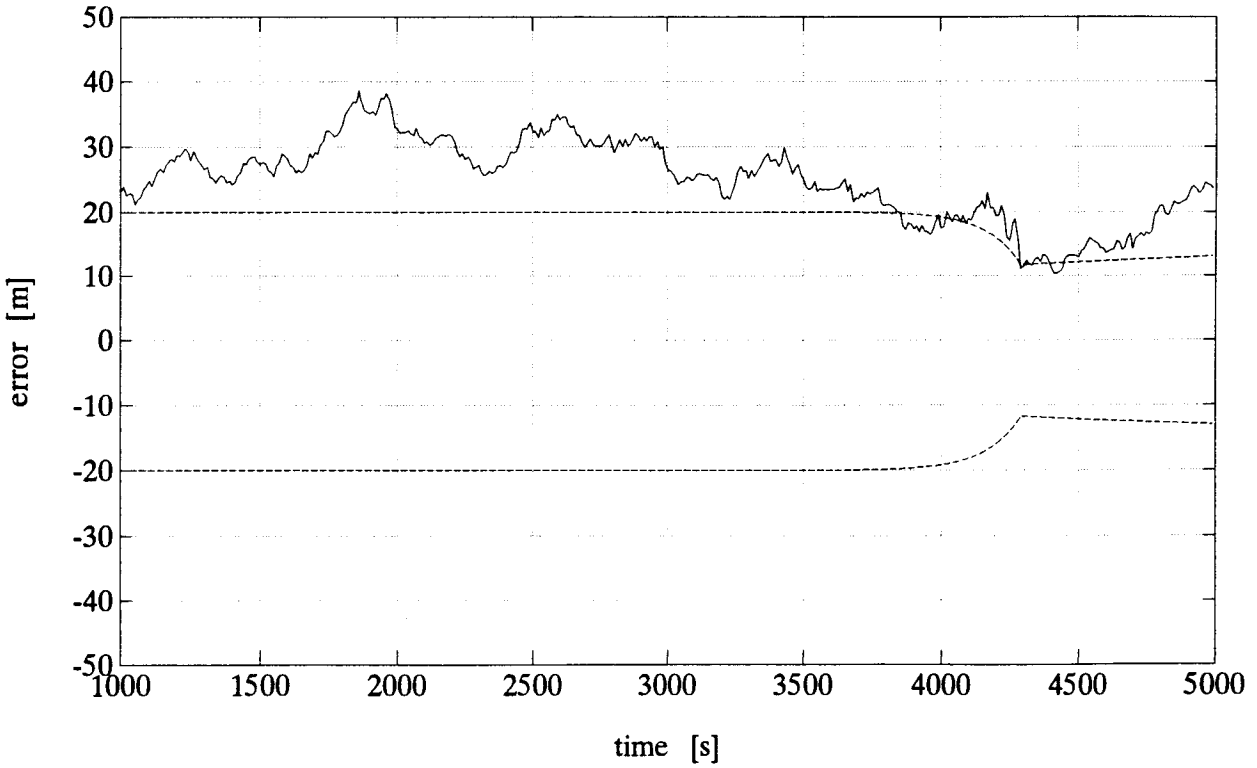

(2a-11) Error in y INS Gyro Bias Estimate
x10-4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


(2a-12) Error in z INS Gyro Bias Estimate
x10-4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

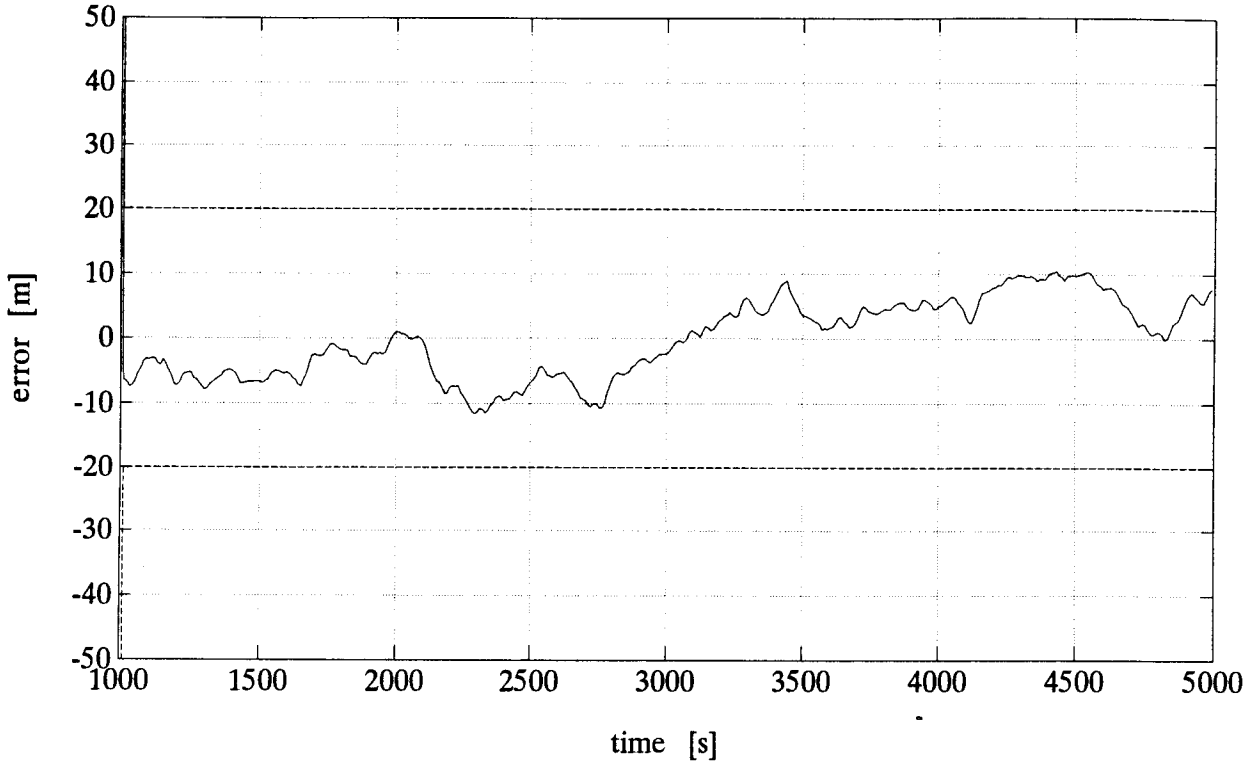
(2a-13) Error in x GPS Position Error Estimate



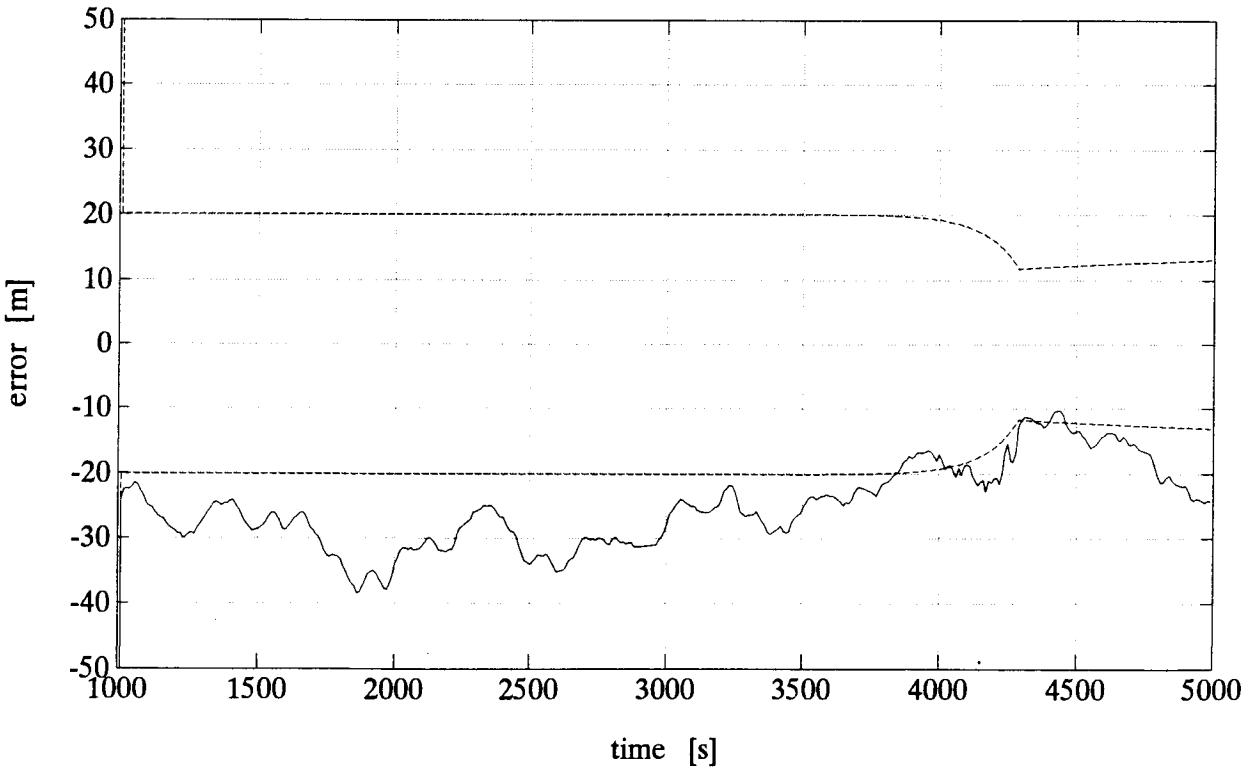
(2a-14) Error in y GPS Position Error Estimate



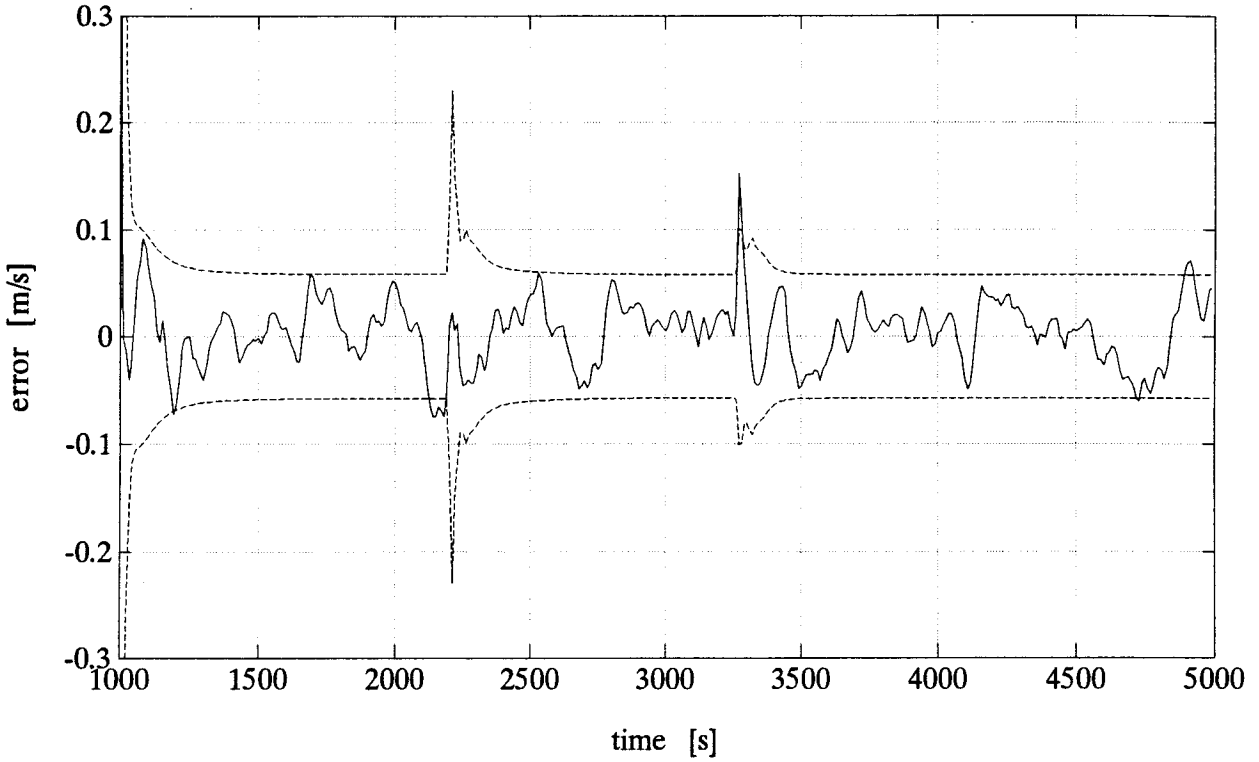
(2a-15) Error in Corrected x IMU Position



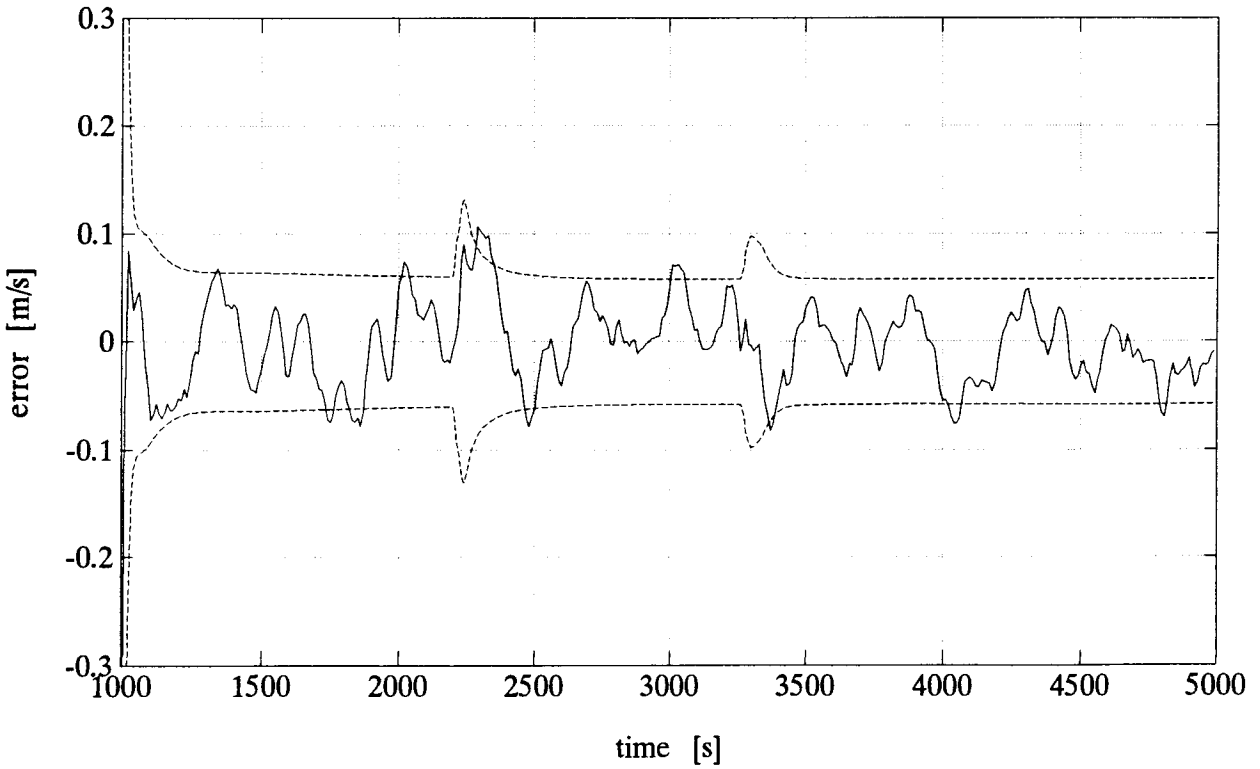
(2a-16) Error in Corrected y IMU Position



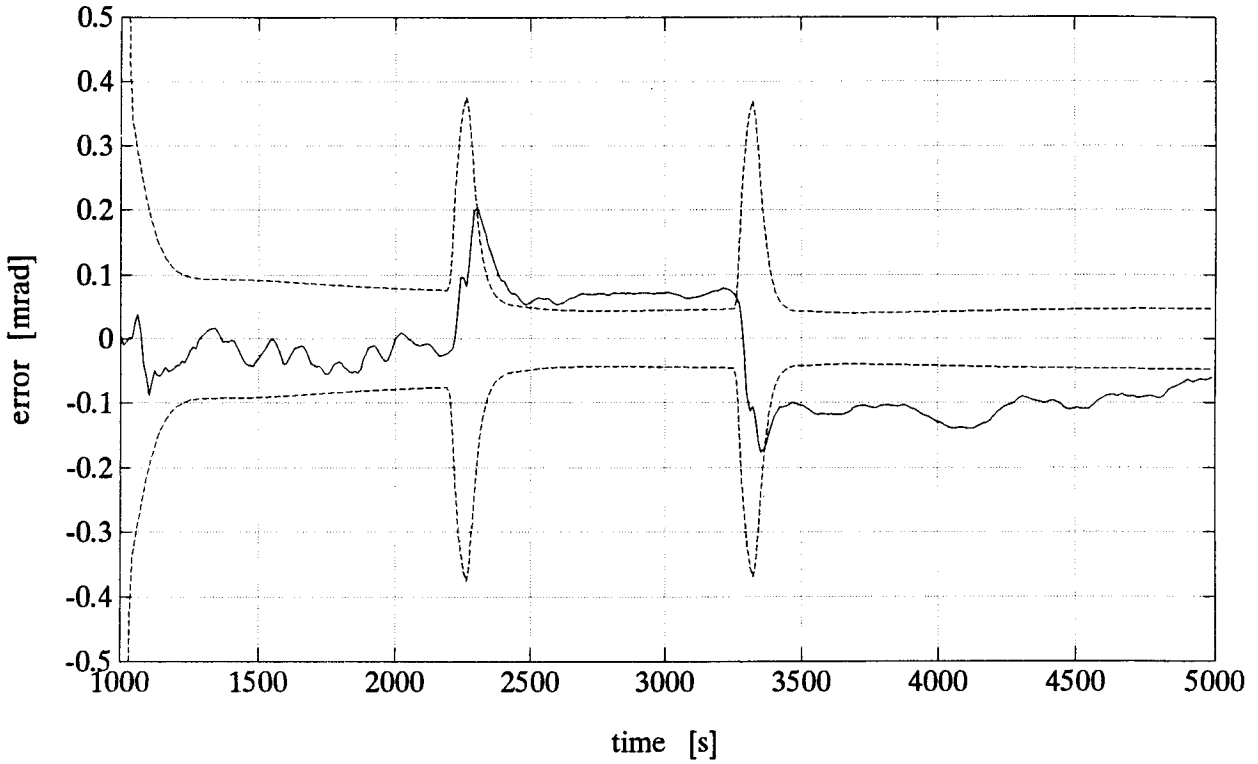
(2a-17) Error in Corrected x IMU Velocity



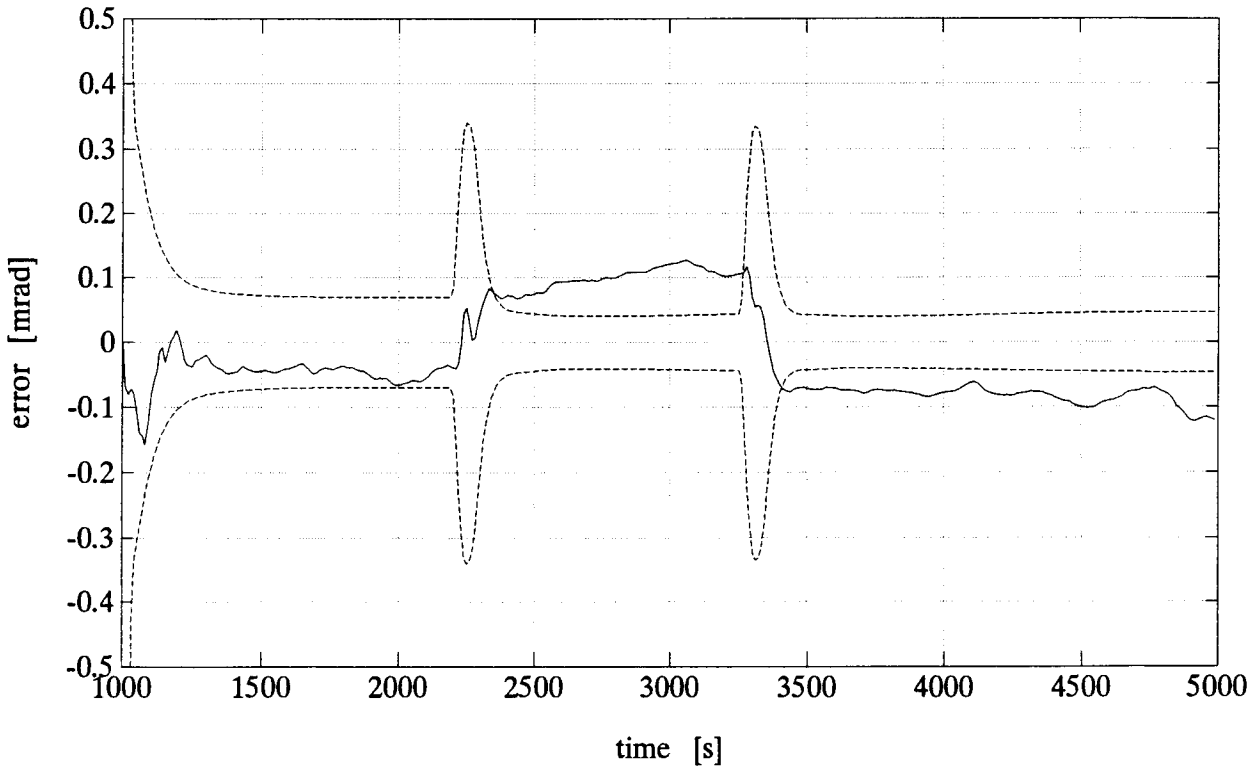
(2a-18) Error in Corrected y IMU Velocity



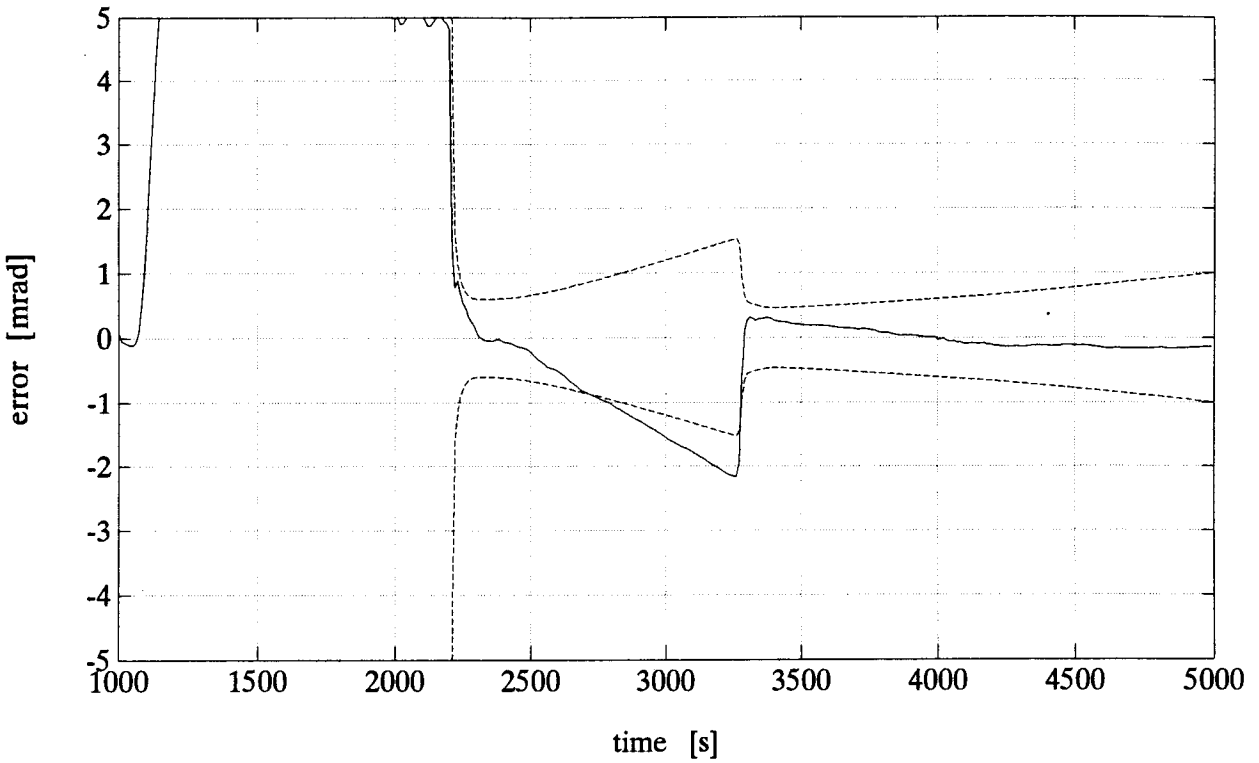
(2a-19) Error in Corrected x IMU Platform Alignment



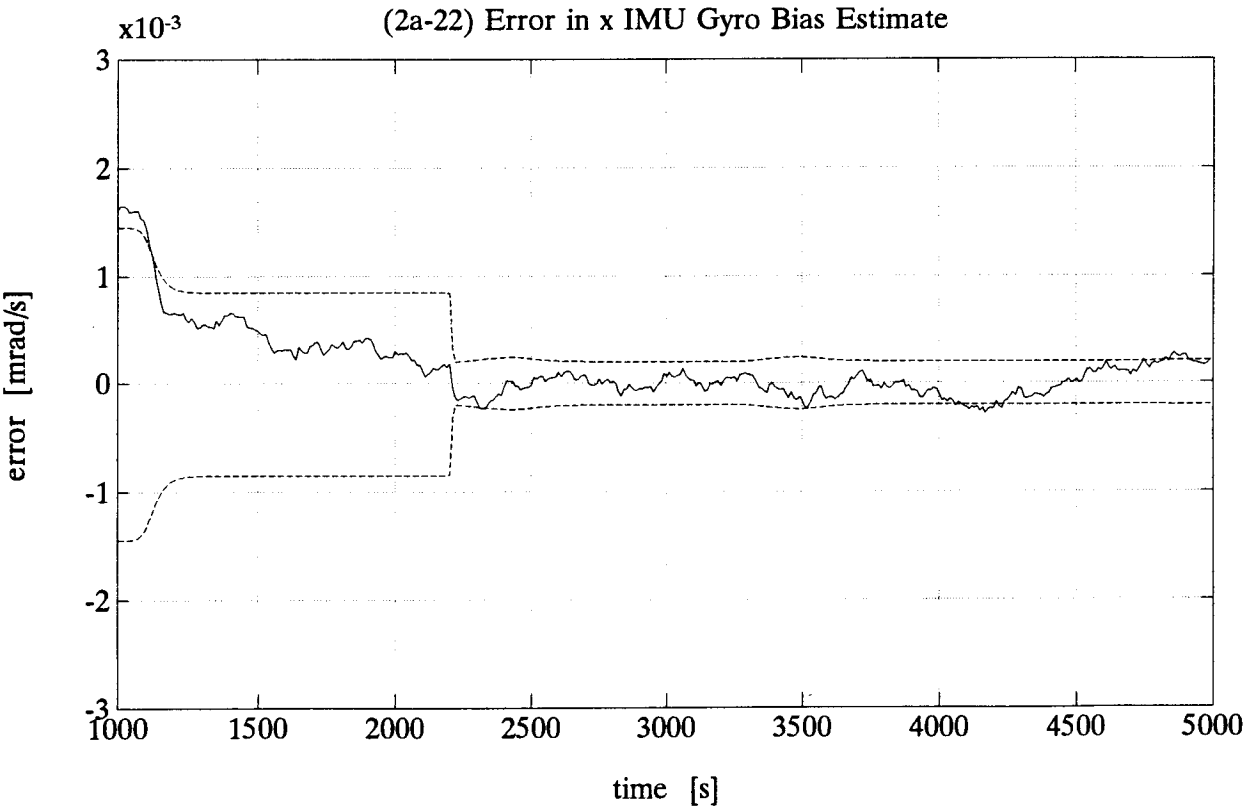
(2a-20) Error in Corrected y IMU Platform Alignment

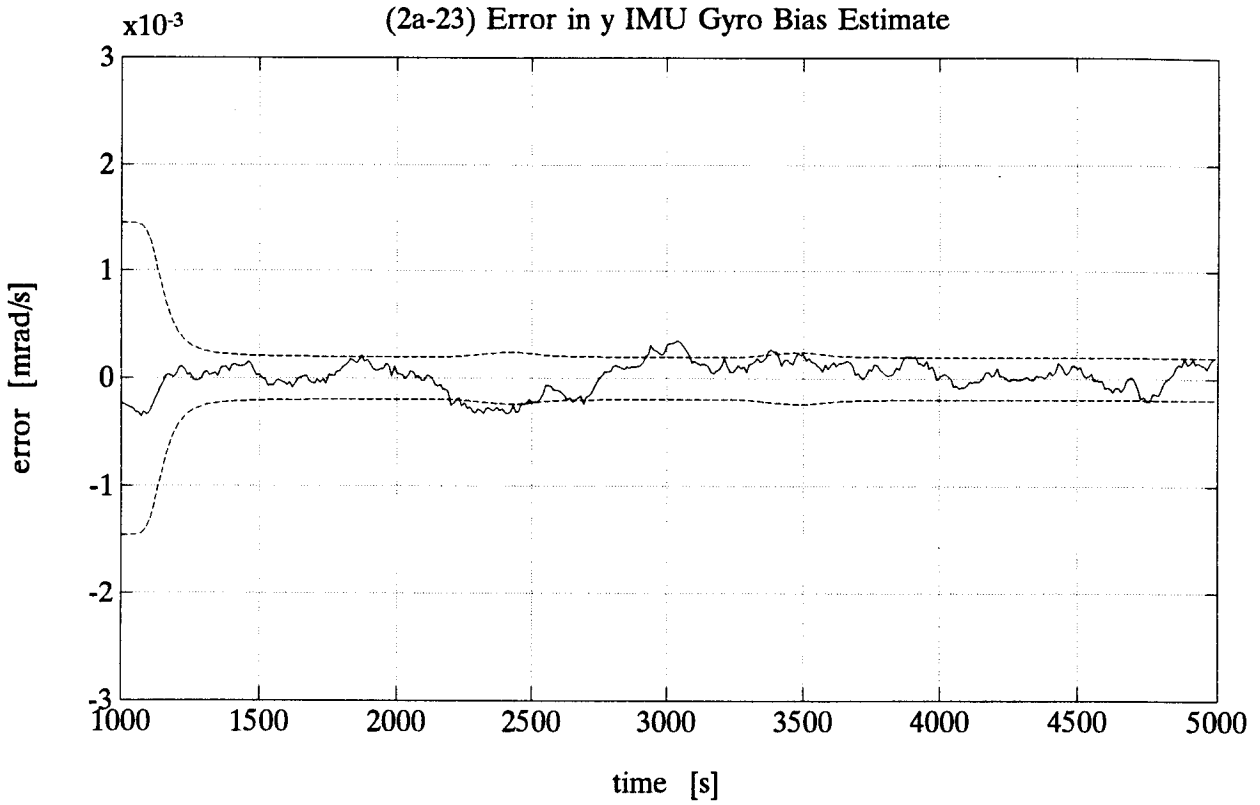


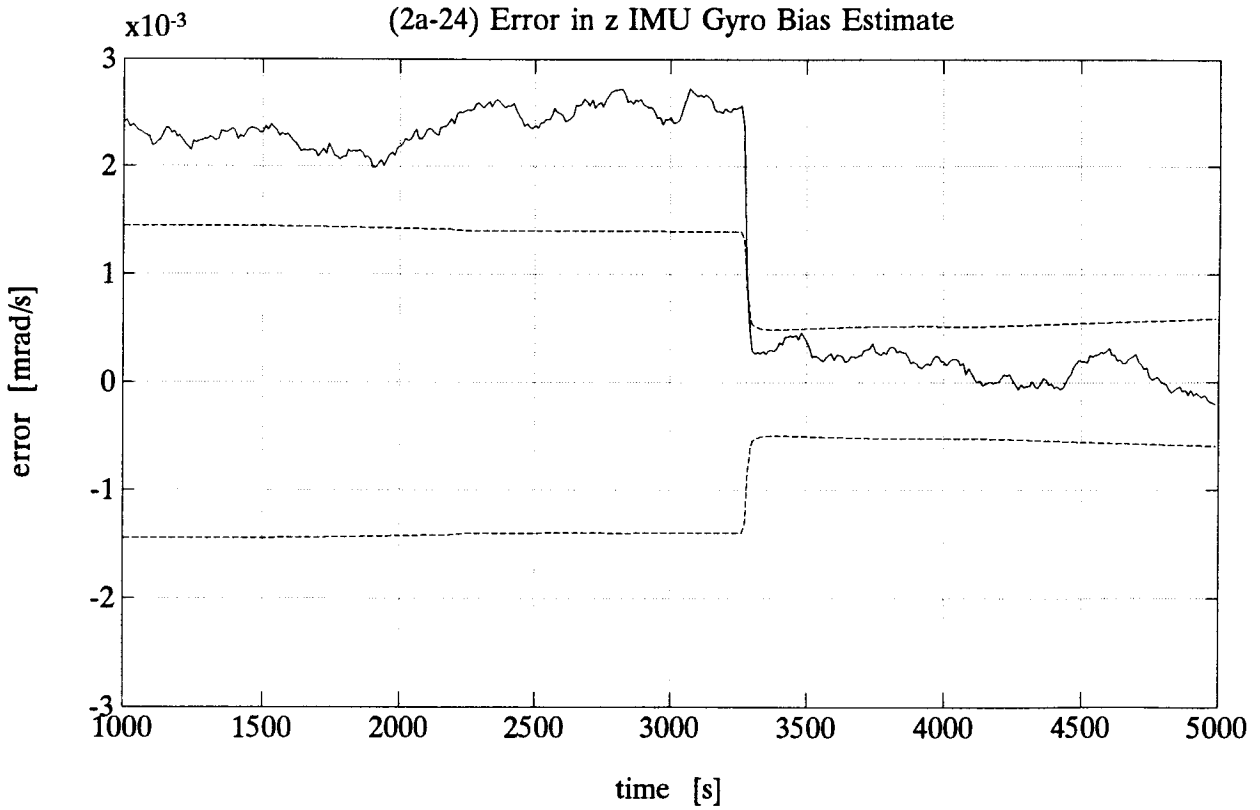
(2a-21) Error in Corrected z IMU Platform Alignment



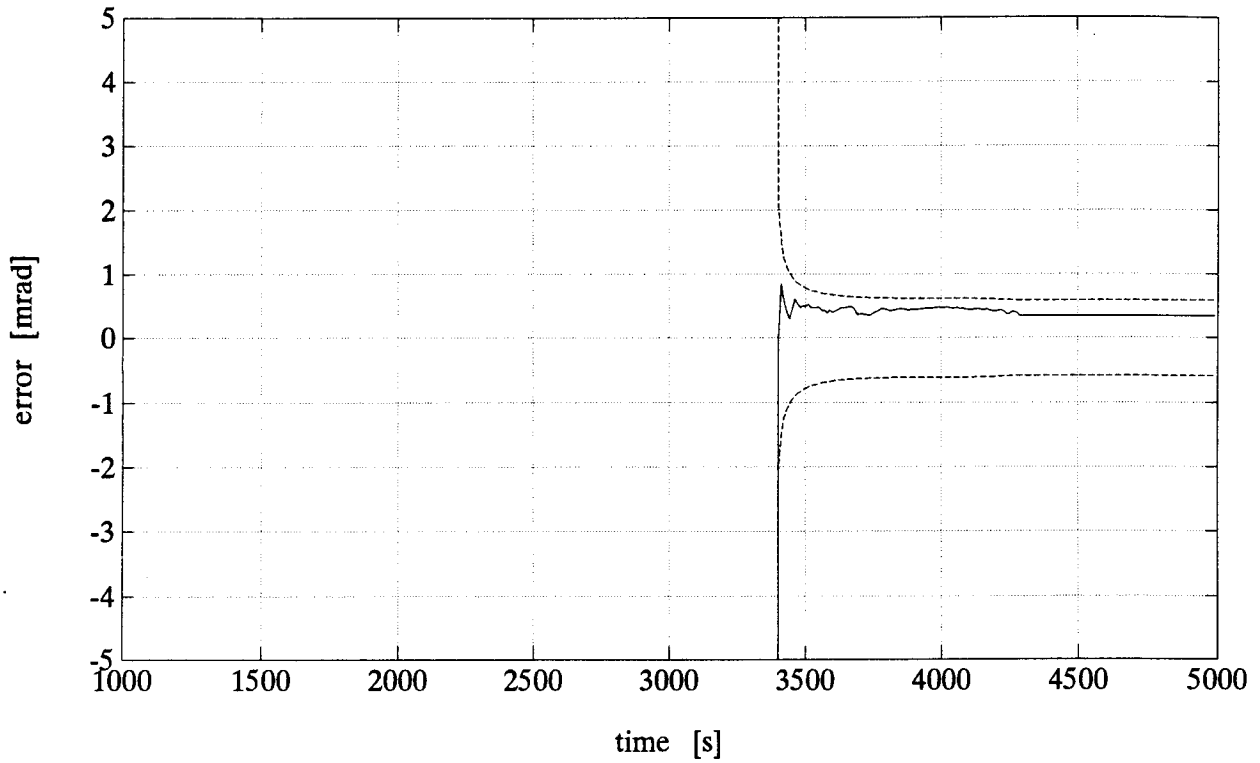
(2a-22) Error in x IMU Gyro Bias Estimate



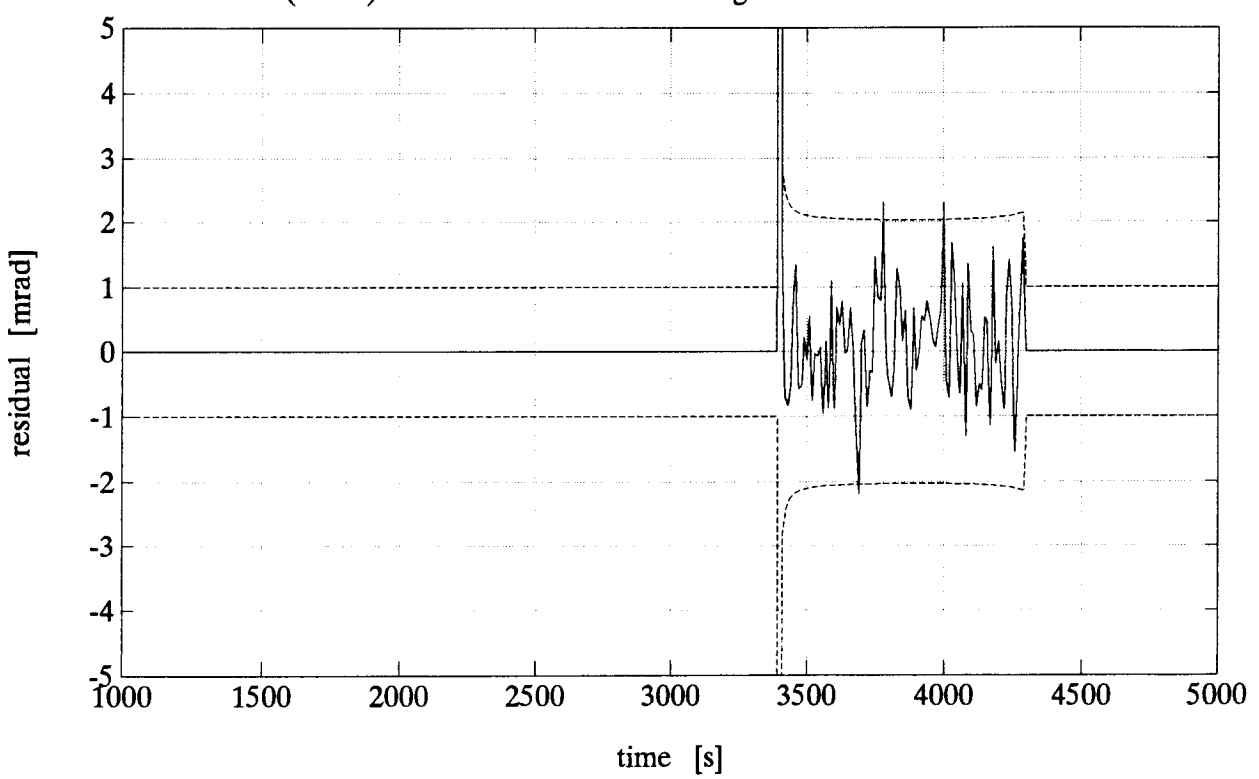

x10-3
(2a-23) Error in y IMU Gyro Bias Estimate
3
2
1
0
-1
-2
-3
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

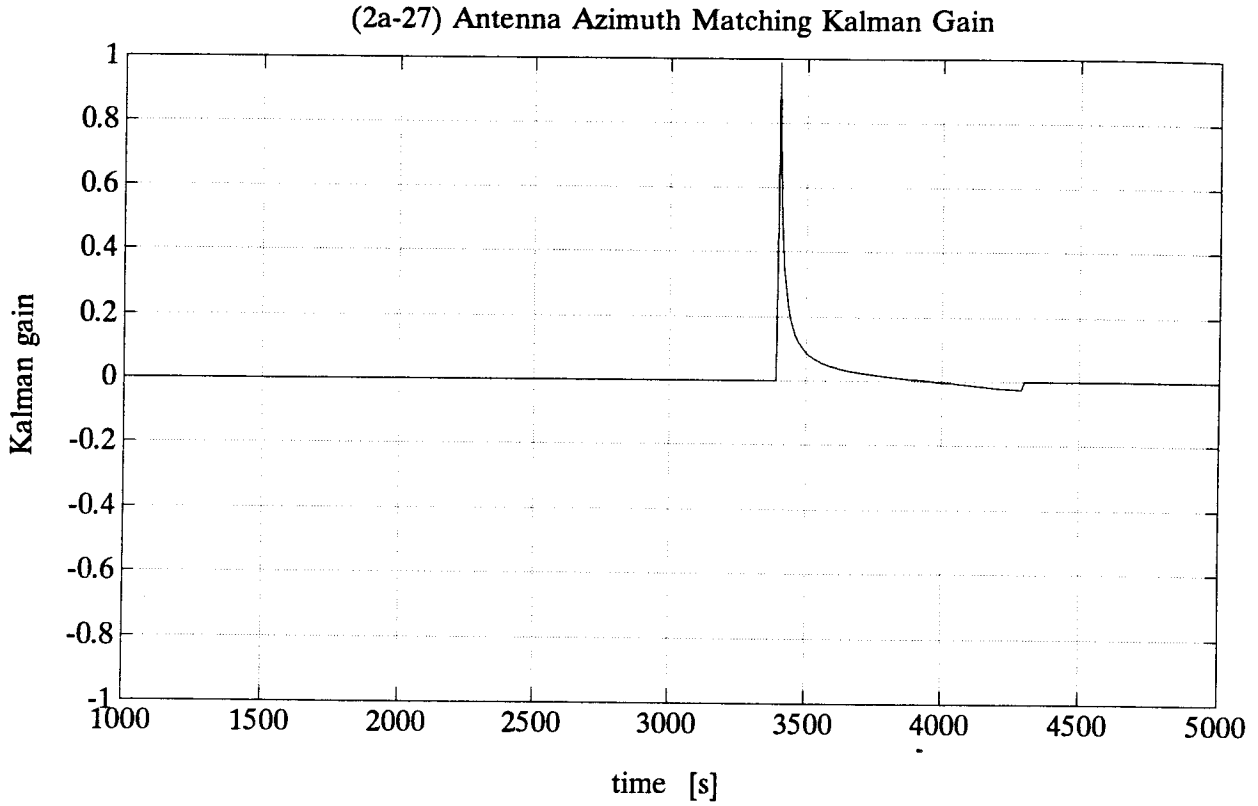

x10-3
(2a-24) Error in z IMU Gyro Bias Estimate
3
2
1
0
-1
-2
-3
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

(2a-25) Error in Antenna Azimuth Misalignment Estimate

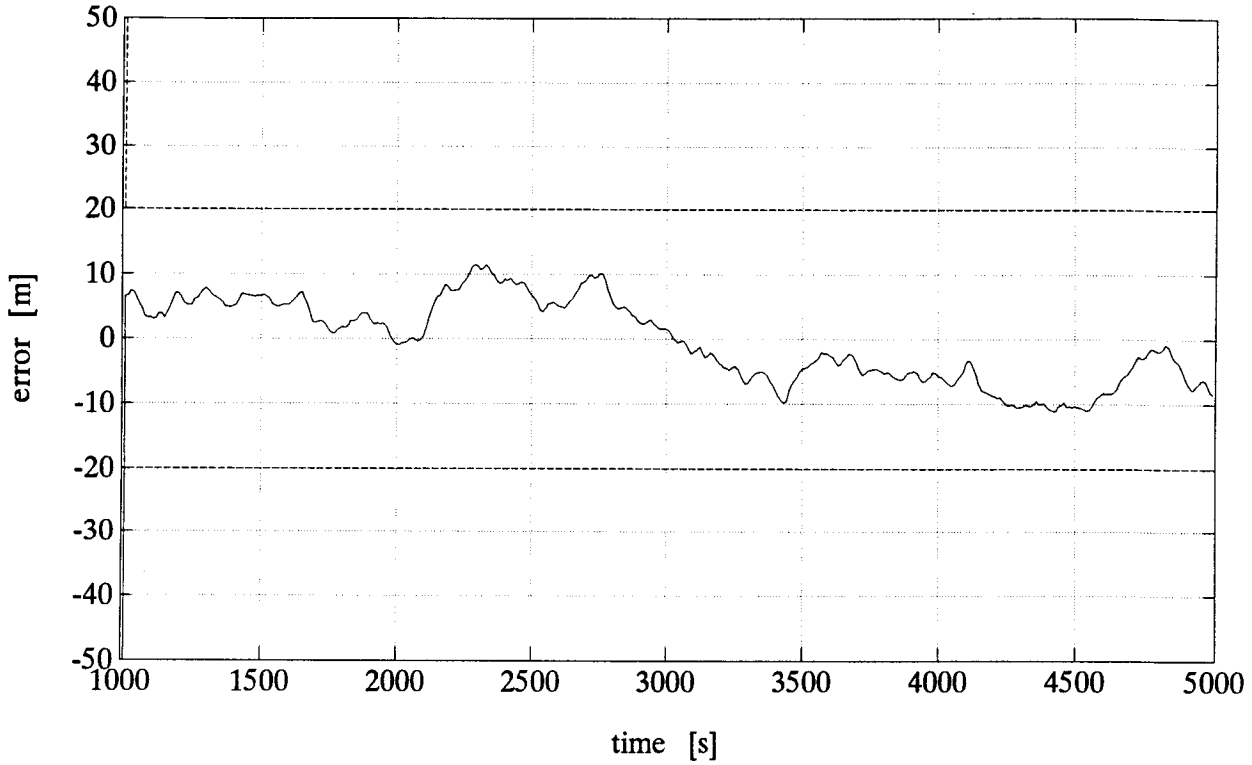


(2a-26) Antenna Azimuth Matching Measurement Residual

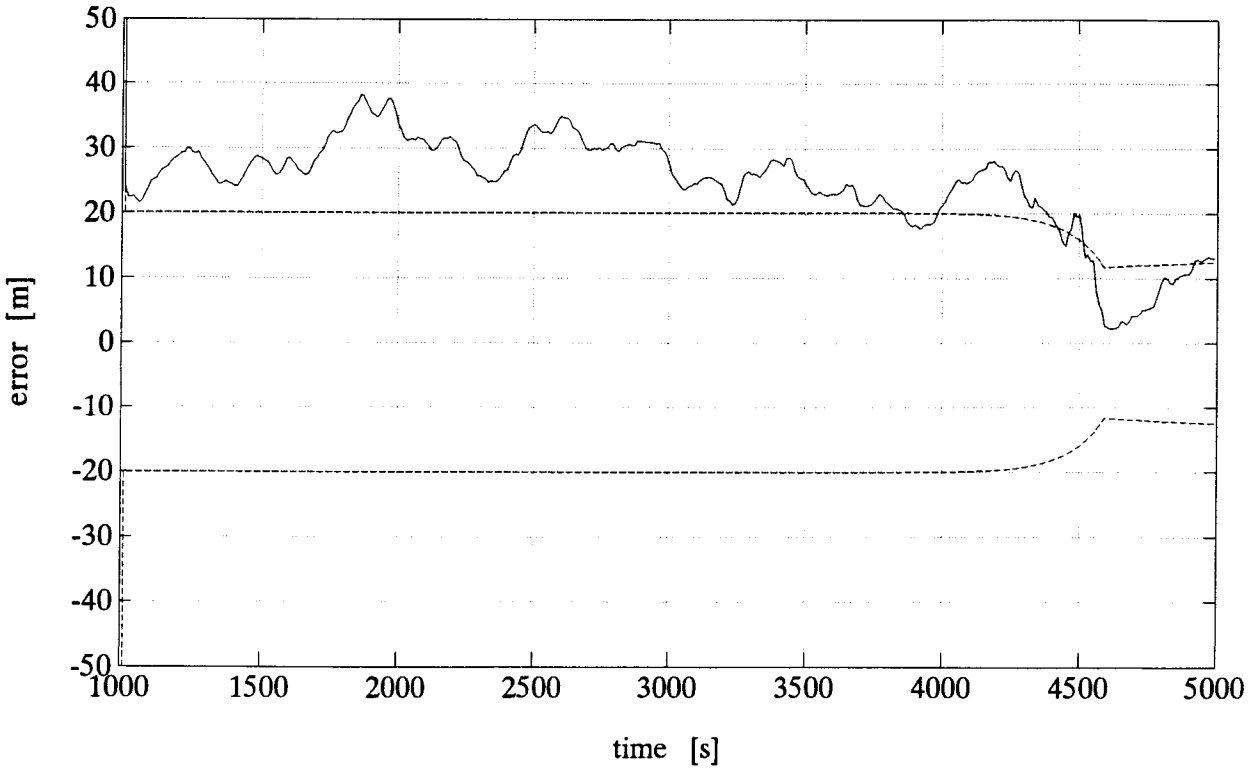



(2a-27) Antenna Azimuth Matching Kalman Gain
Kalman gain
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

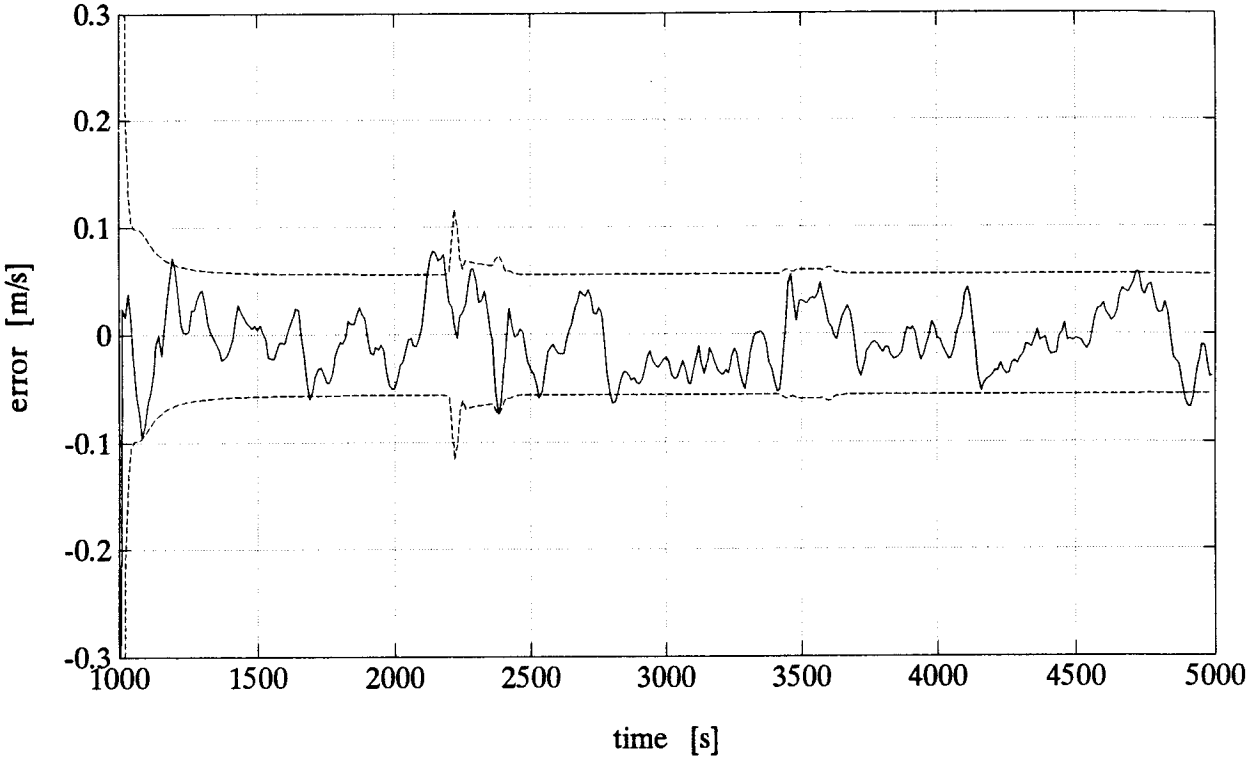
(3a-1) Error in x INS Position Error Estimate



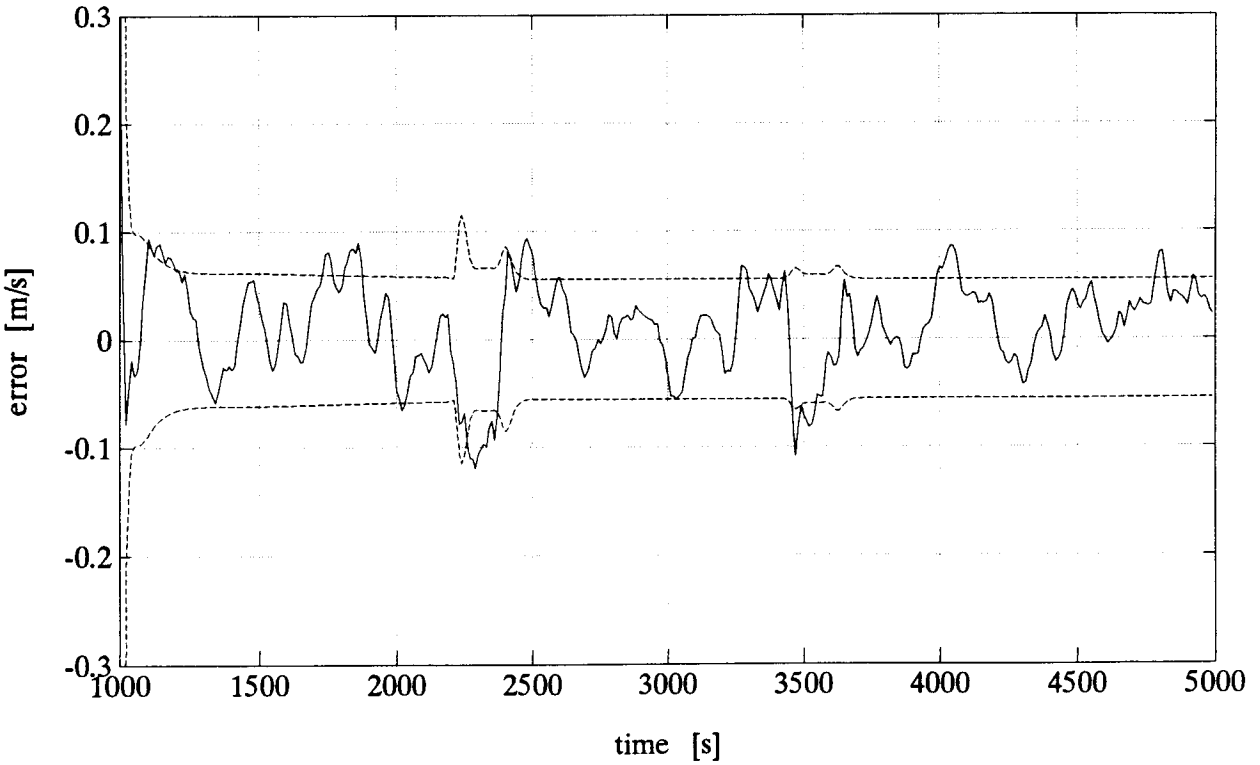
(3a-2) Error in y INS Position Error Estimate



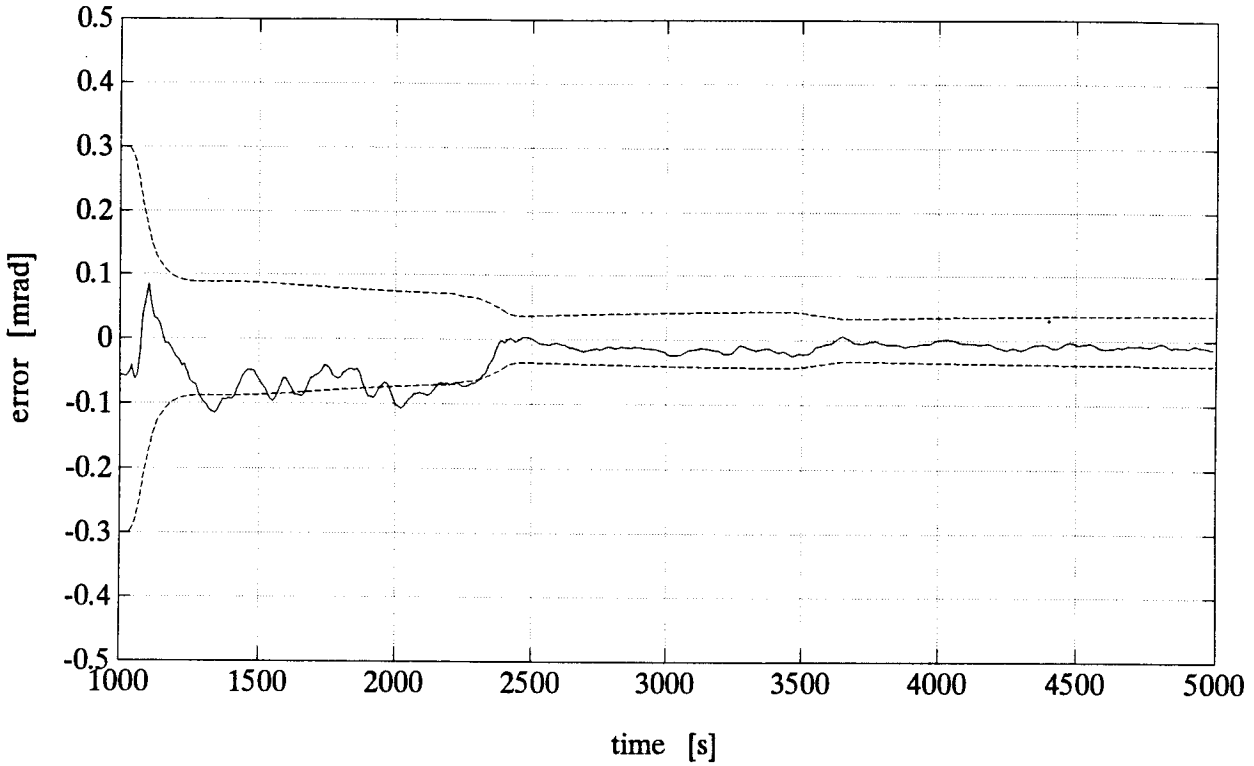
(3a-3) Error in x INS Velocity Error Estimate



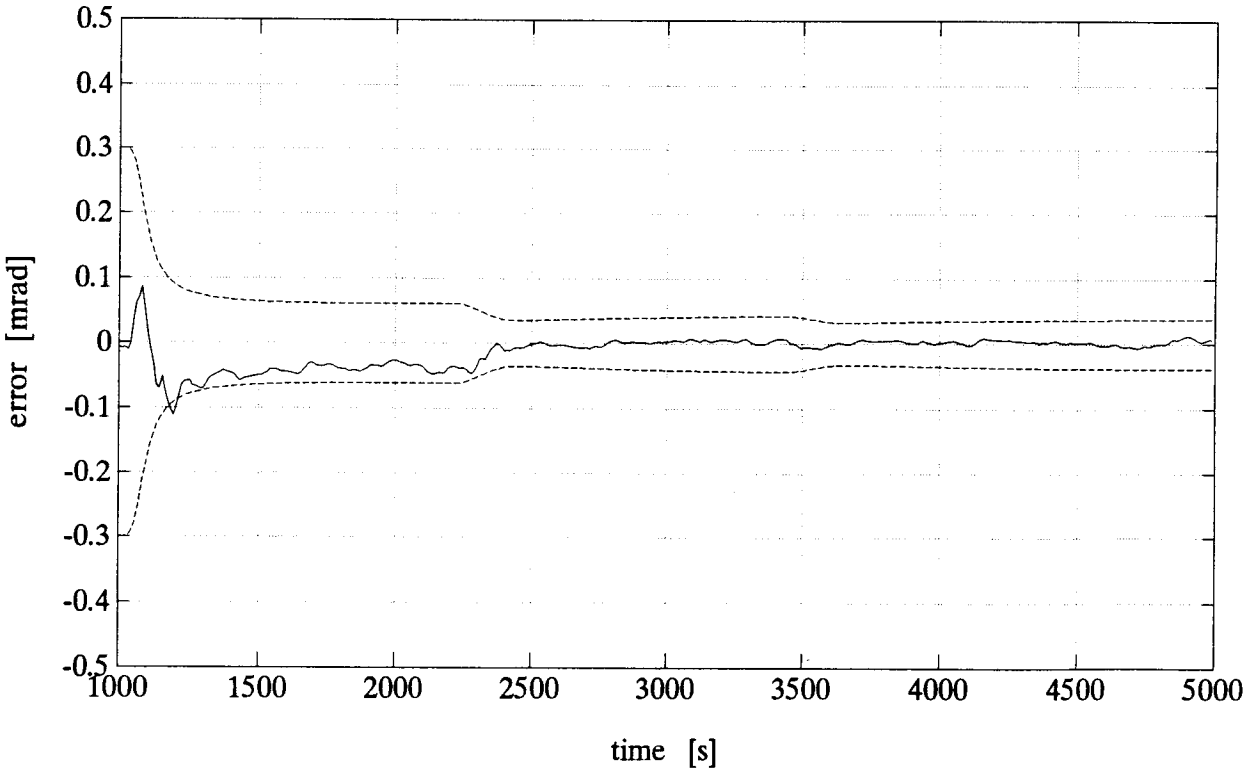
(3a-4) Error in y INS Velocity Error Estimate



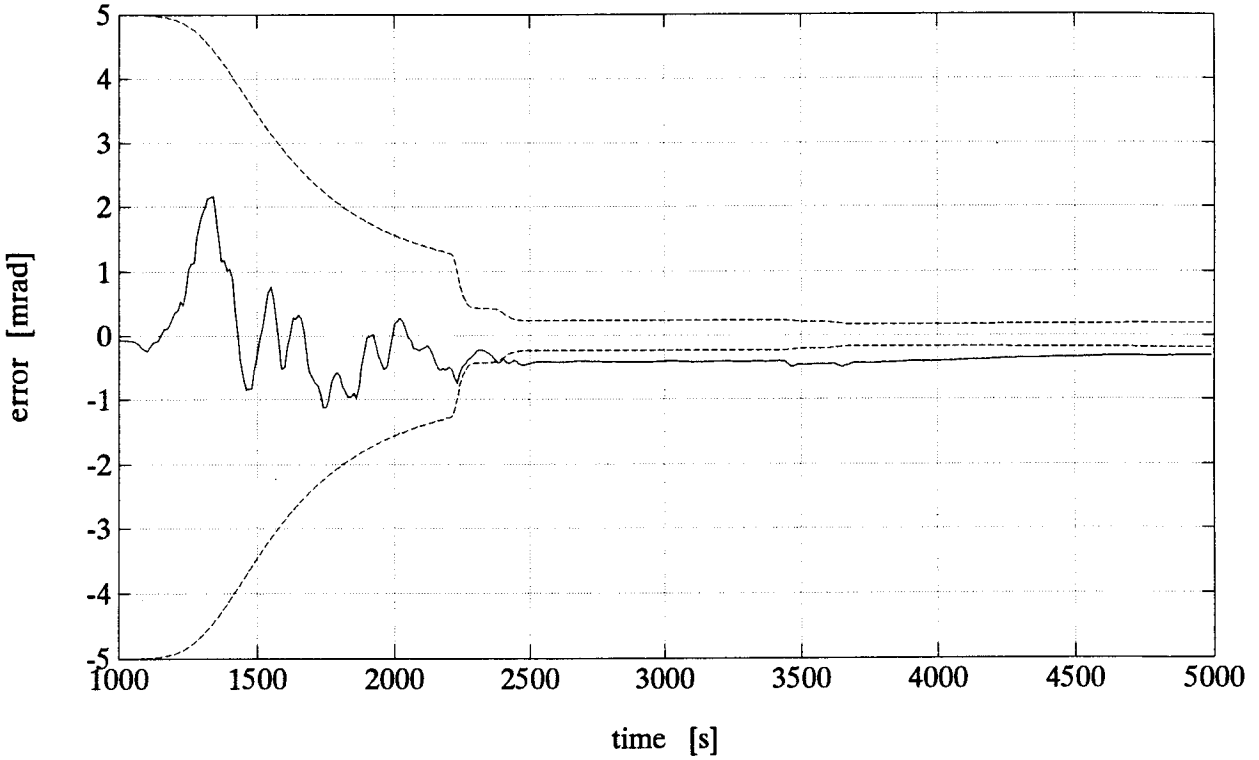
(3a-5) Error in x INS Platform Misalignment Estimate



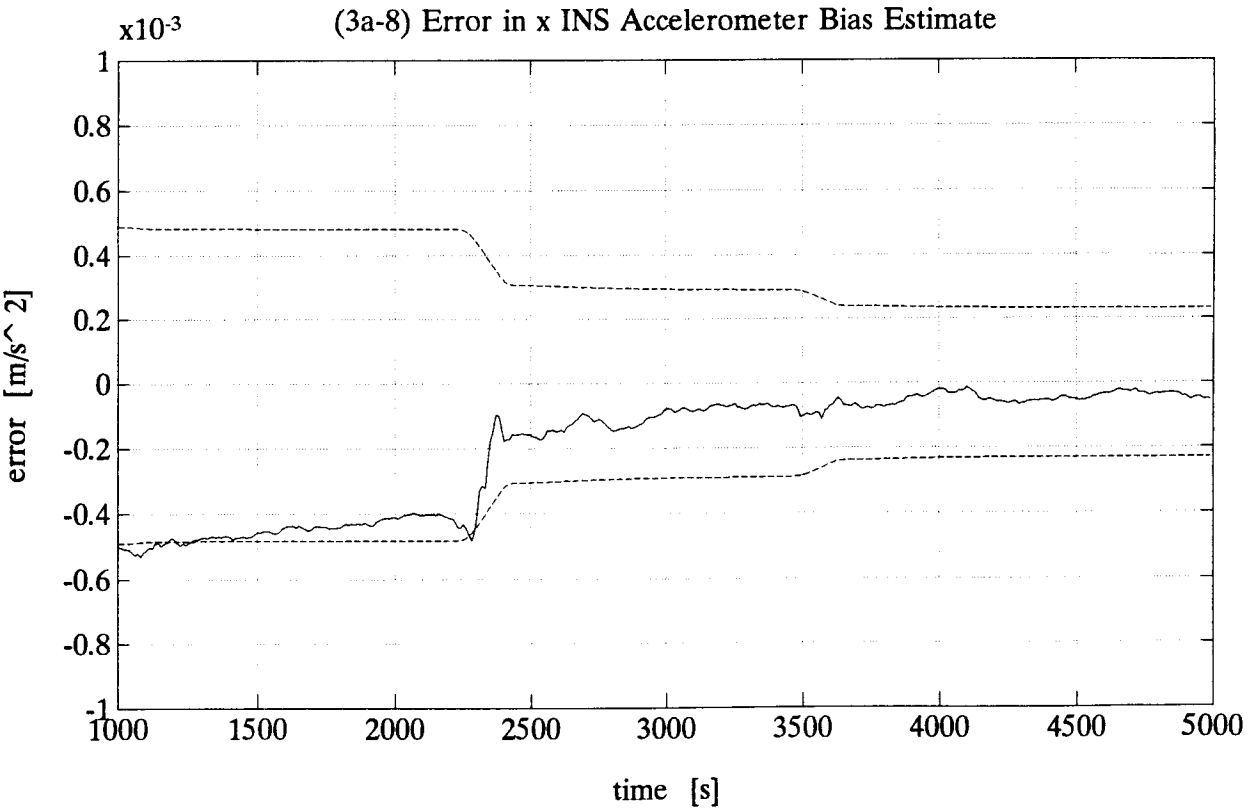
(3a-6) Error in y INS Platform Misalignment Estimate

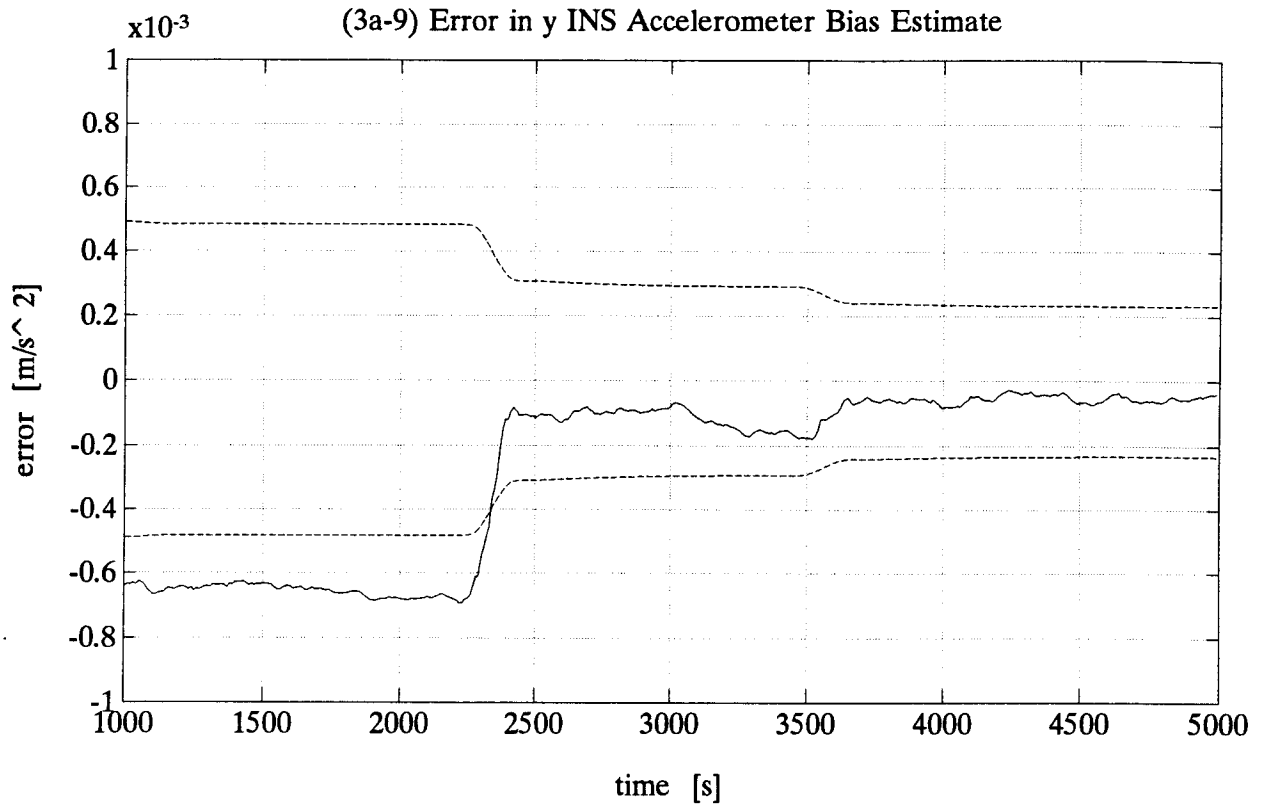


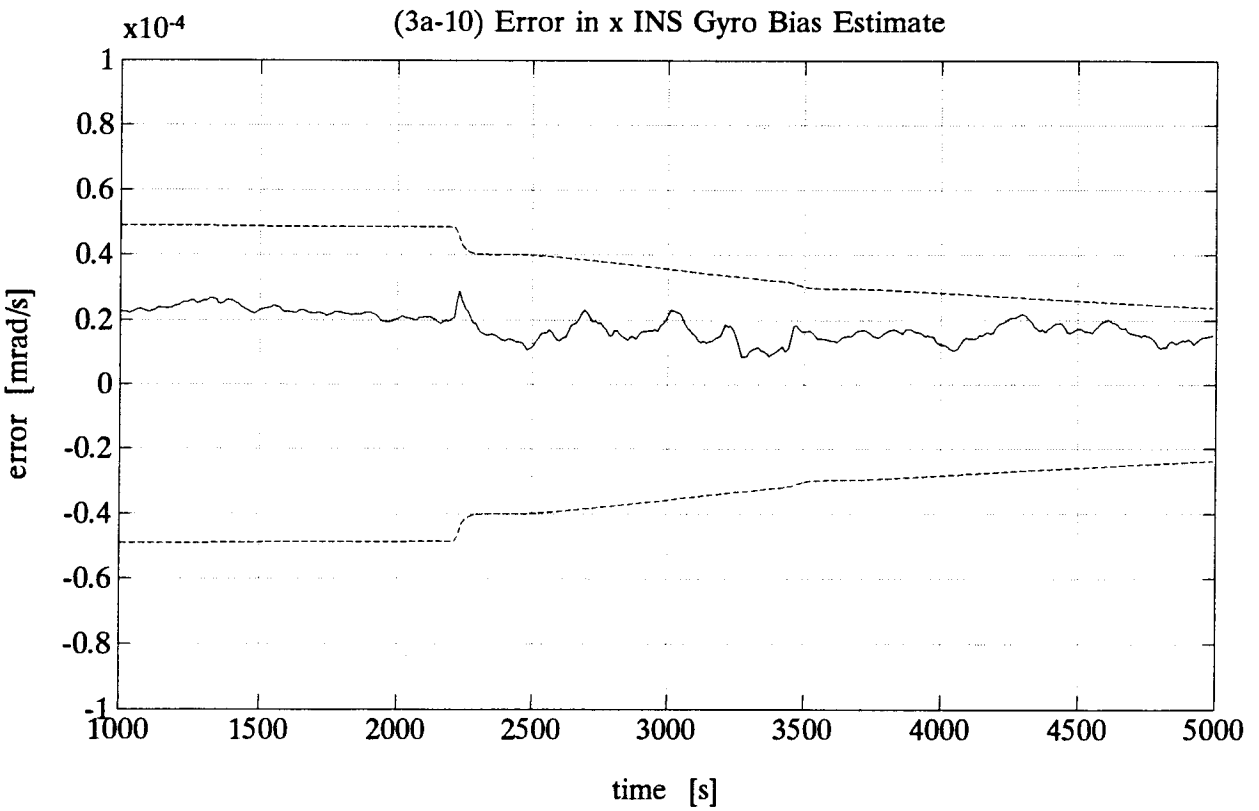
(3a-7) Error in z INS Platform Misalignment Estimate

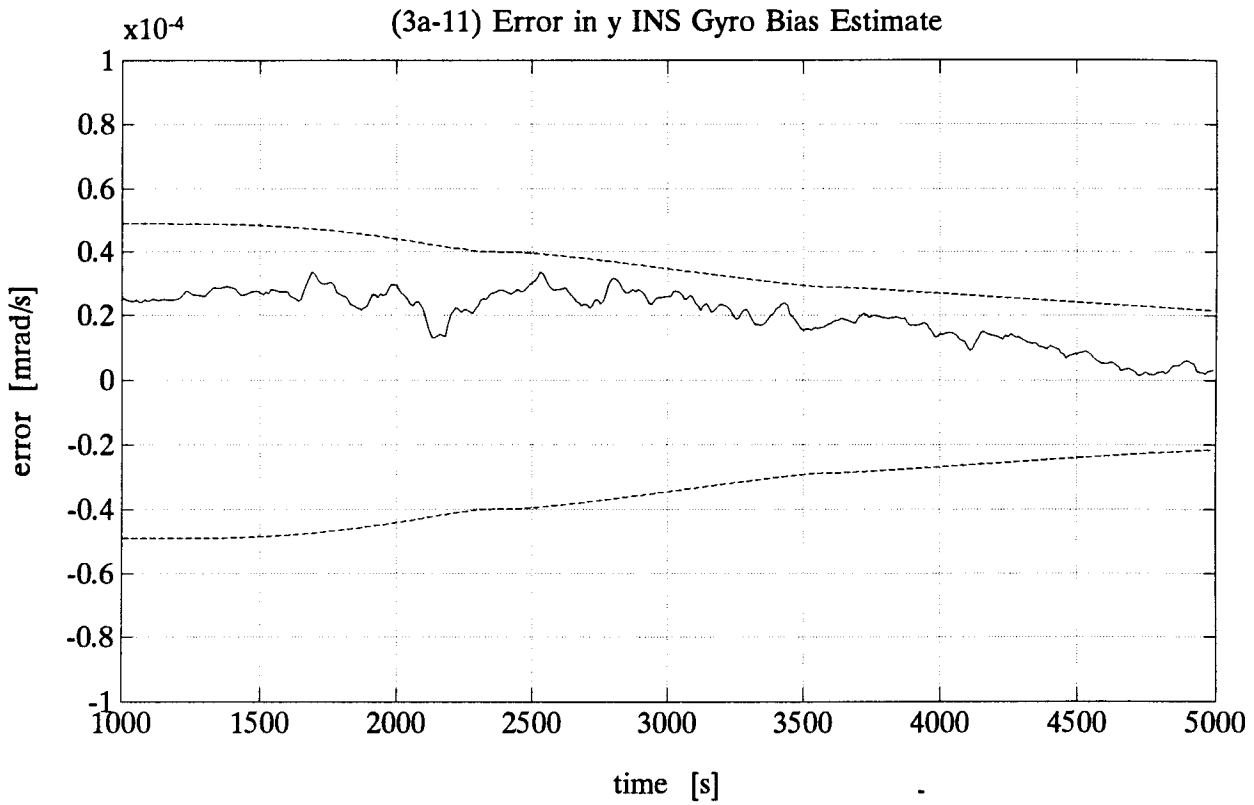


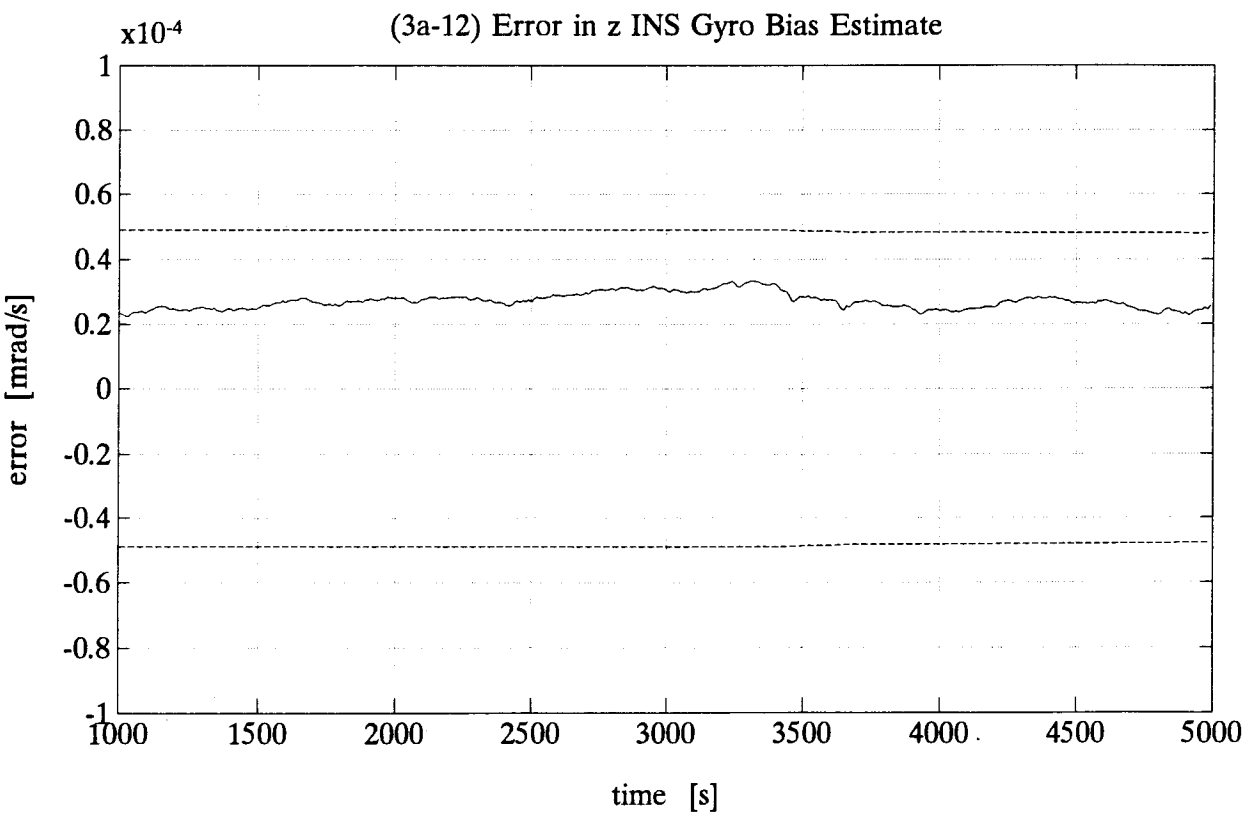
(3a-8) Error in x INS Accelerometer Bias Estimate



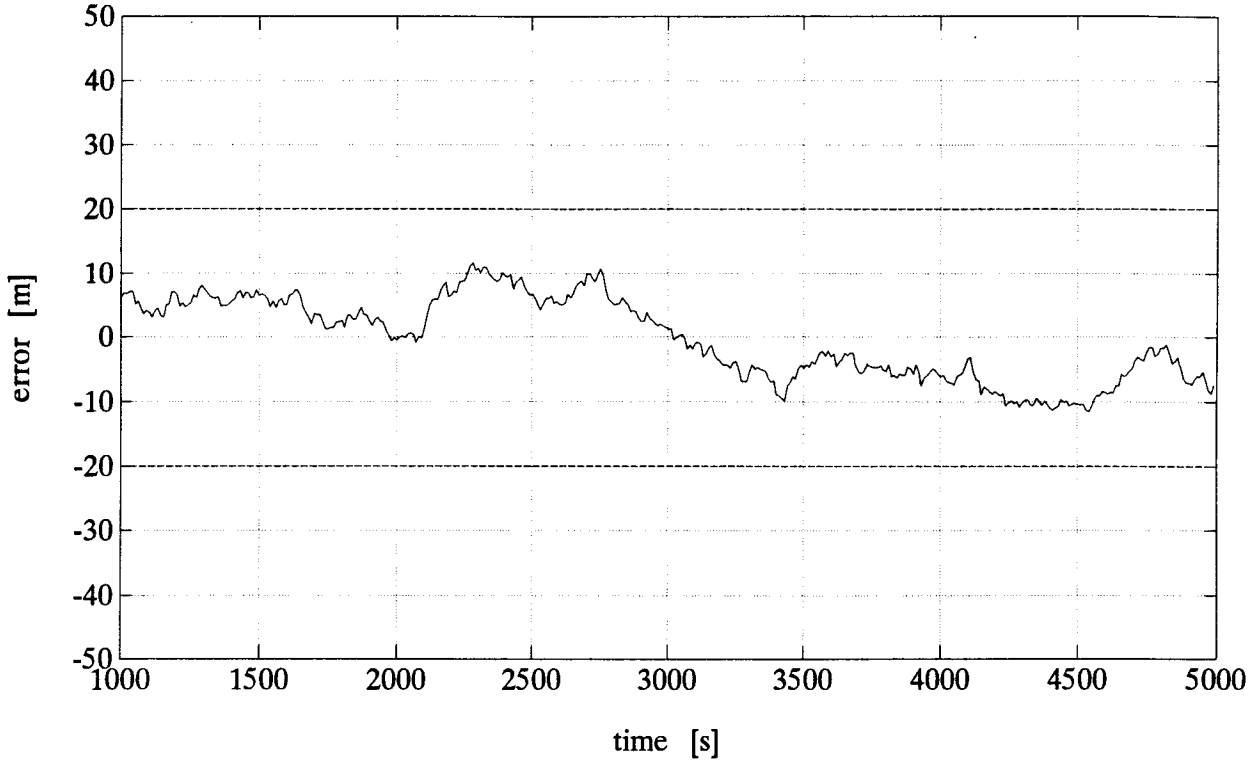

x10-3
(3a-9) Error in y INS Accelerometer Bias Estimate
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [m/s^ 2]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


x10-4
(3a-10) Error in x INS Gyro Bias Estimate
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

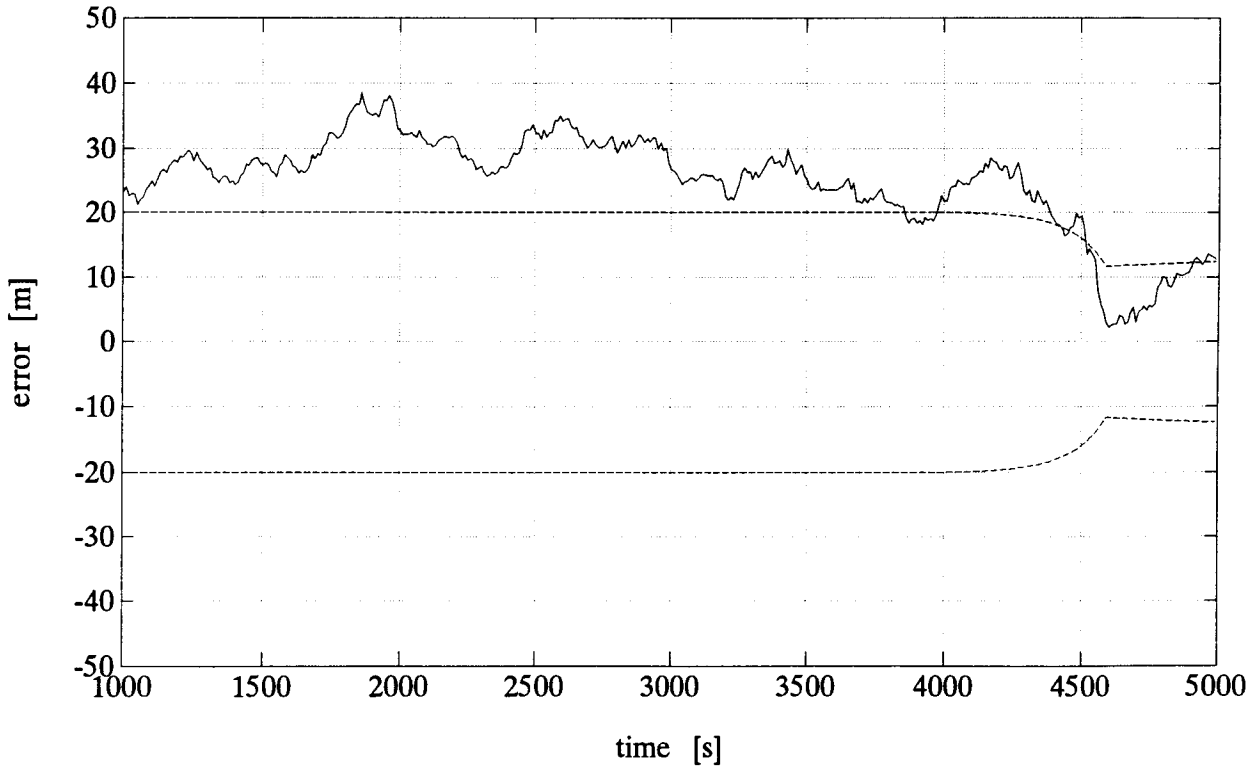

(3a-11) Error in y INS Gyro Bias Estimate
x10-4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]


(3a-12) Error in z INS Gyro Bias Estimate
x10-4
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

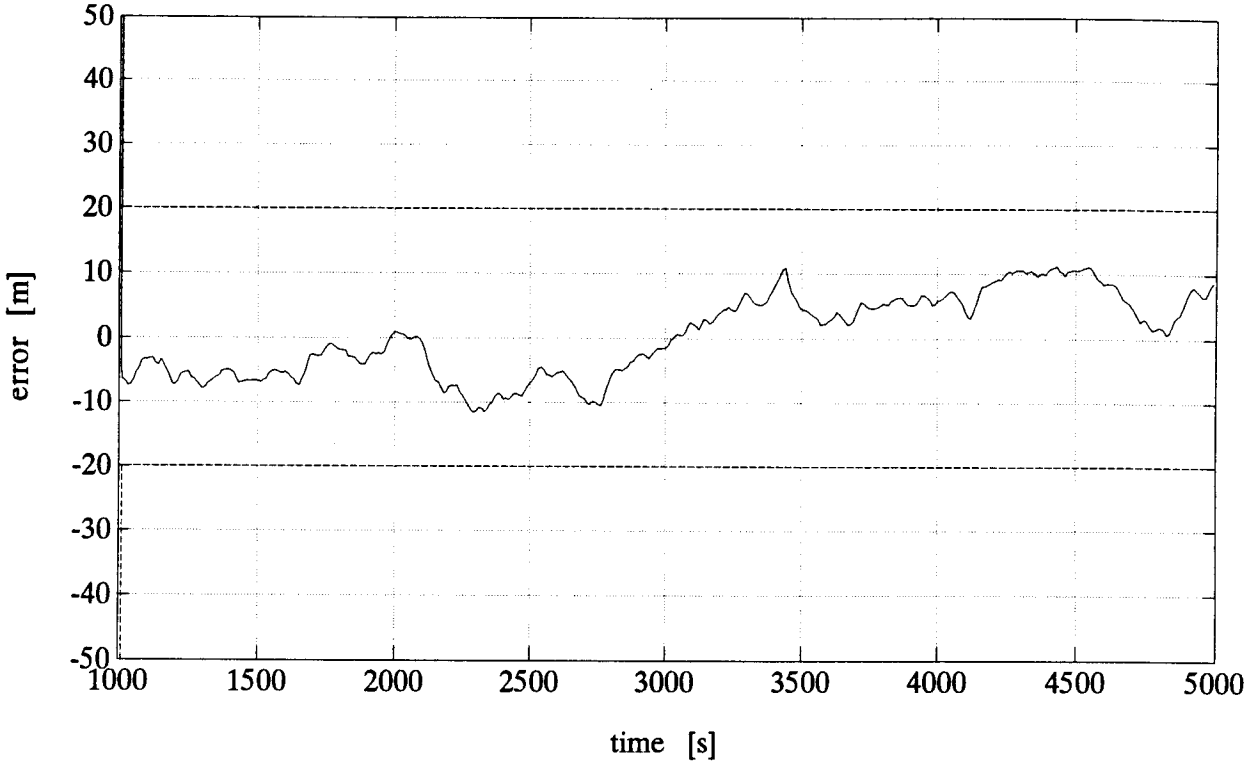
(3a-13) Error in x GPS Position Error Estimate



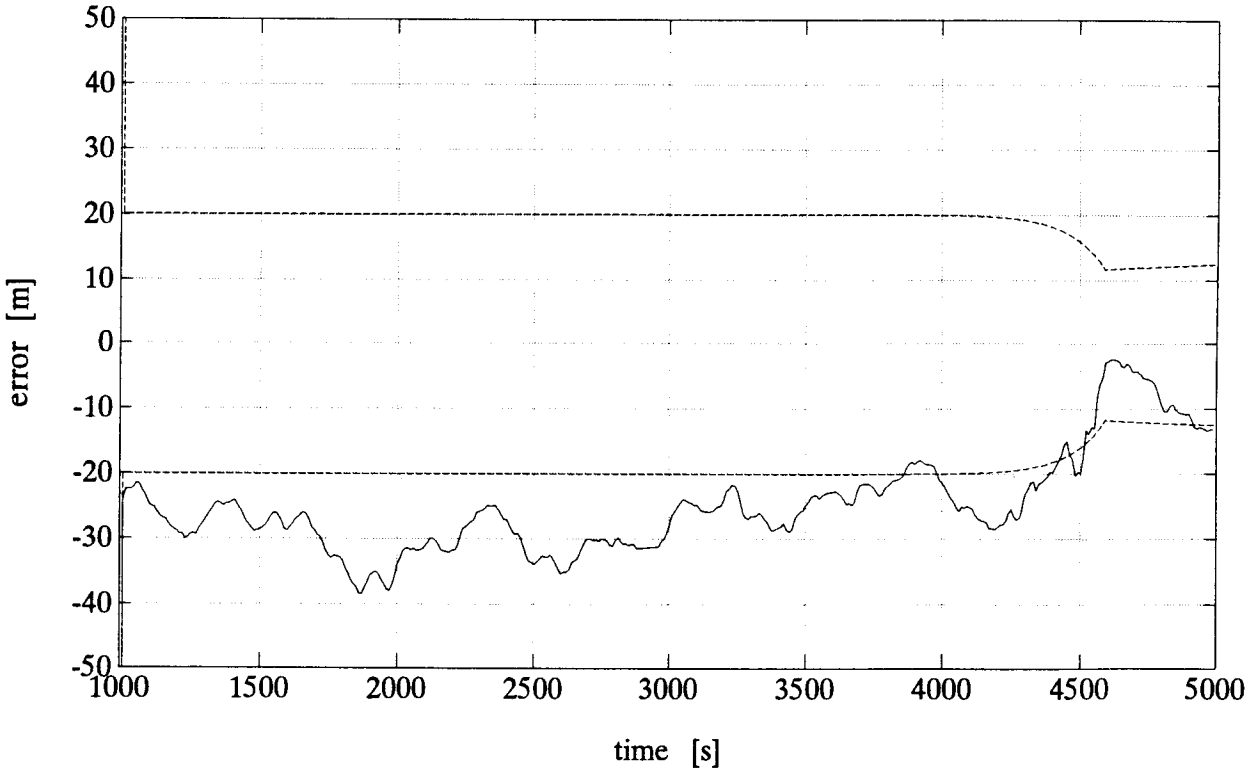
(3a-14) Error in y GPS Position Error Estimate



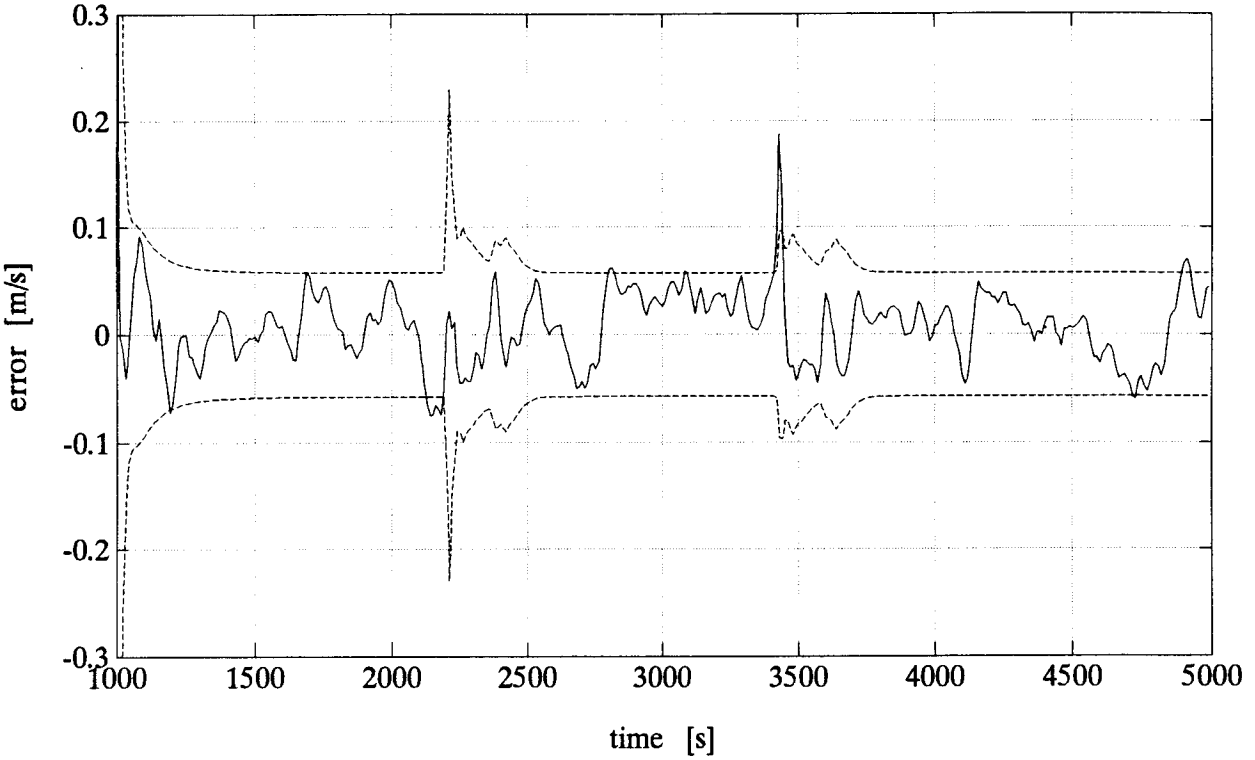
(3a-15) Error in Corrected x IMU Position



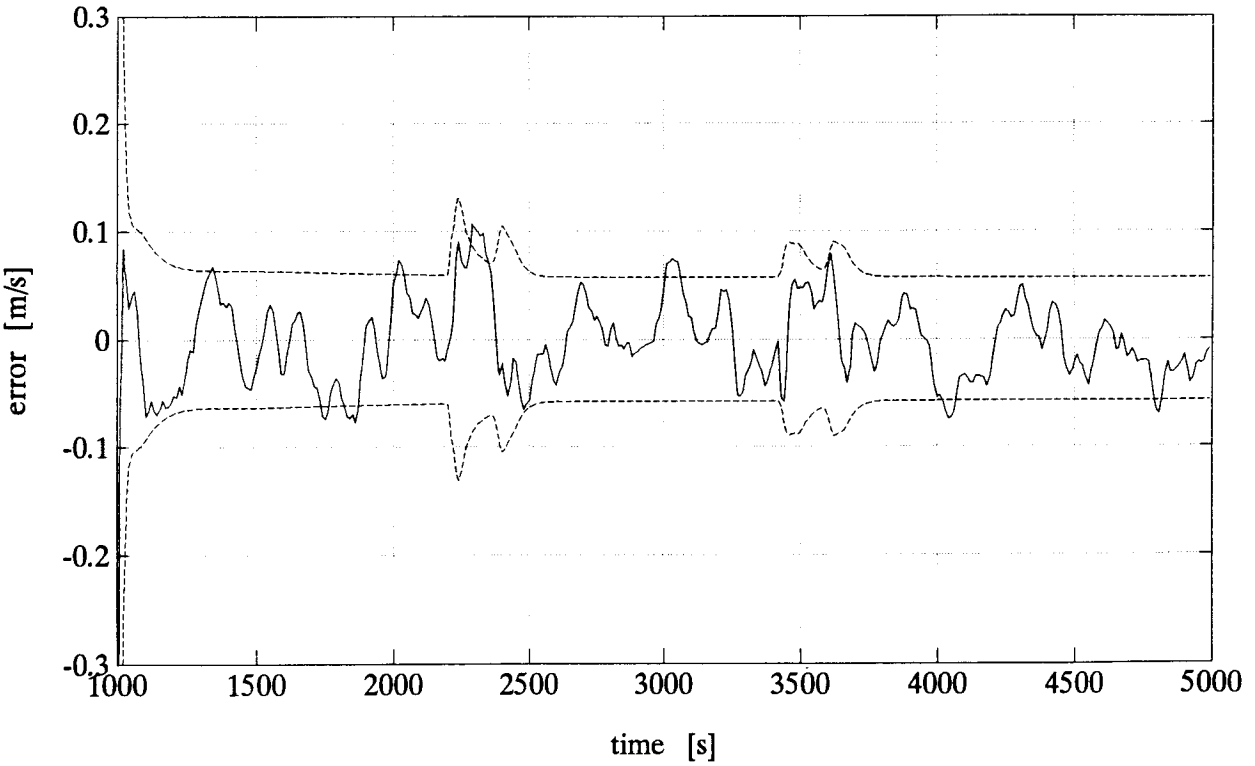
(3a-16) Error in Corrected y IMU Position



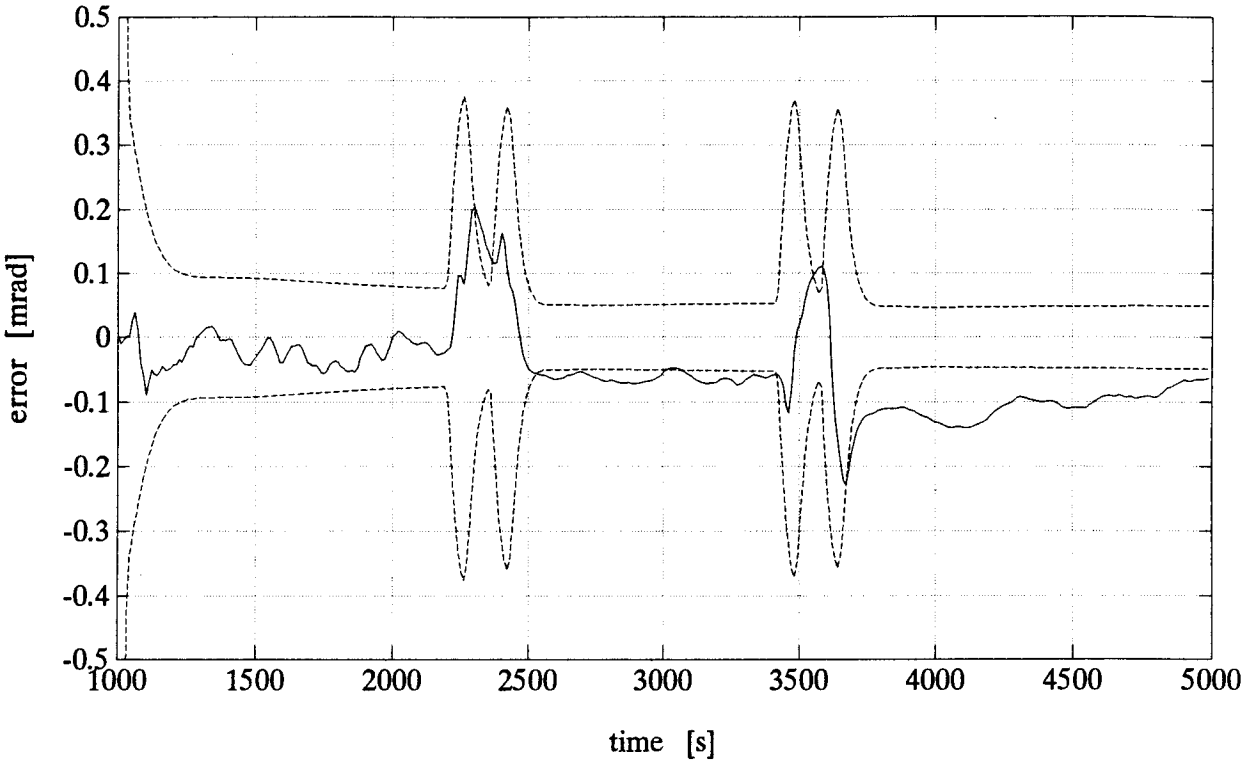
(3a-17) Error in Corrected x IMU Velocity



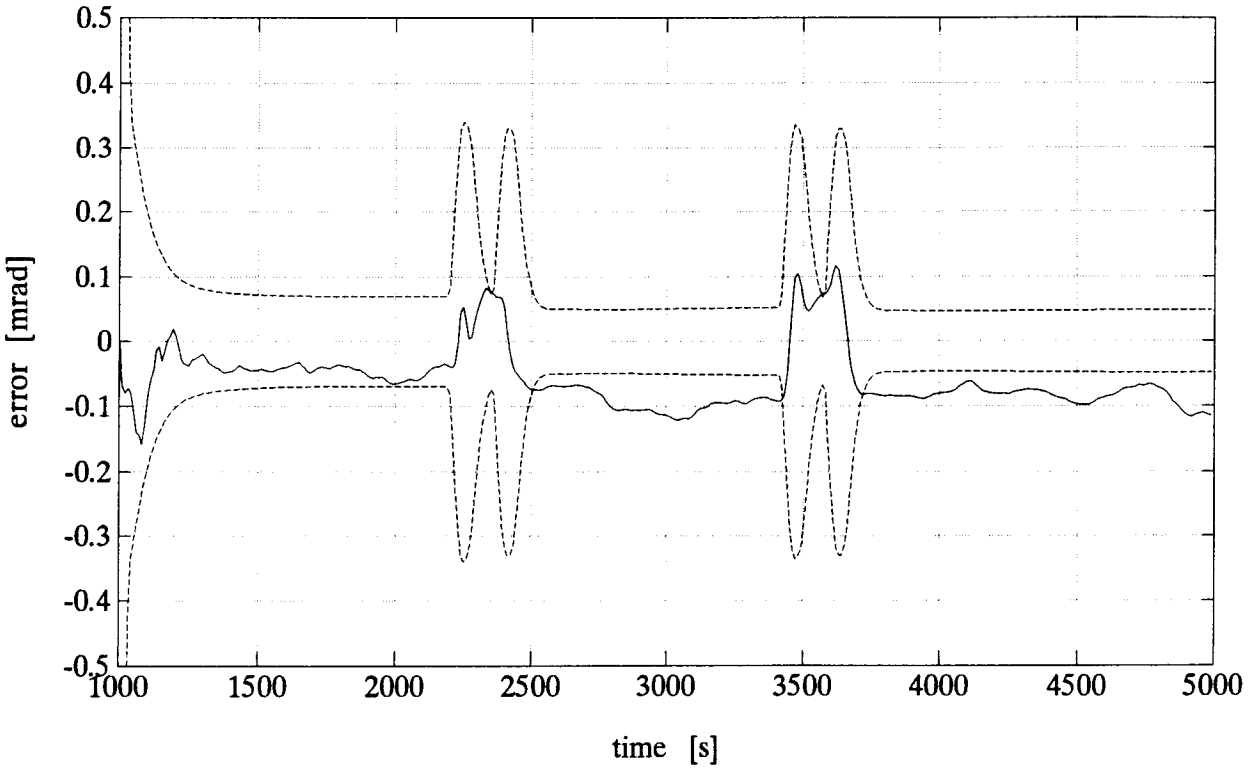
(3a-18) Error in Corrected y IMU Velocity



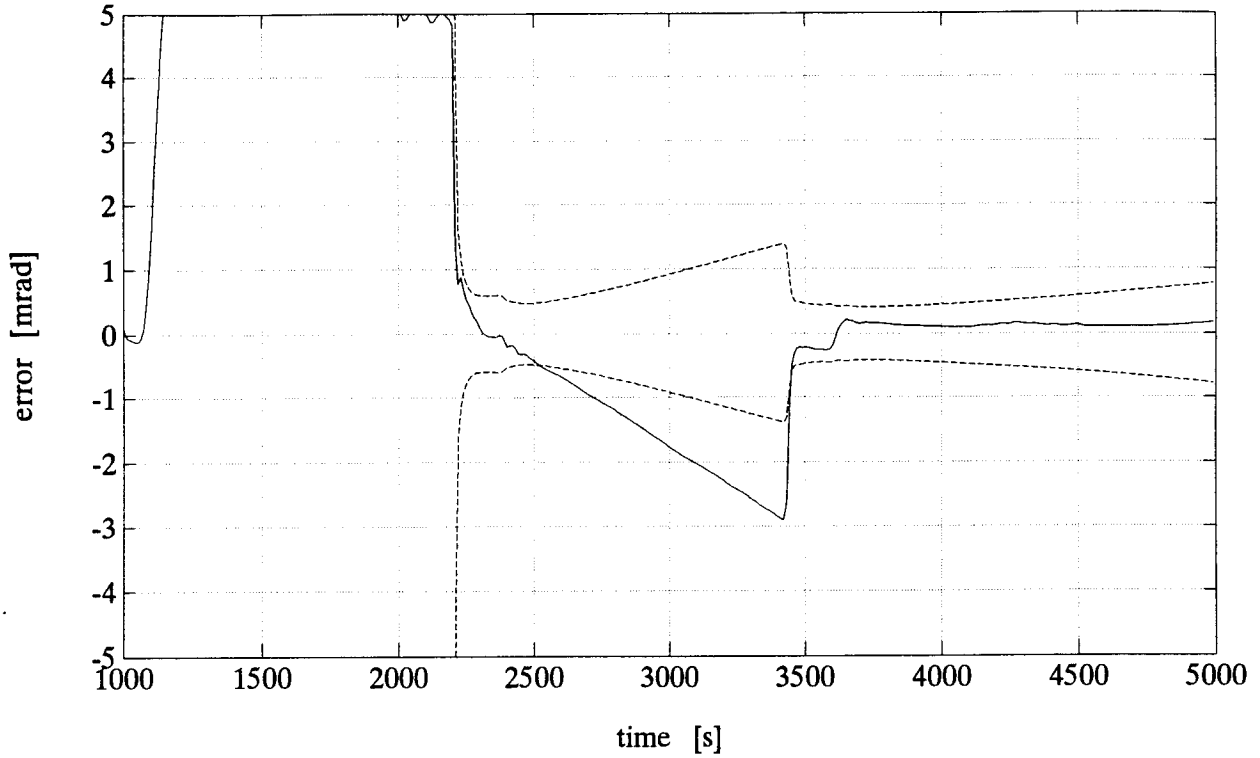
(3a-19) Error in Corrected x IMU Platform Alignment



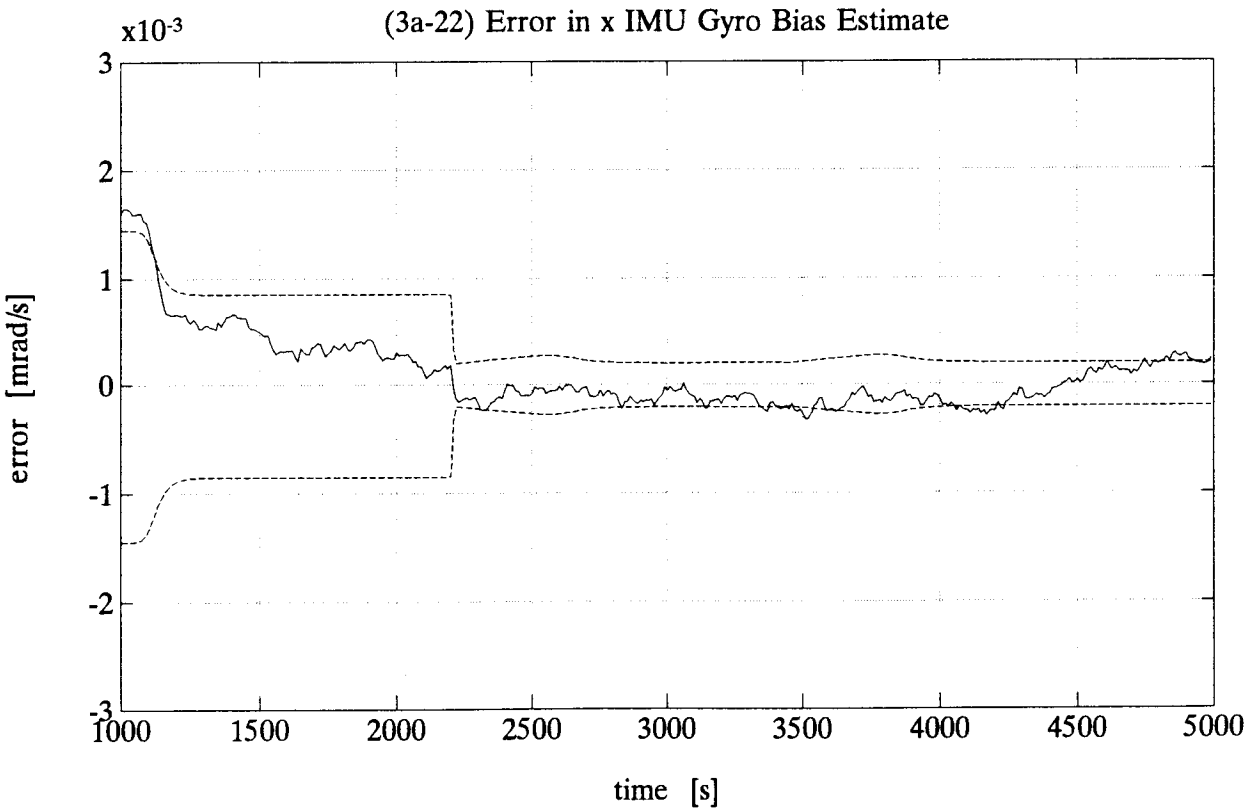
(3a-20) Error in Corrected y IMU Platform Alignment

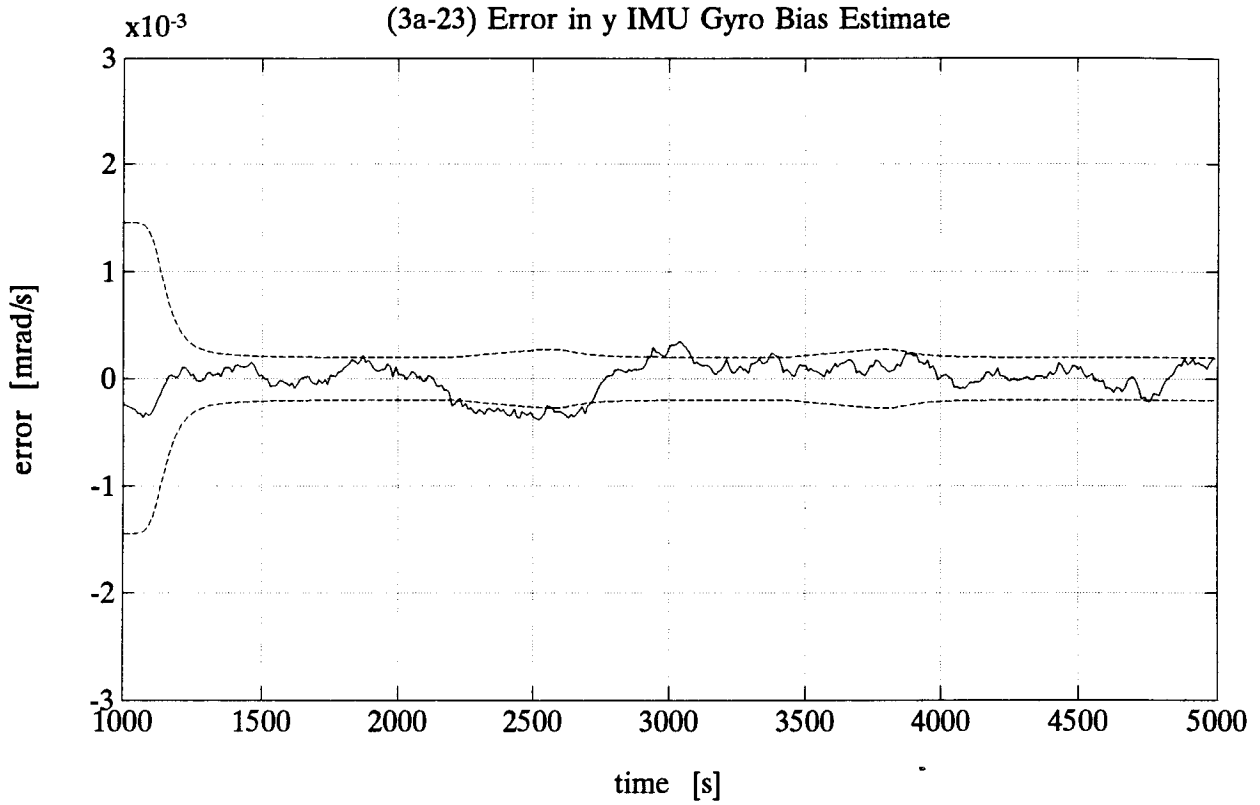


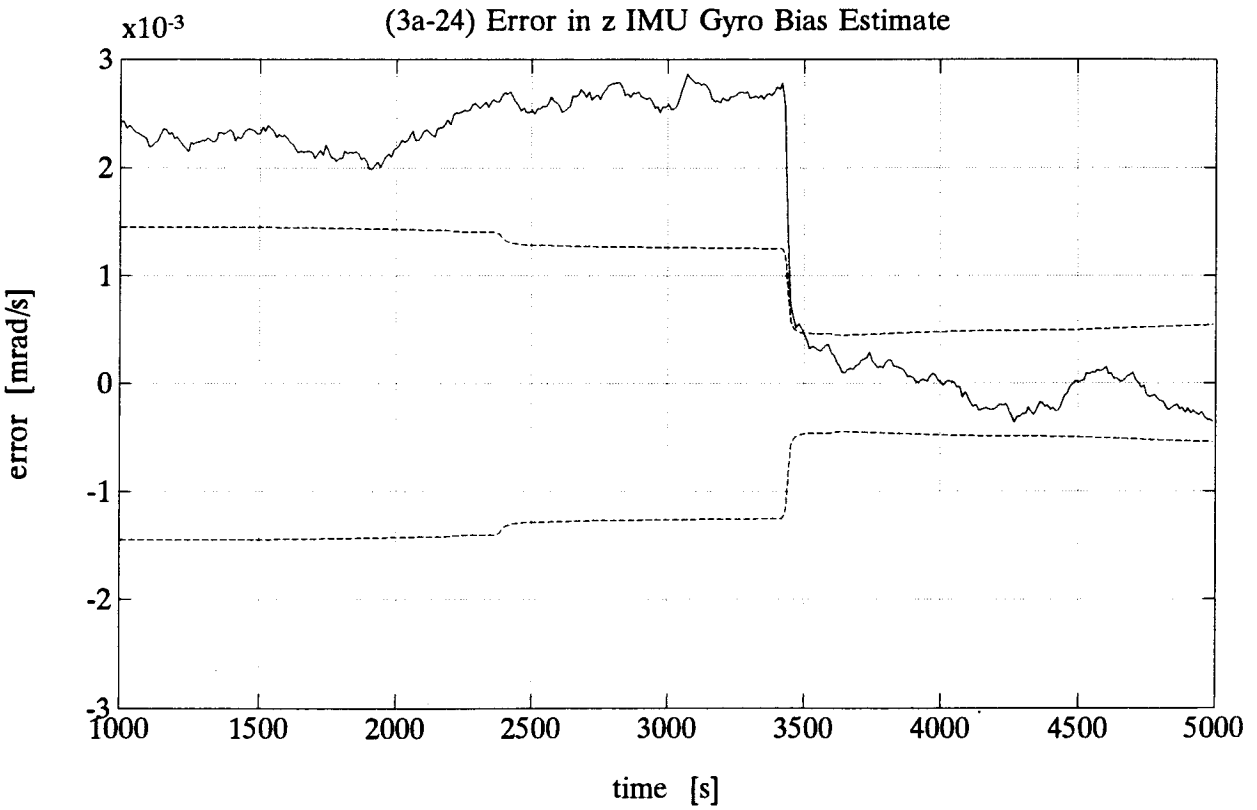
(3a-21) Error in Corrected z IMU Platform Alignment



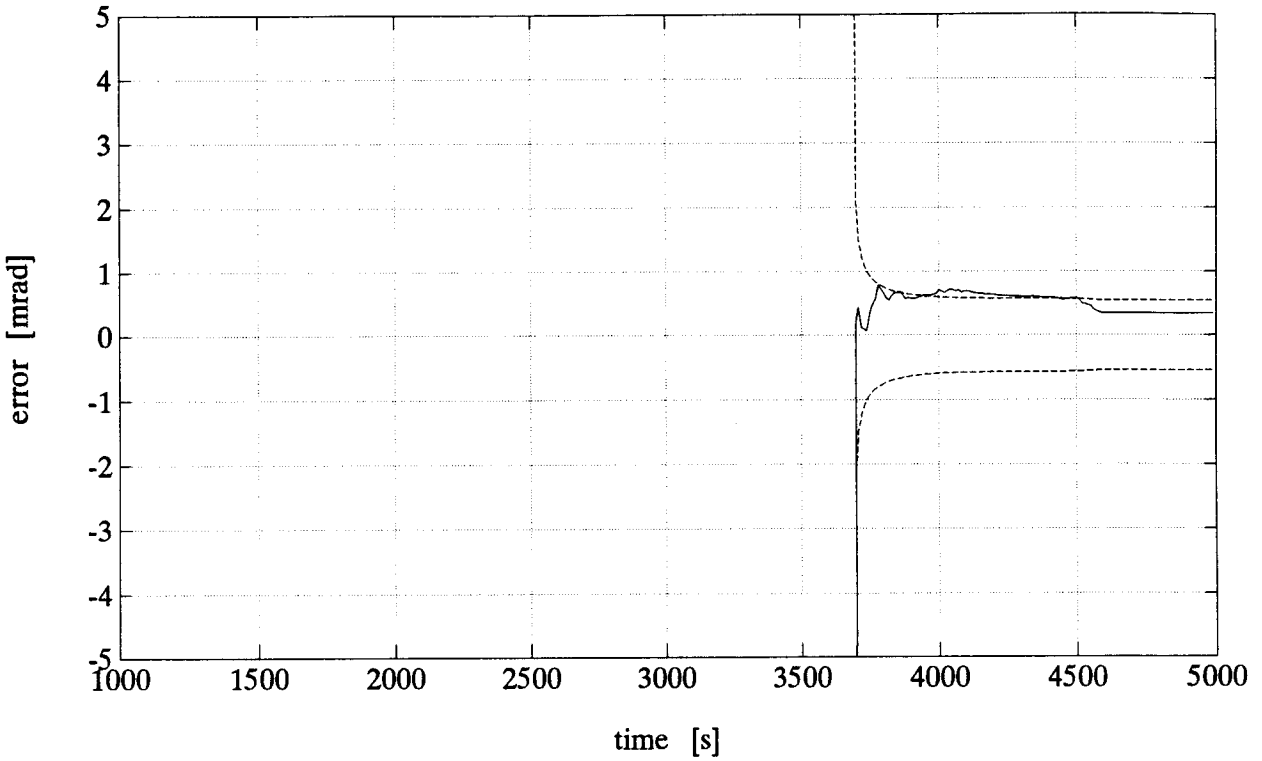
(3a-22) Error in x IMU Gyro Bias Estimate



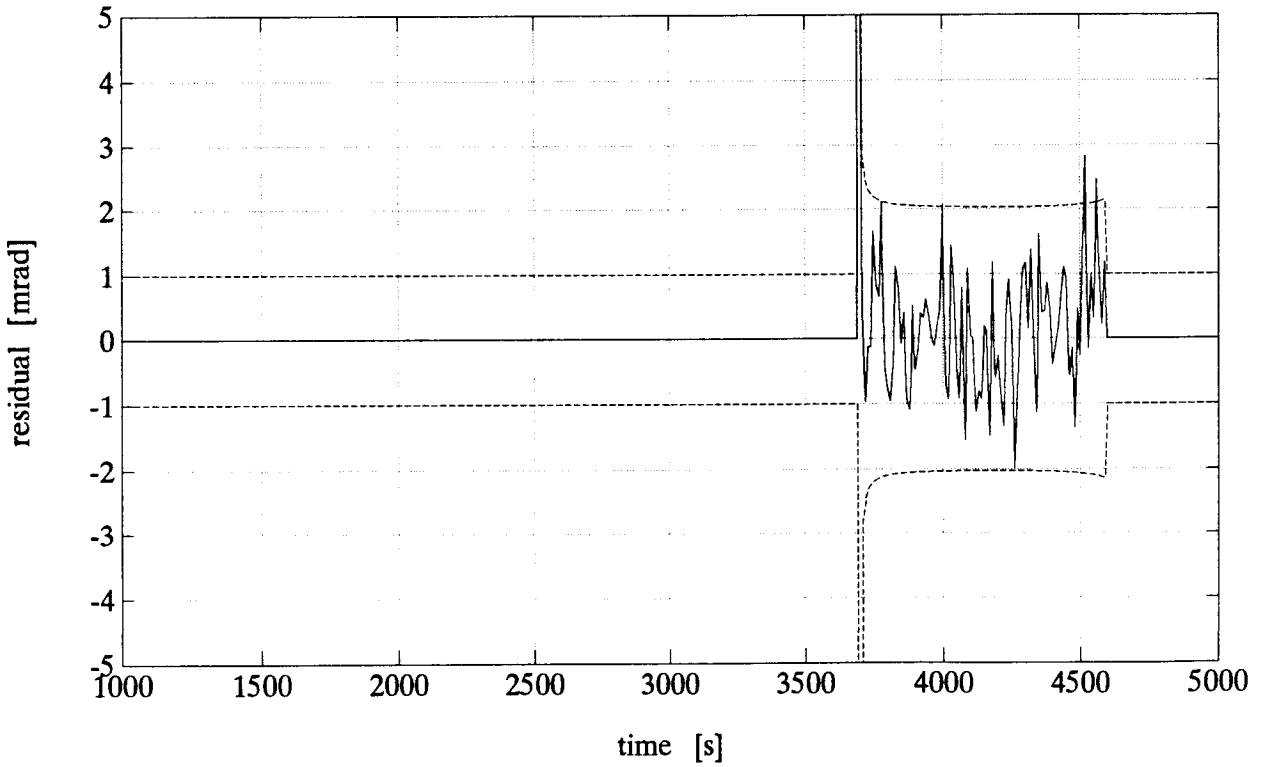

x10-3
(3a-23) Error in y IMU Gyro Bias Estimate
3
2
1
0
-1
-2
-3
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

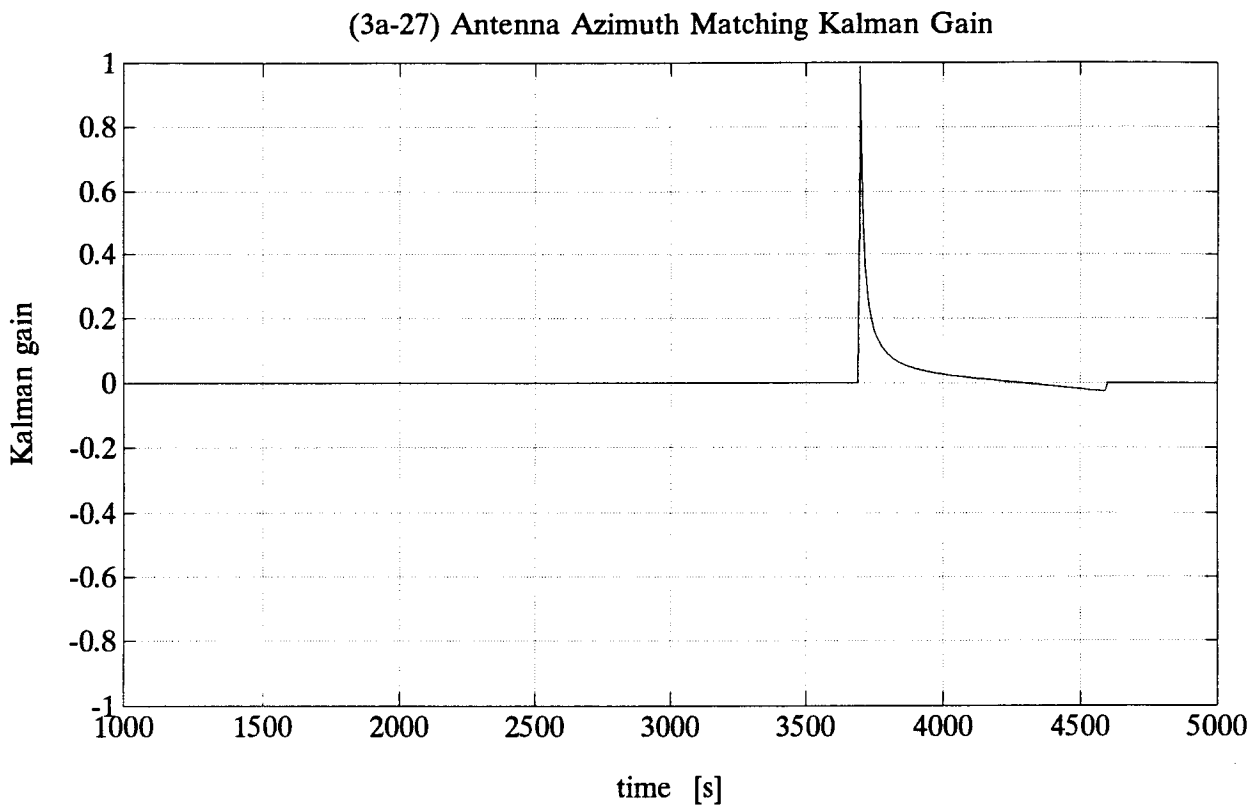

x10-3
(3a-24) Error in z IMU Gyro Bias Estimate
3
2
1
0
-1
-2
-3
error [mrad/s]
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

(3a-25) Error in Antenna Azimuth Misalignment Estimate



(3a-26) Antenna Azimuth Matching Measurement Residual




(3a-27) Antenna Azimuth Matching Kalman Gain
Kalman gain
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
1000
1500
2000
2500
3000
3500
4000
4500
5000
time [s]

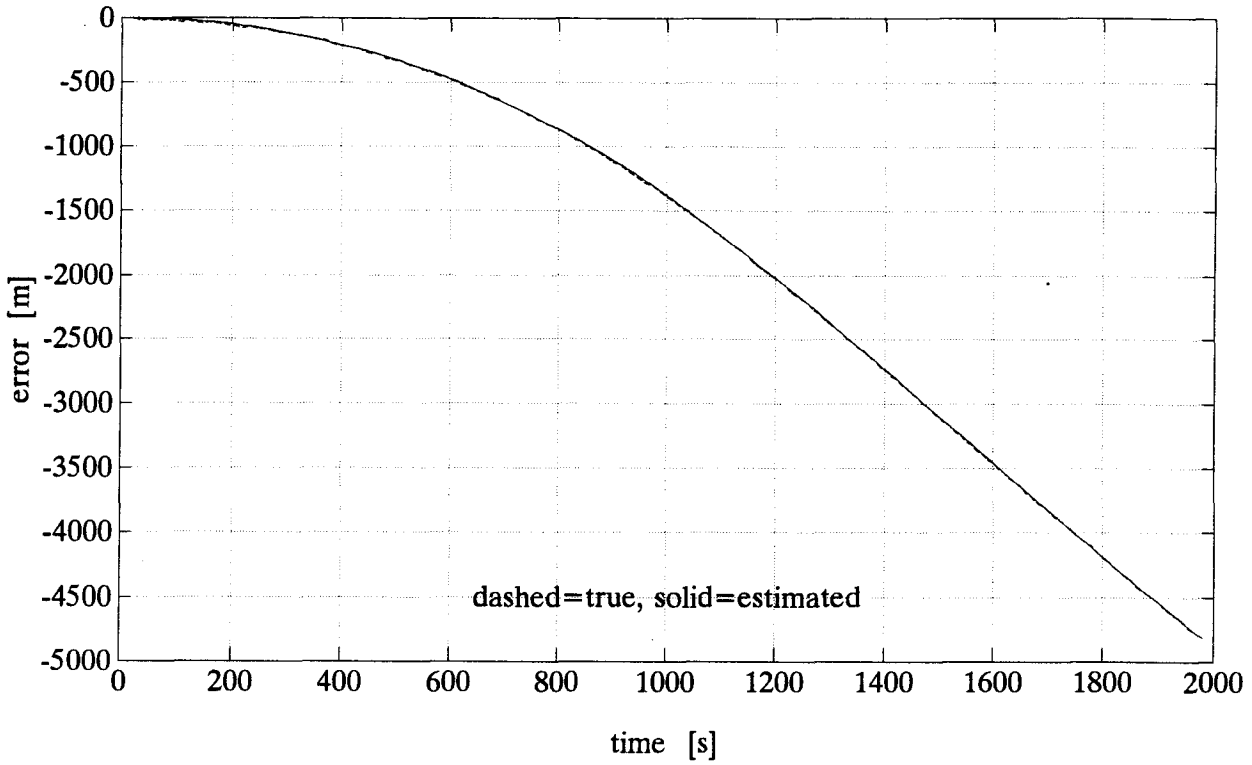
APPENDIX B

PERFORMANCE EVALUATION SIMULATION PLOTS

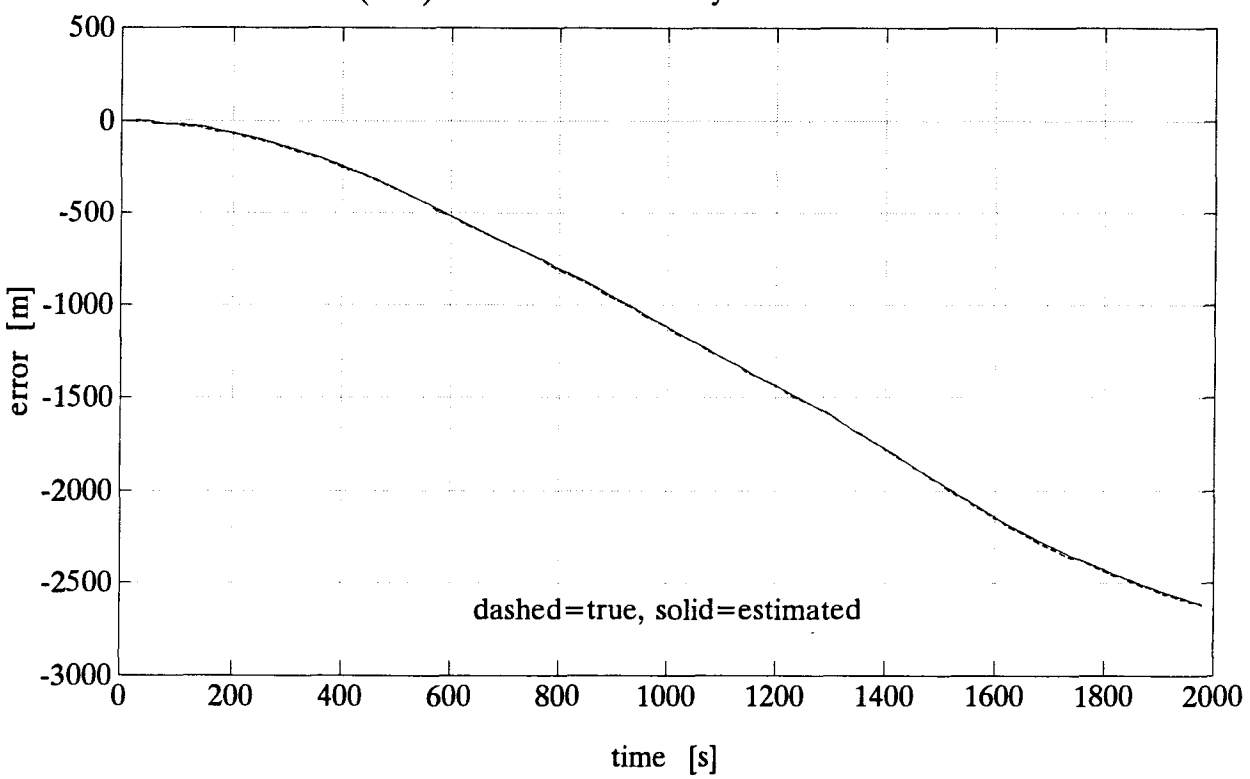
This appendix contains a full set of plots from a representative antenna boresight calibration Kalman filter performance evaluation simulation run (Section 6, simulation run 4c).

It will be noted that some plots have a saw-tooth appearance. In generating the plots, "true" INS errors are computed from the synthesized INS data after subtracting the reference INS data. Quantization of the synthesized INS data results in "noise" in the computed true errors. This artifact of the quantization is most readily observed in the plots of true x and y INS platform misalignment (plots 4c-5 and 4c-6). The effect on plots of the error in Kalman filter estimated INS system states is also very apparent, especially in plots 4c-8, 4c-9, 4c-12 and 4c-13.

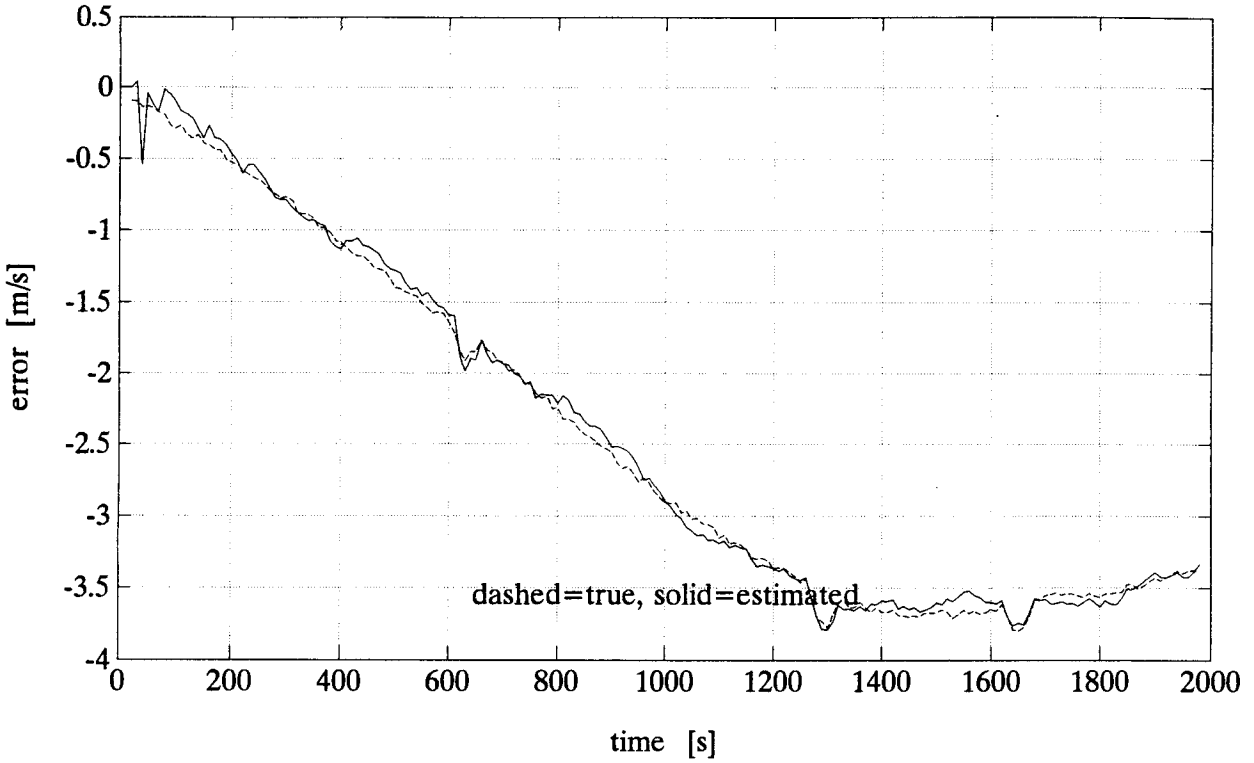
(4c-1) True and Estimated x INS Position Error



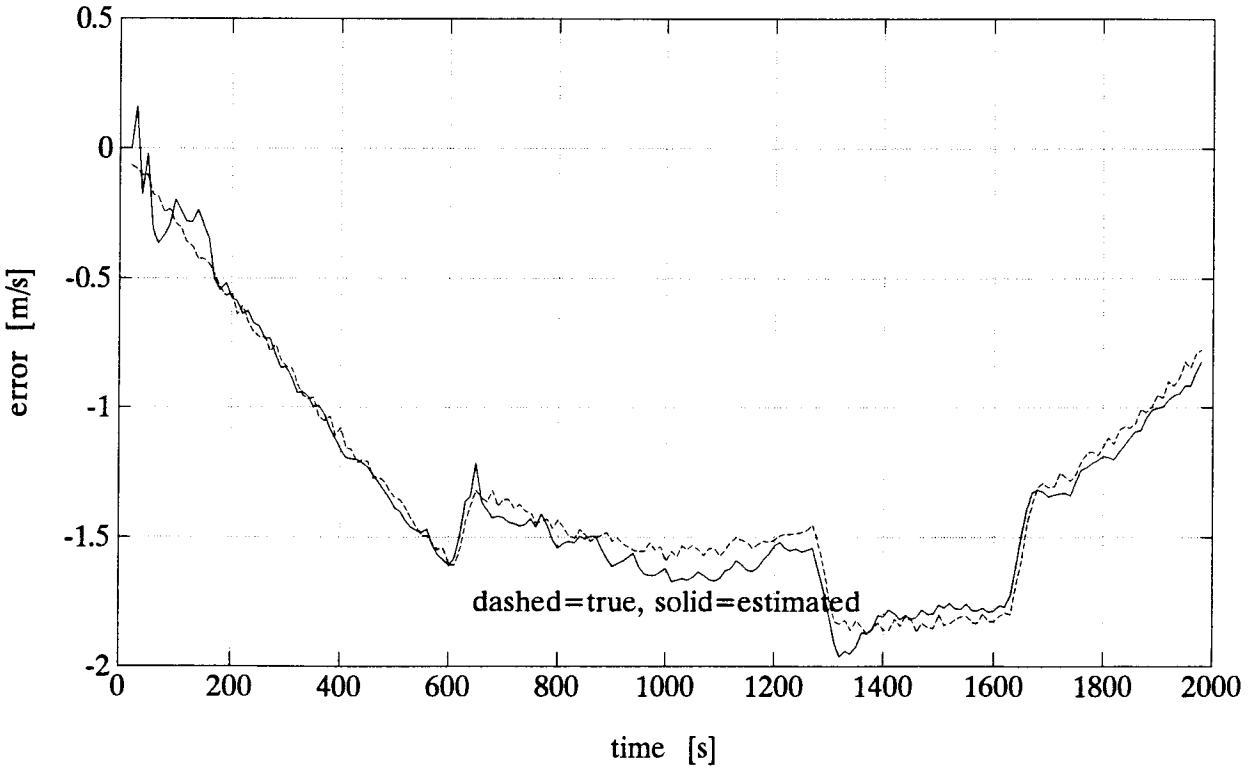
(4c-2) True and Estimated y INS Position Error



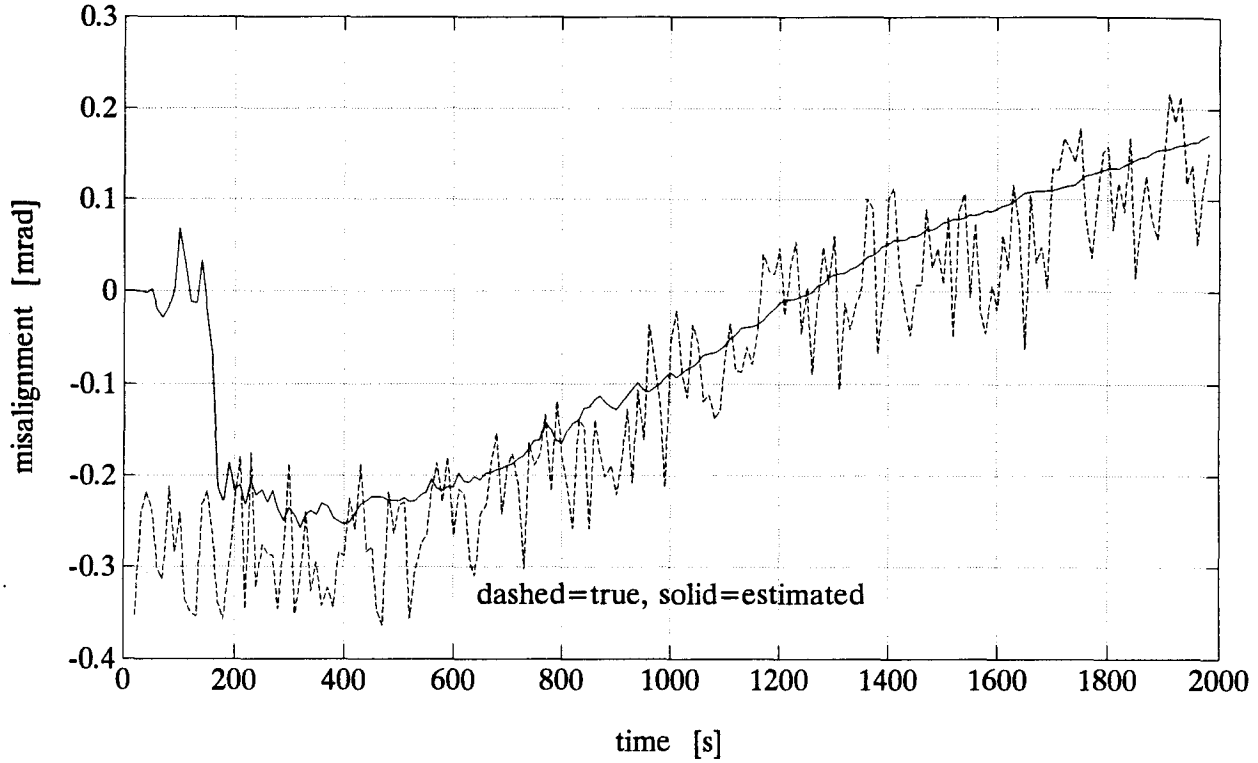
(4c-3) True and Estimated x INS Velocity Error



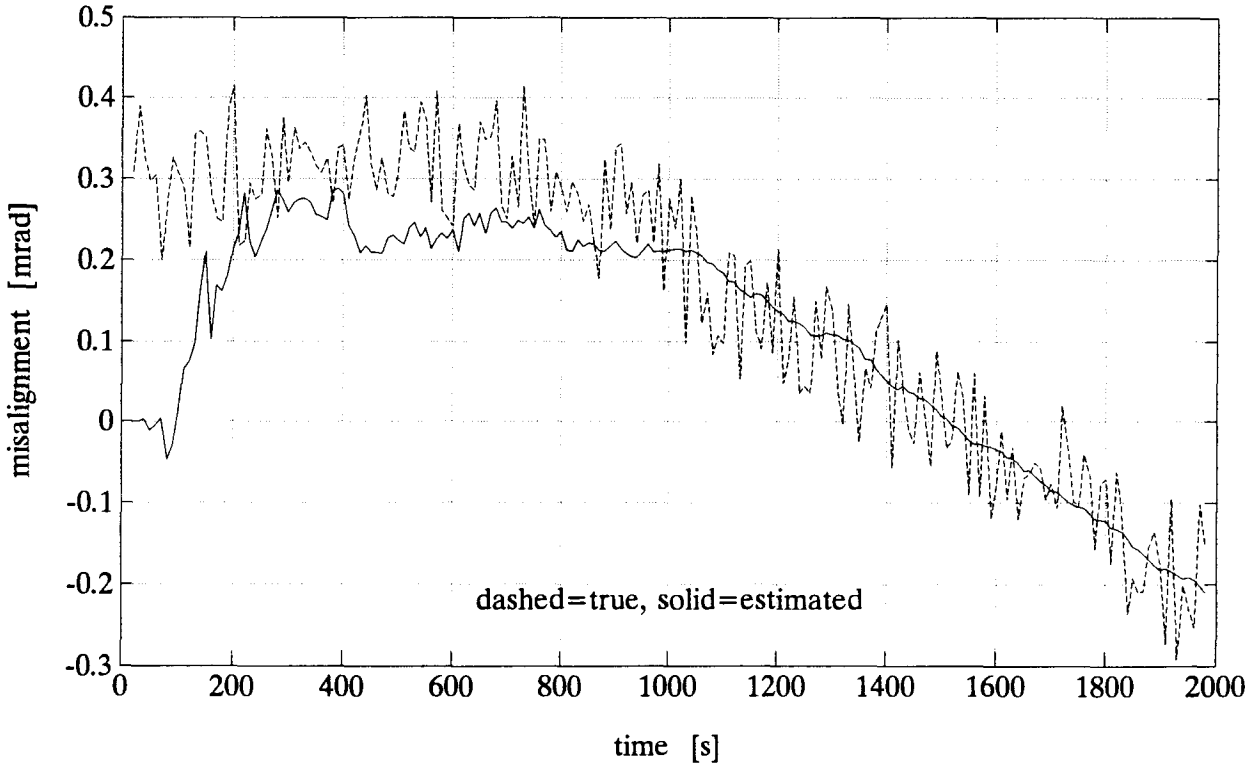
(4c-4) True and Estimated y INS Velocity Error

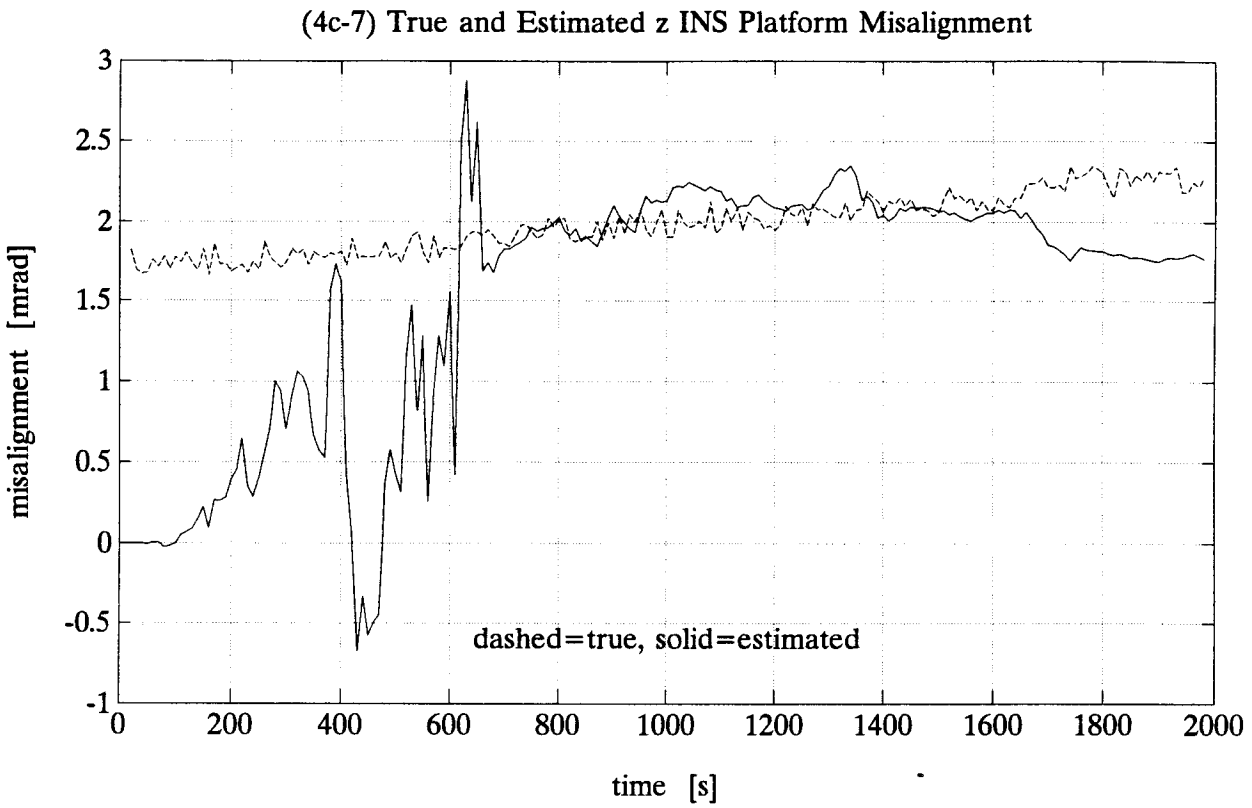


(4c-5) True and Estimated x INS Platform Misalignment

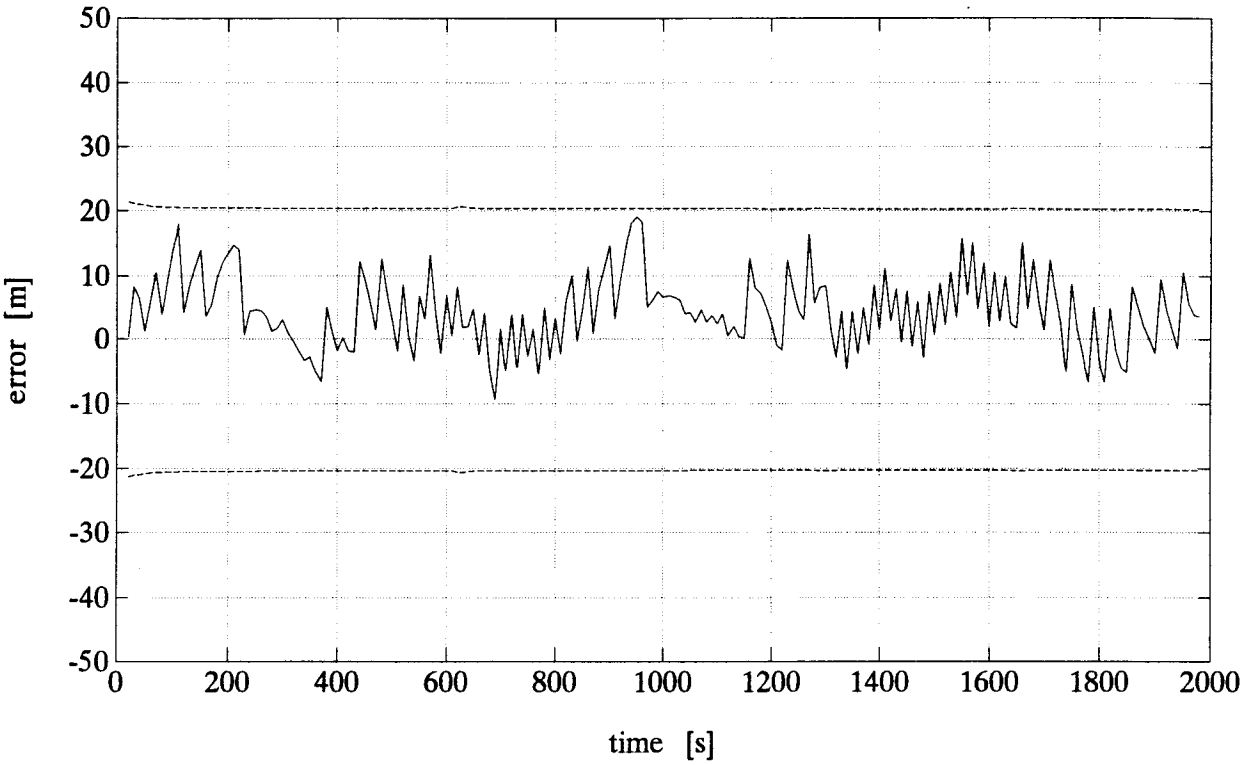


(4c-6) True and Estimated y INS Platform Misalignment

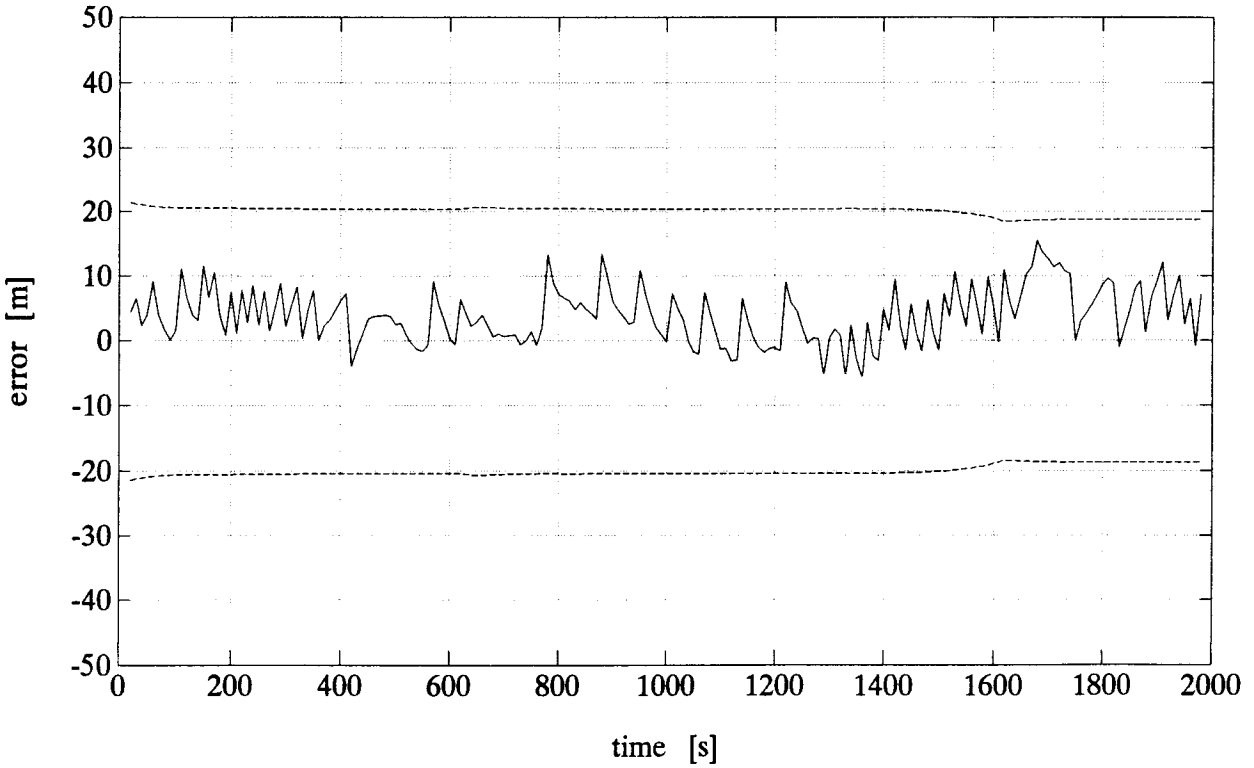



(4c-7) True and Estimated z INS Platform Misalignment
misalignment [mrad]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
200
400
600
800
1000
1200
1400
1600
1800
2000
time [s]
dashed=true, solid=estimated

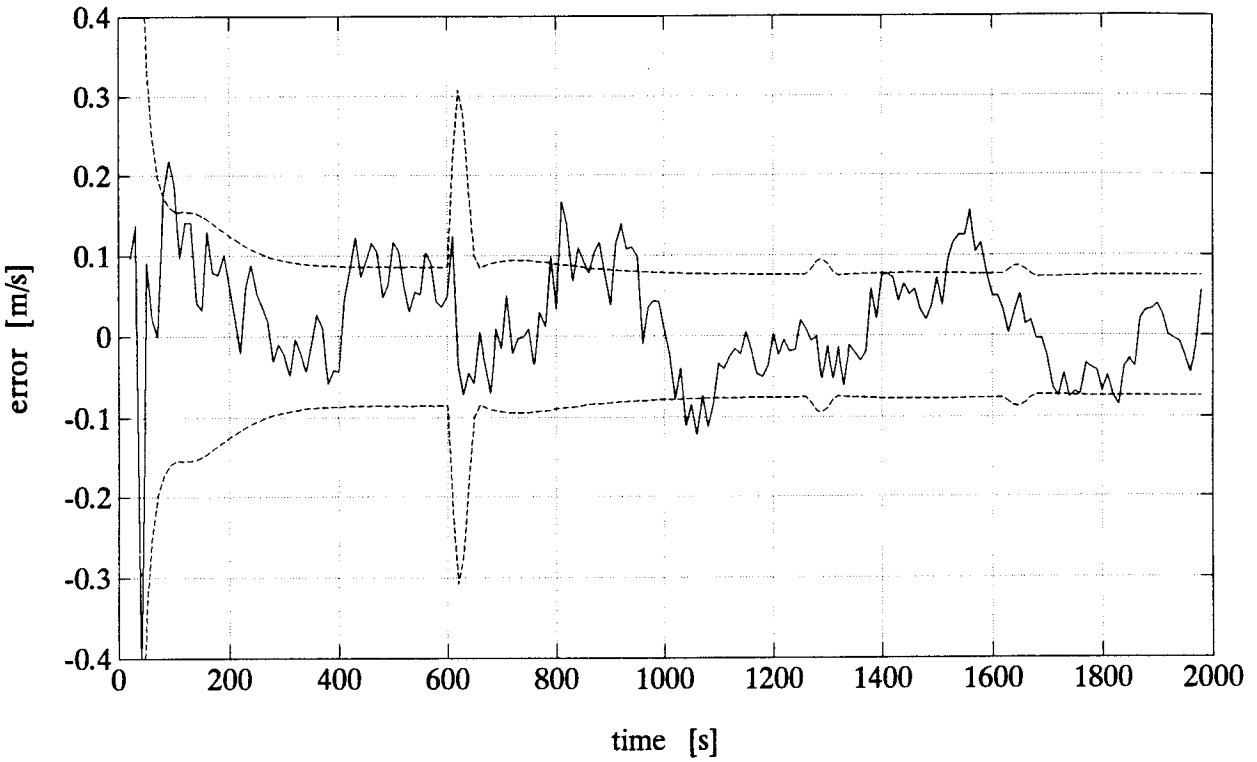
(4c-8) Error in x INS Position Error Estimate



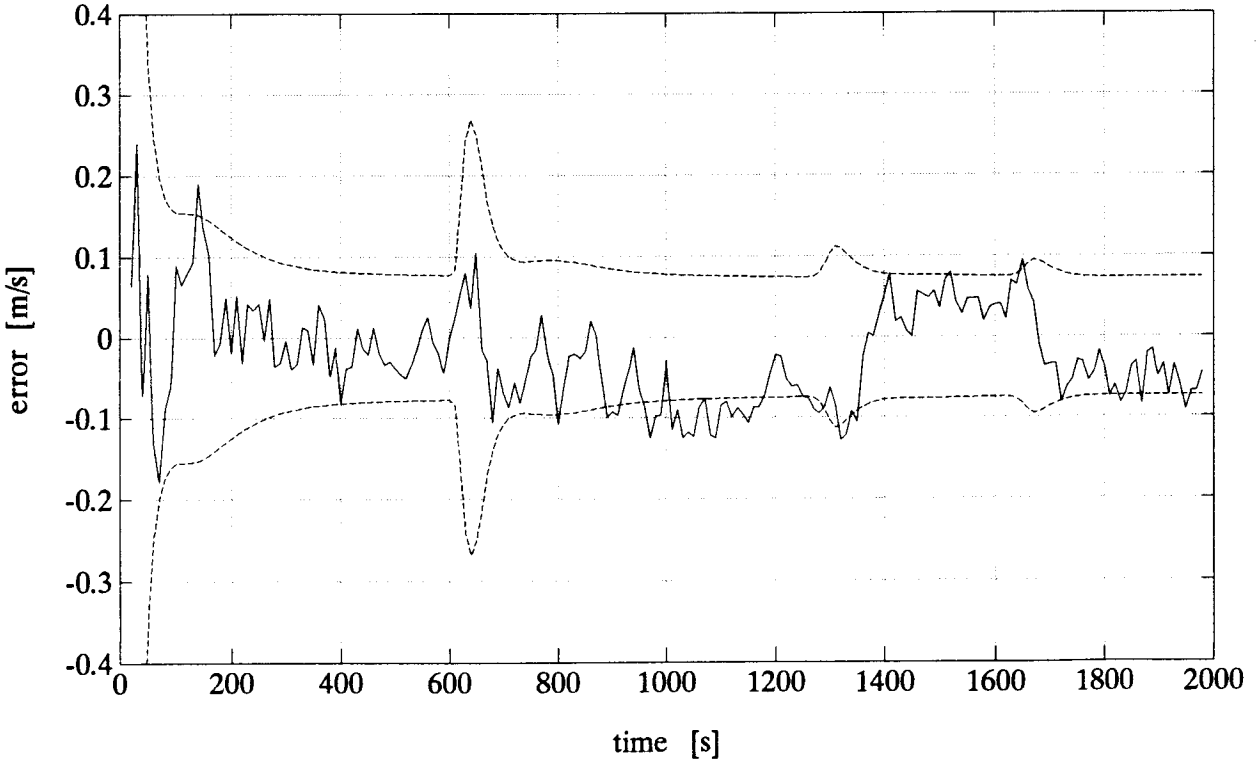
(4c-9) Error in y INS Position Error Estimate



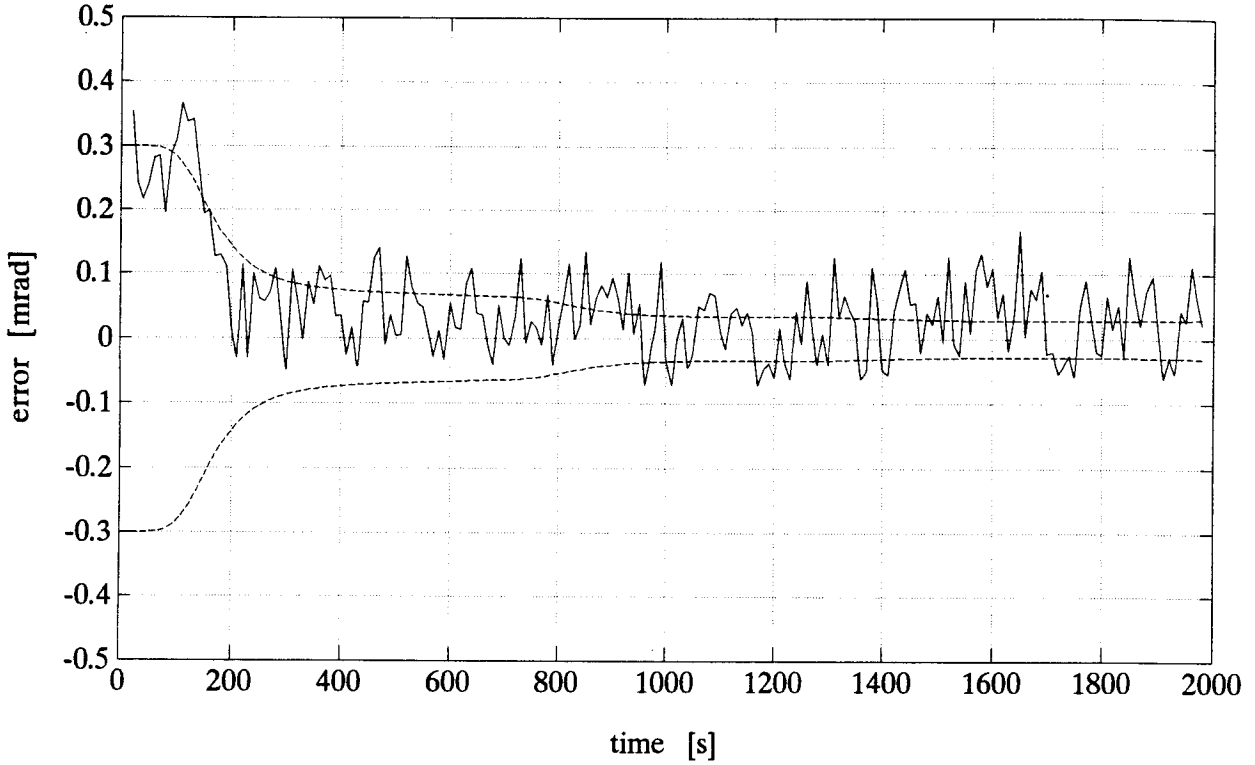
(4c-10) Error in x INS Velocity Error Estimate



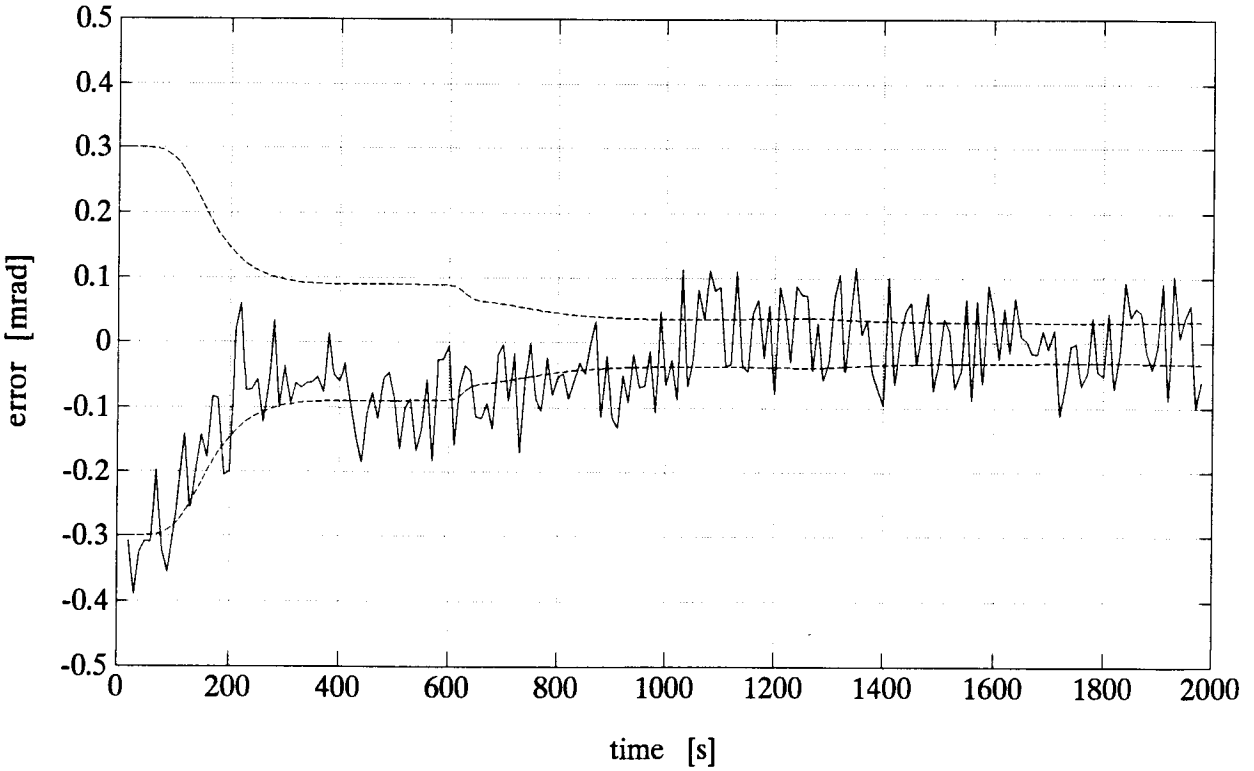
(4c-11) Error in y INS Velocity Error Estimate



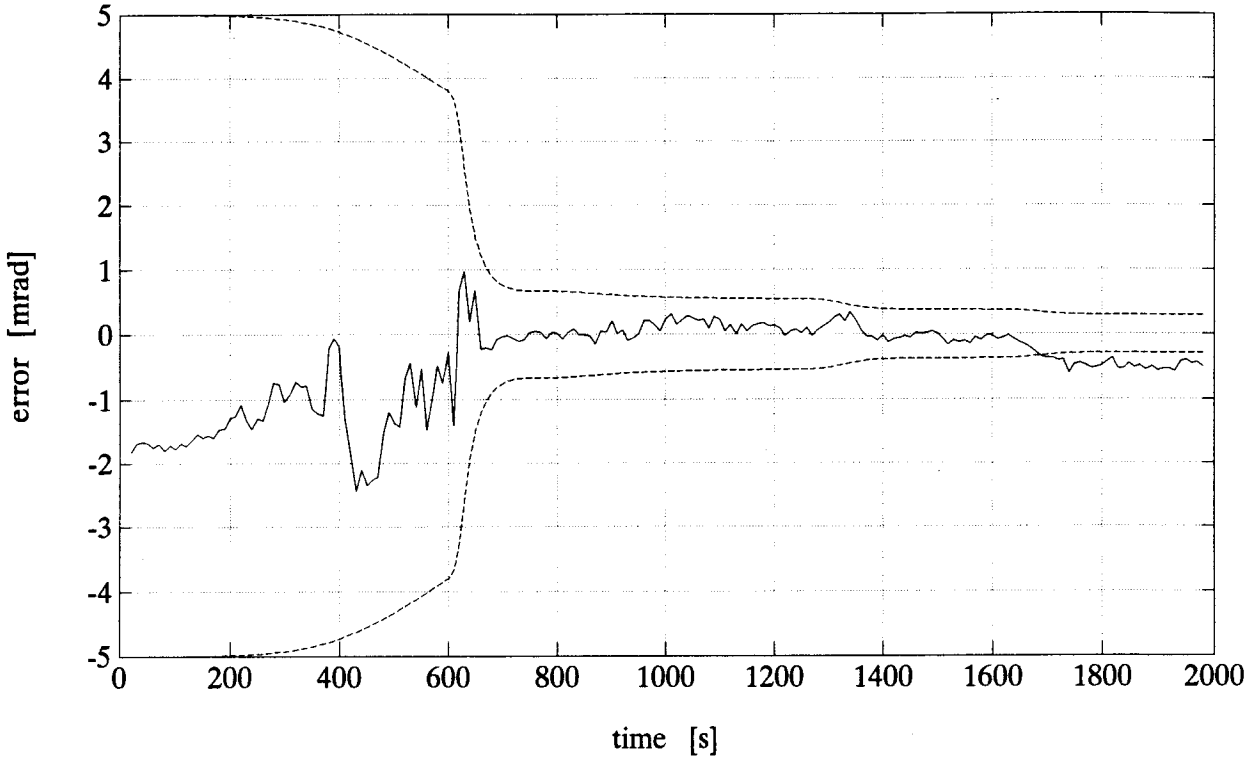
(4c-12) Error in x INS Platform Misalignment Estimate



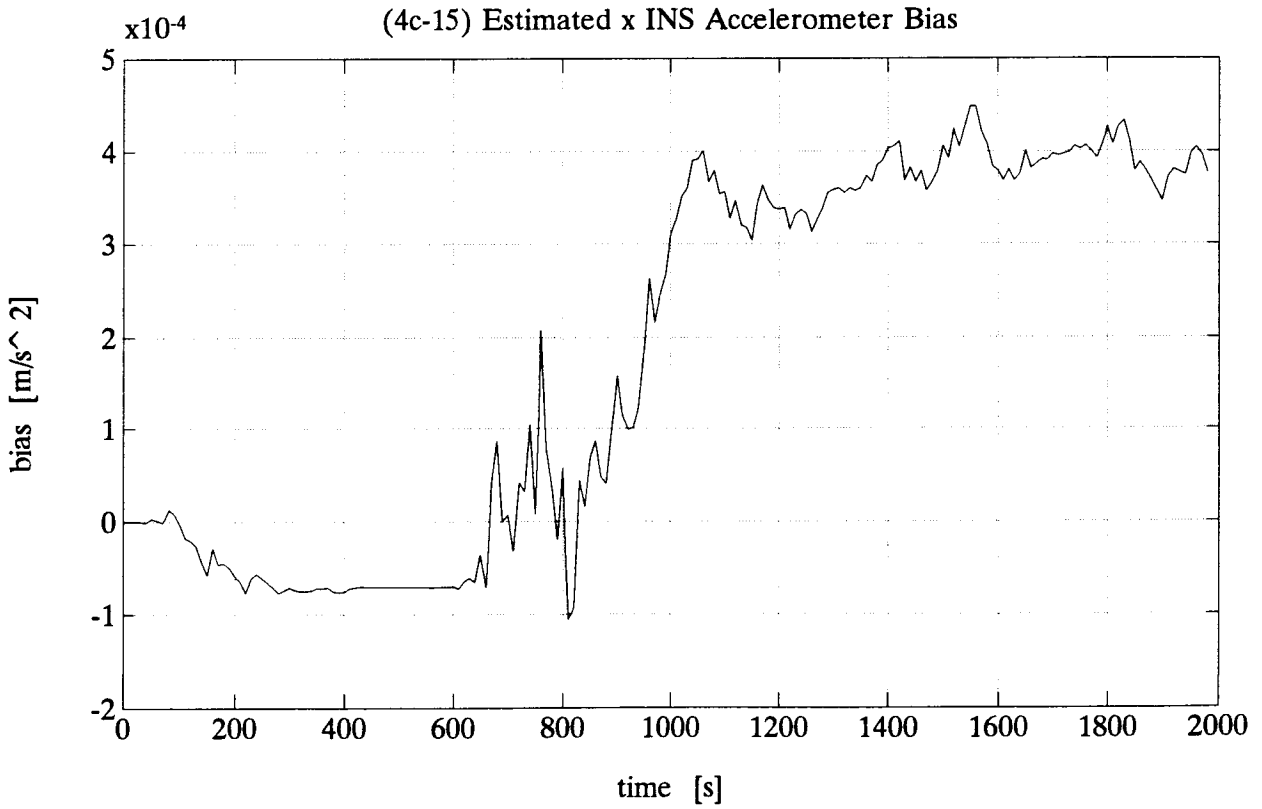
(4c-13) Error in y INS Platform Misalignment Estimate

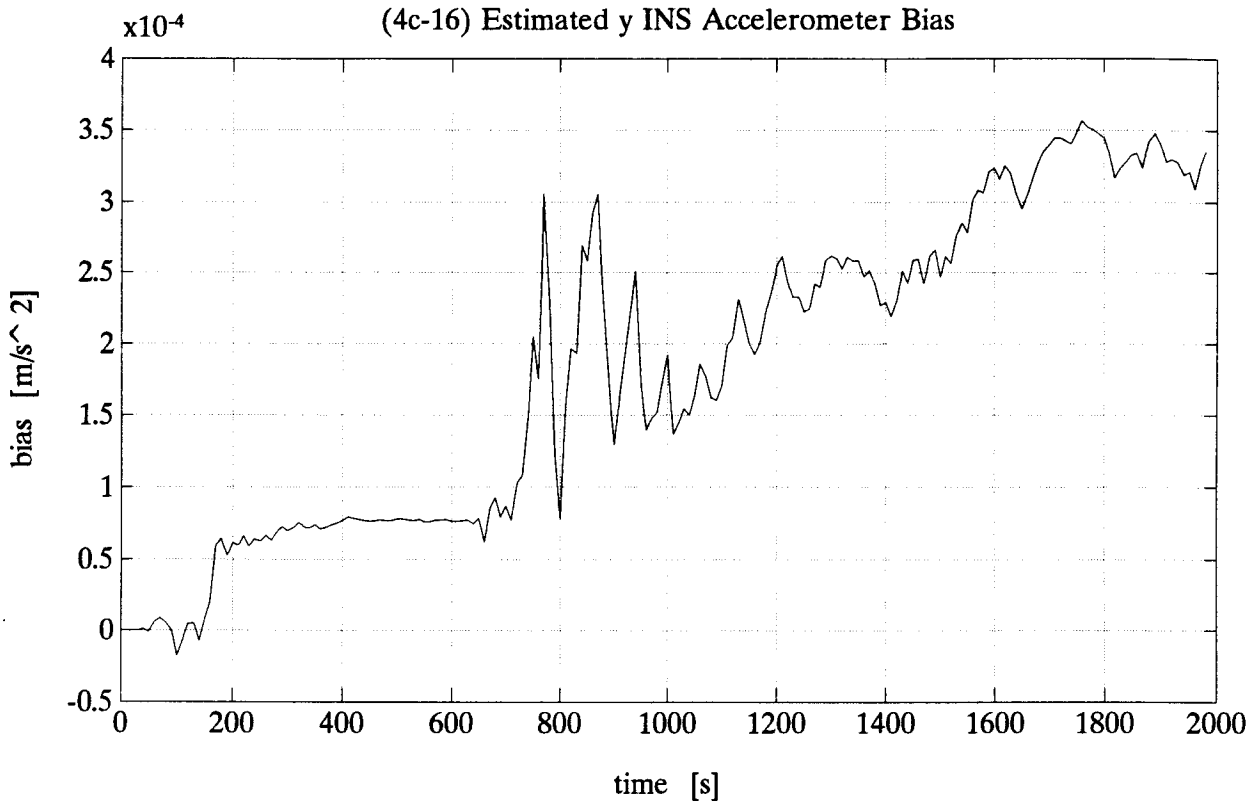


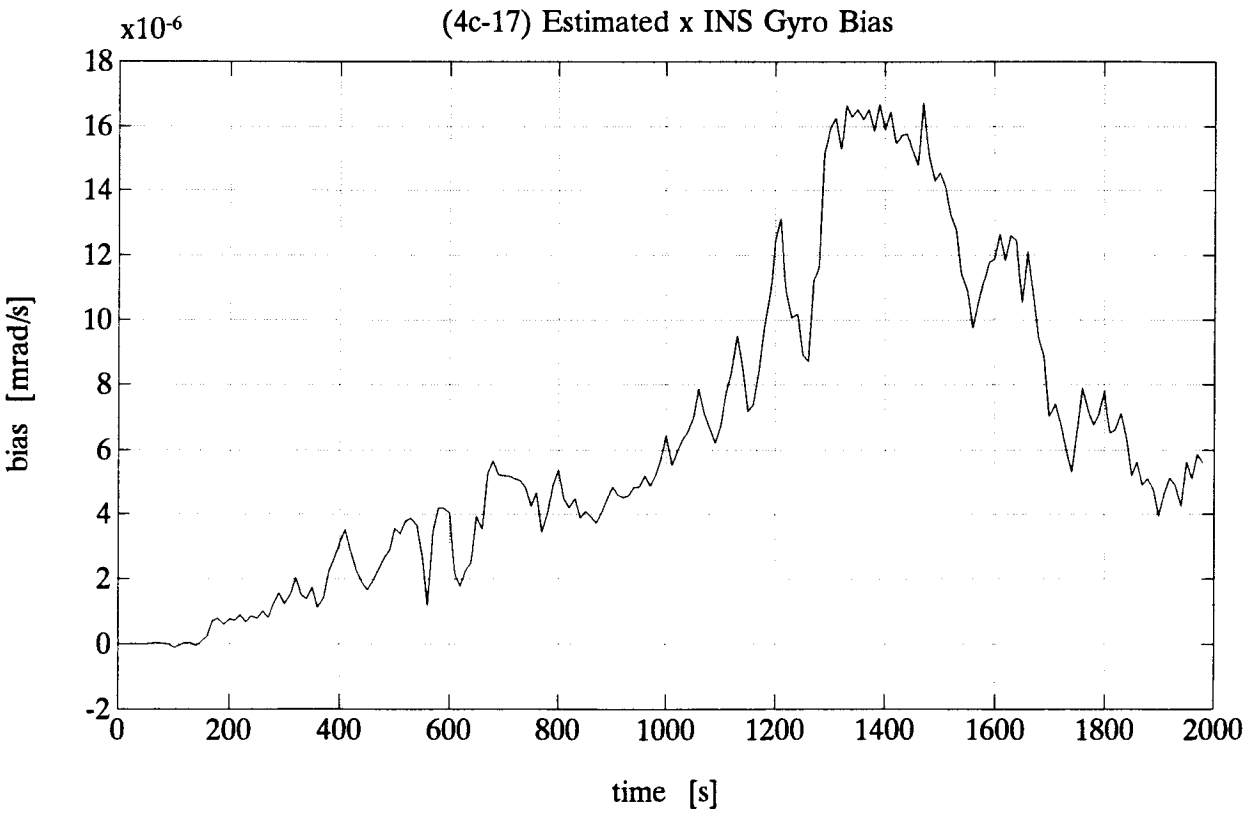
(4c-14) Error in z INS Platform Misalignment Estimate

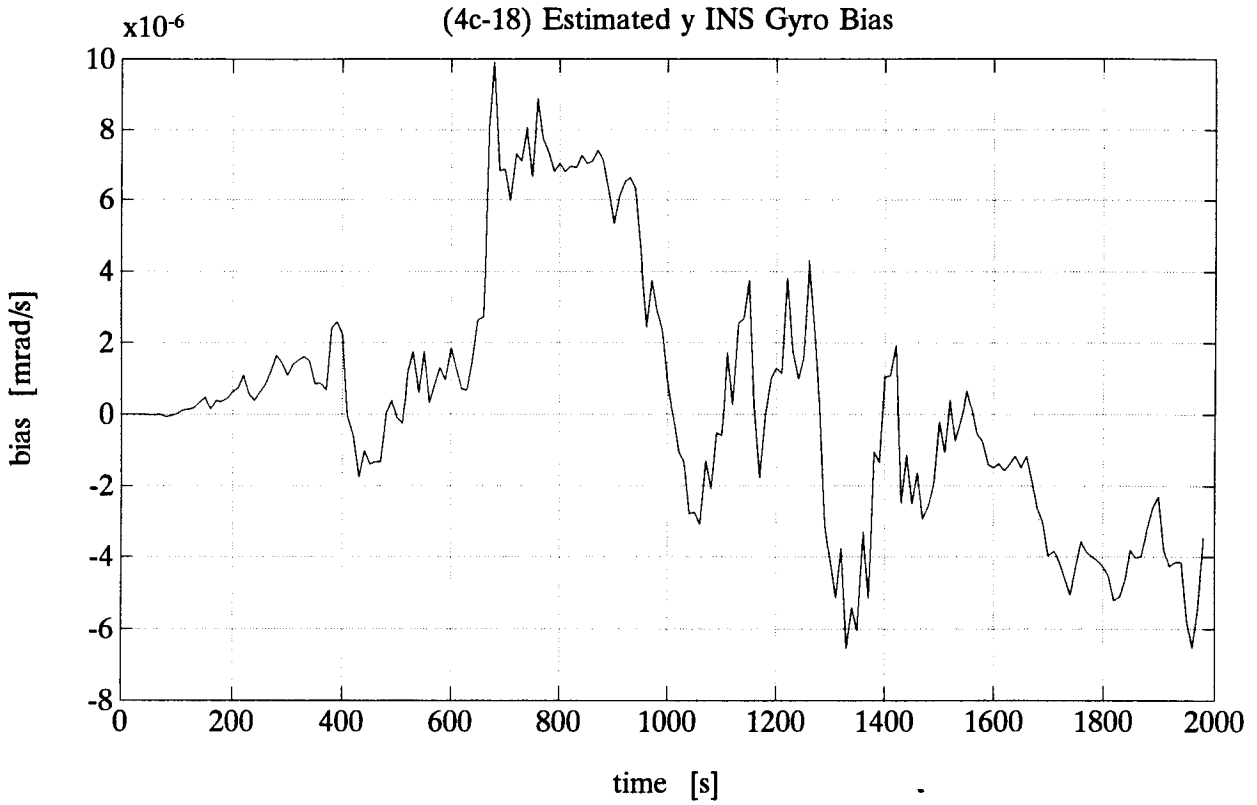


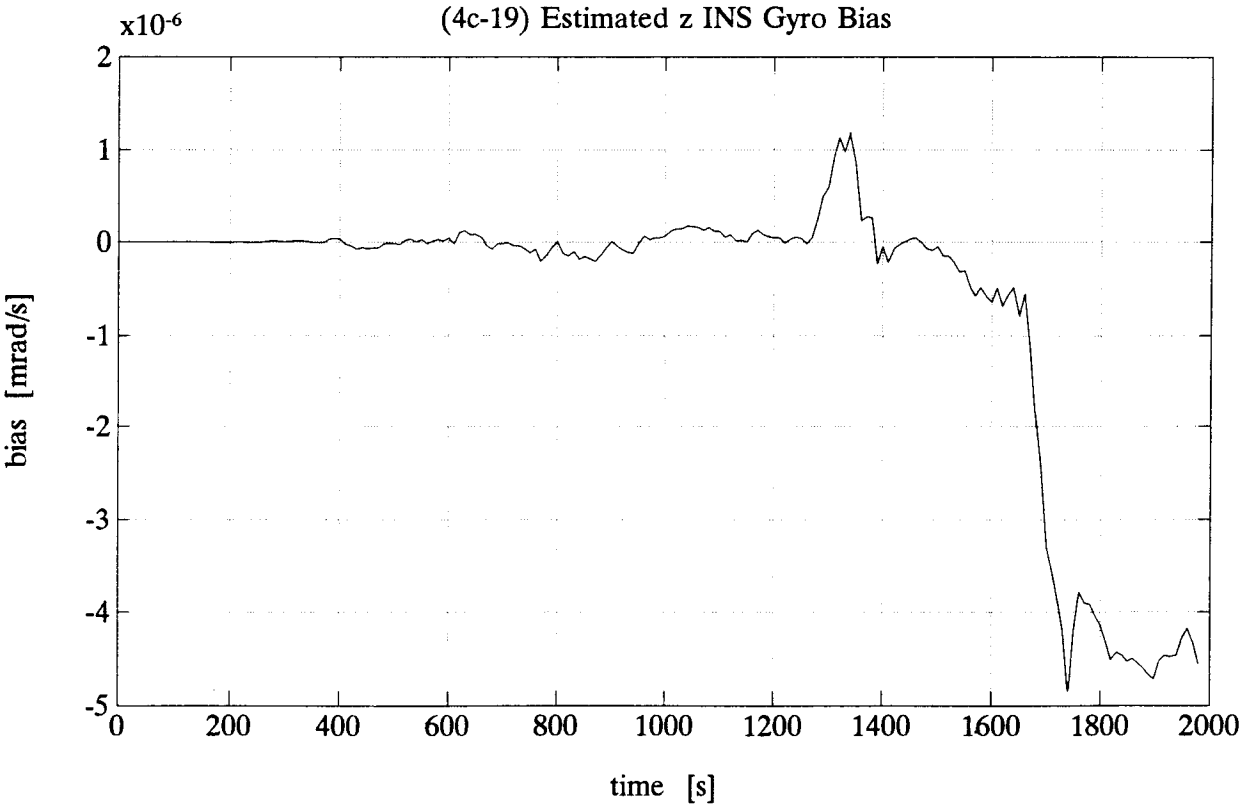
(4c-15) Estimated x INS Accelerometer Bias



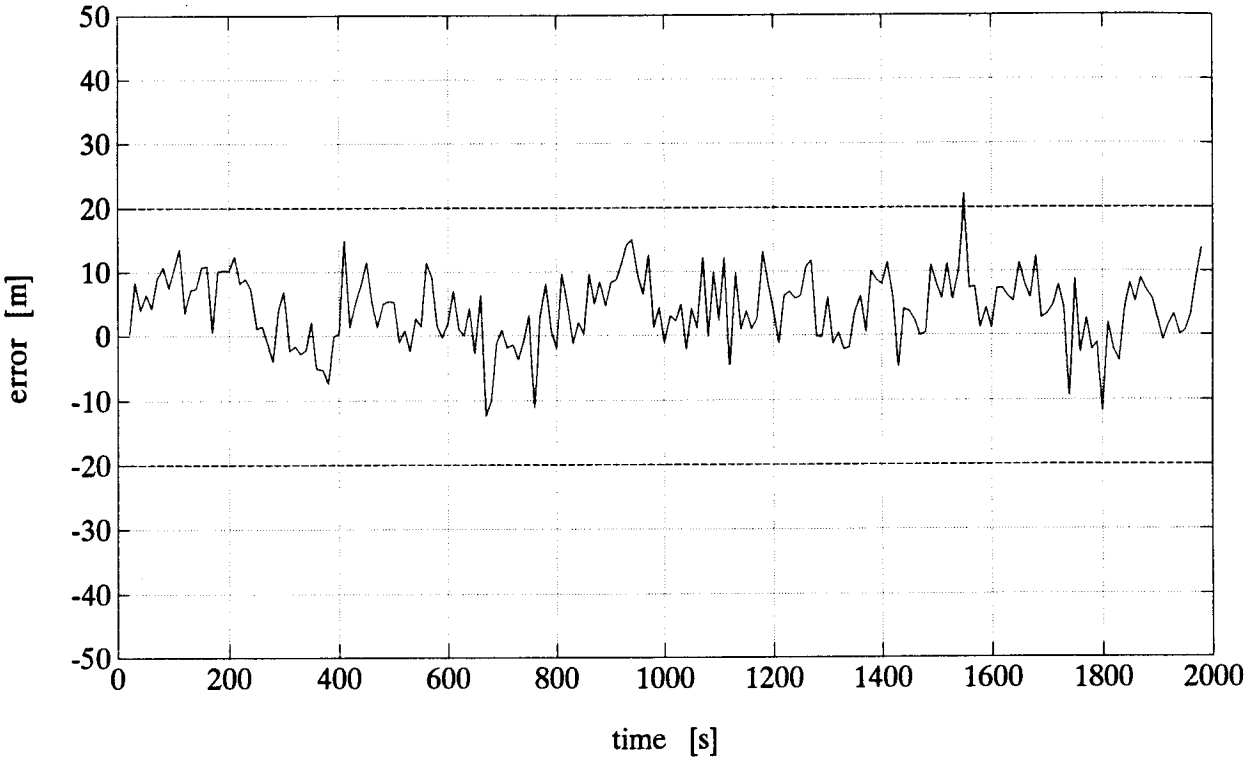

x10-4
(4c-16) Estimated y INS Accelerometer Bias
bias [m/s^ 2]
time [s]


x10-6
(4c-17) Estimated x INS Gyro Bias
bias [mrad/s]
time [s]

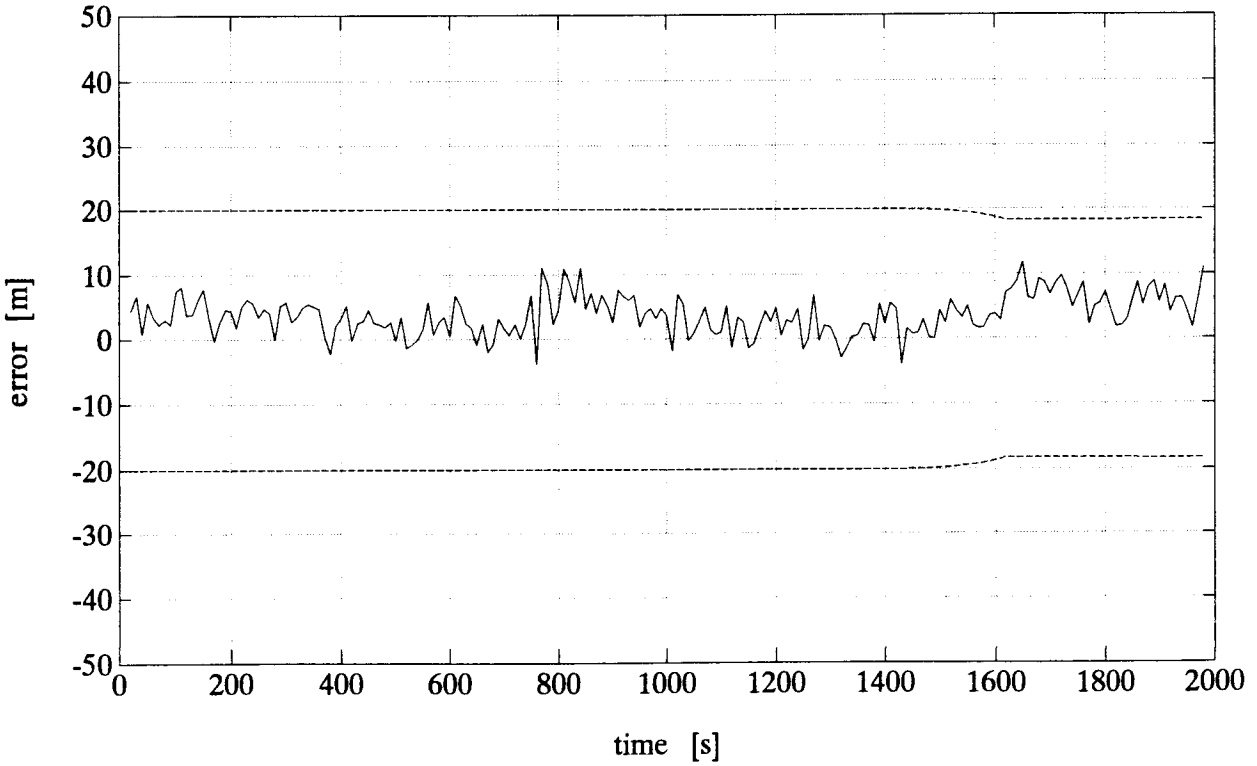

(4c-18) Estimated y INS Gyro Bias
x10-6
10
8
6
4
2
0
-2
-4
-6
-8
bias [mrad/s]
0
200
400
600
800
1000
1200
1400
1600
1800
2000
time [s]


(4c-19) Estimated z INS Gyro Bias
x10-6
2
1
0
-1
-2
-3
-4
-5
bias [mrad/s]
0
200
400
600
800
1000
1200
1400
1600
1800
2000
time [s]

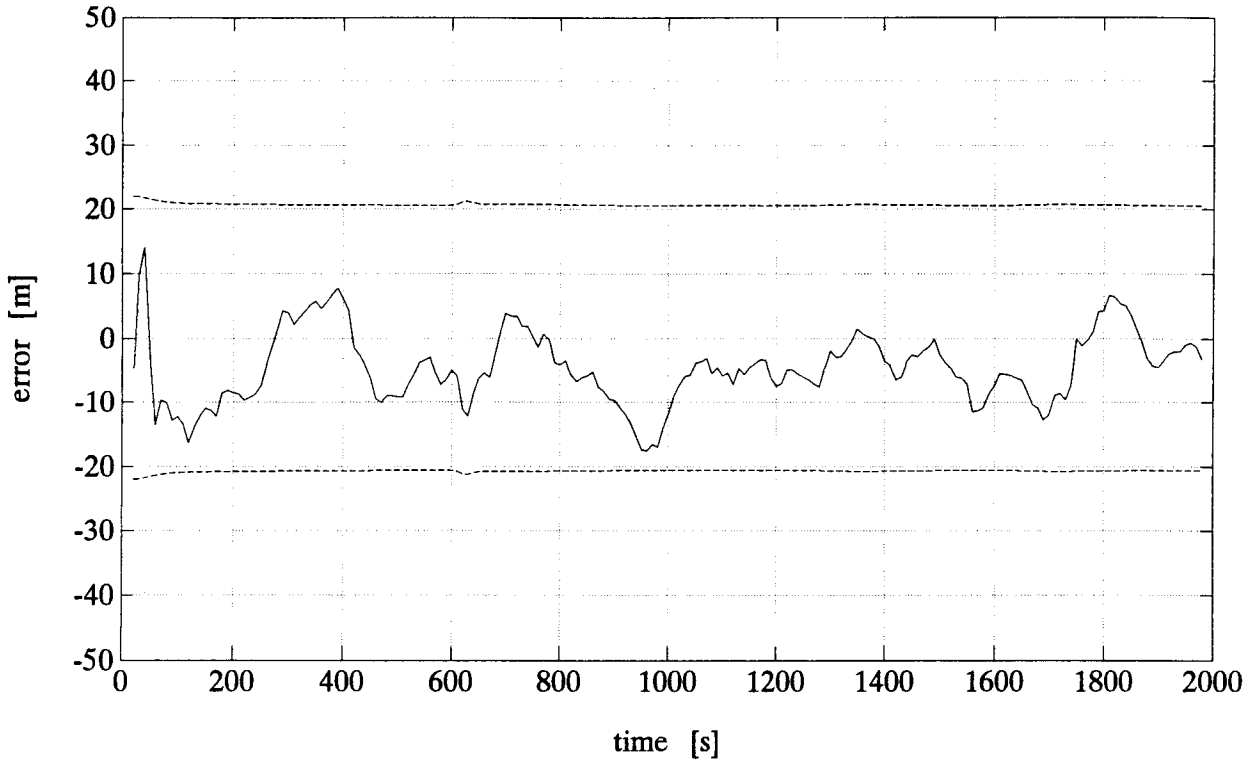
(4c-20) Error in x GPS Position Error Estimate



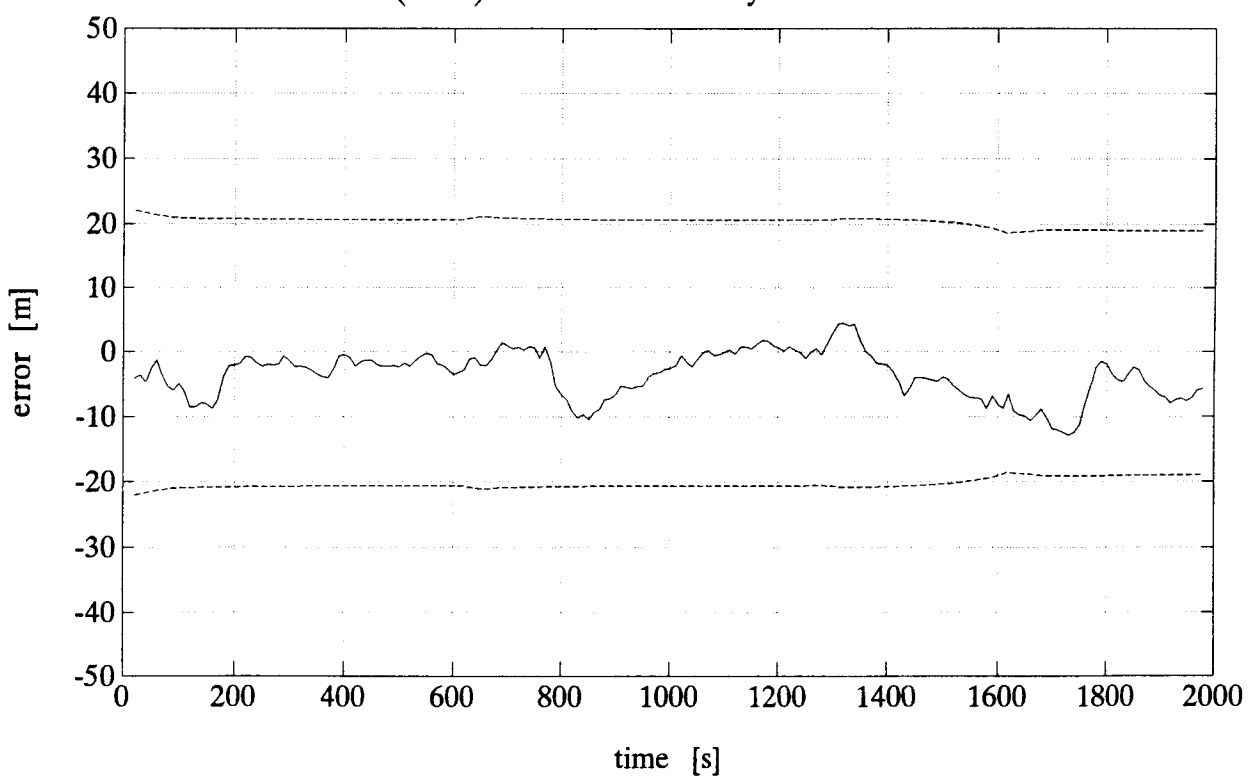
(4c-21) Error in y GPS Position Error Estimate



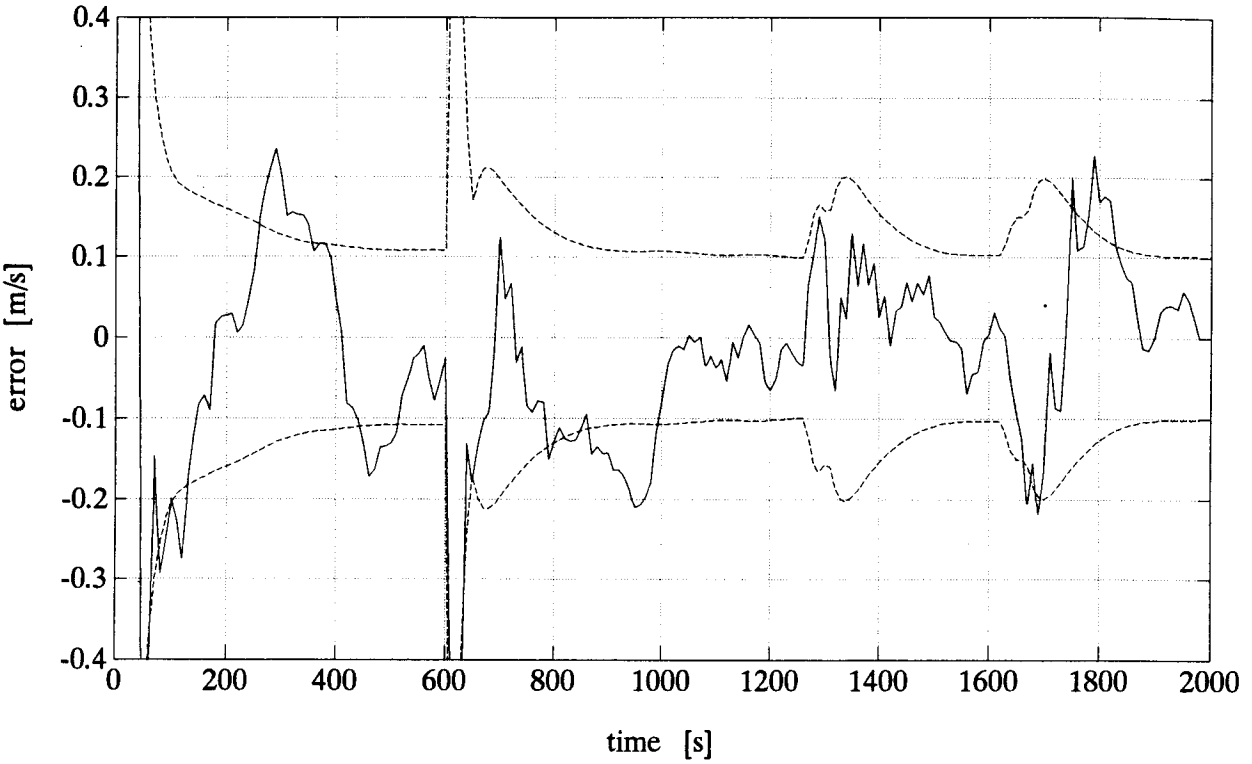
(4c-22) Error in Corrected x IMU Position



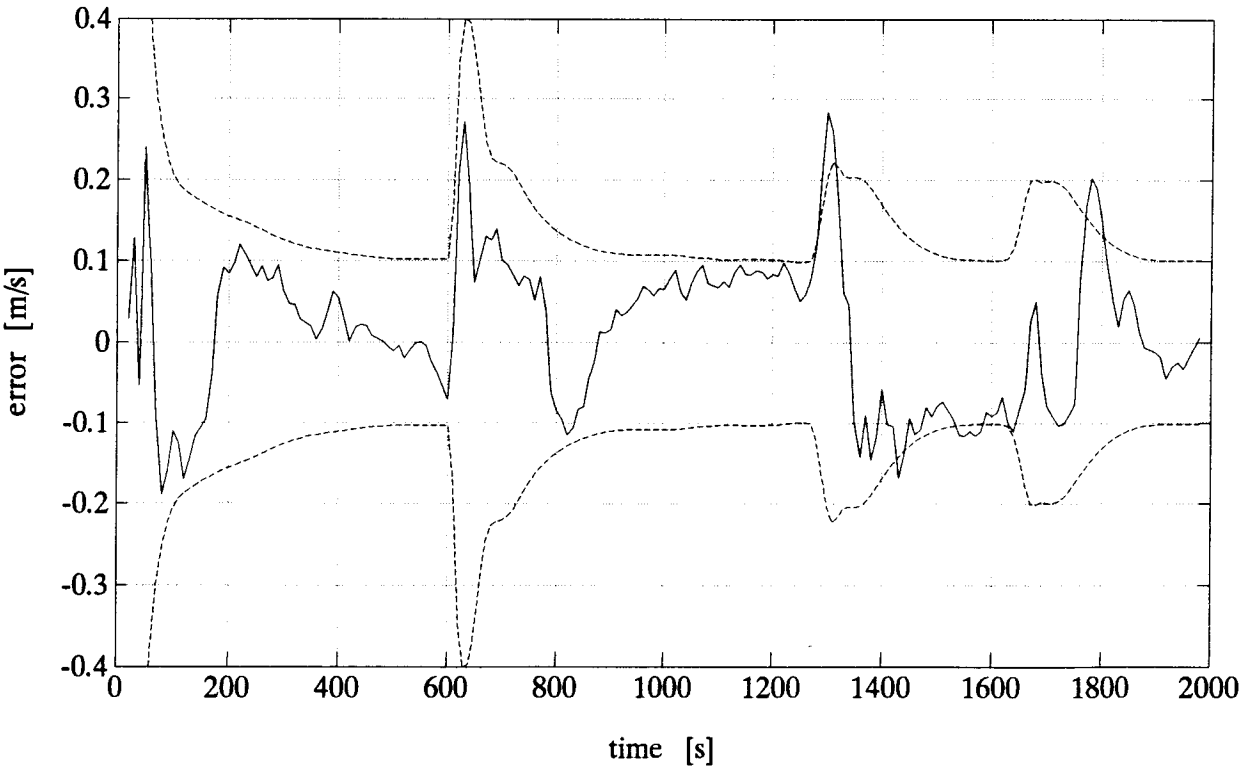
(4c-23) Error in Corrected y IMU Position



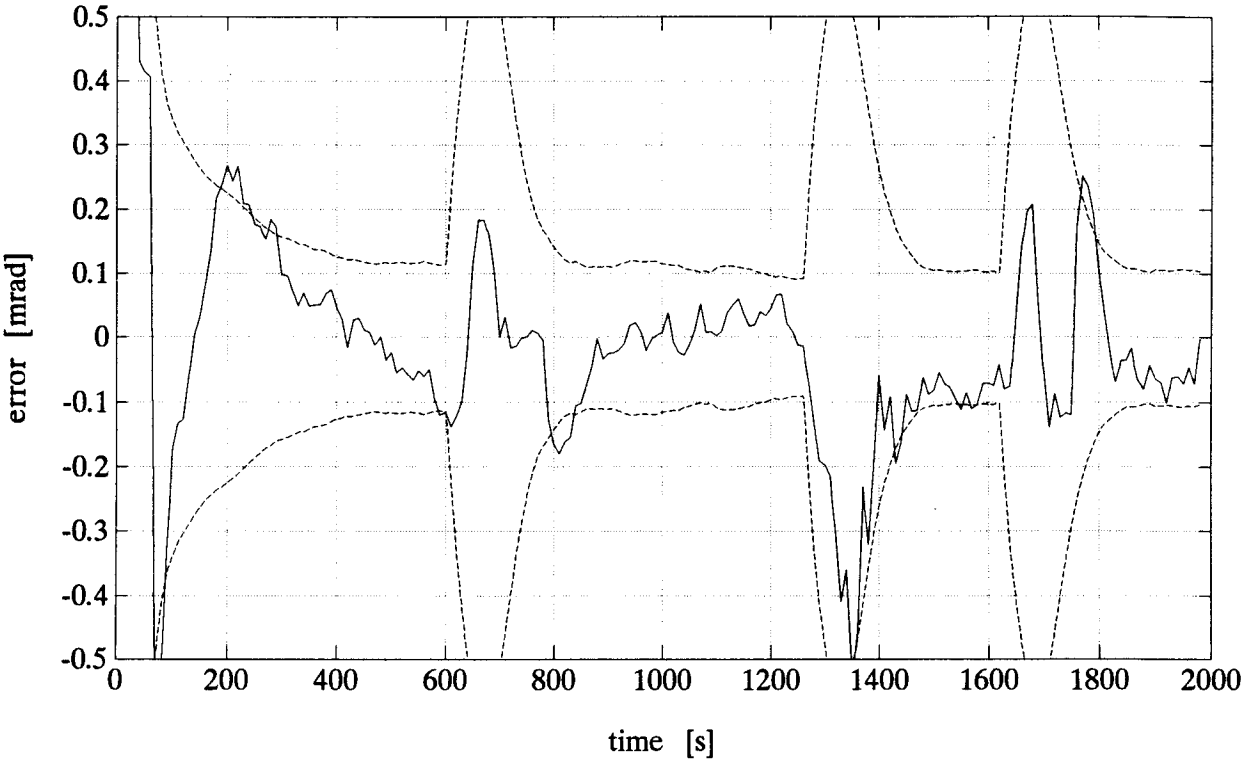
(4c-24) Error in Corrected x IMU Velocity



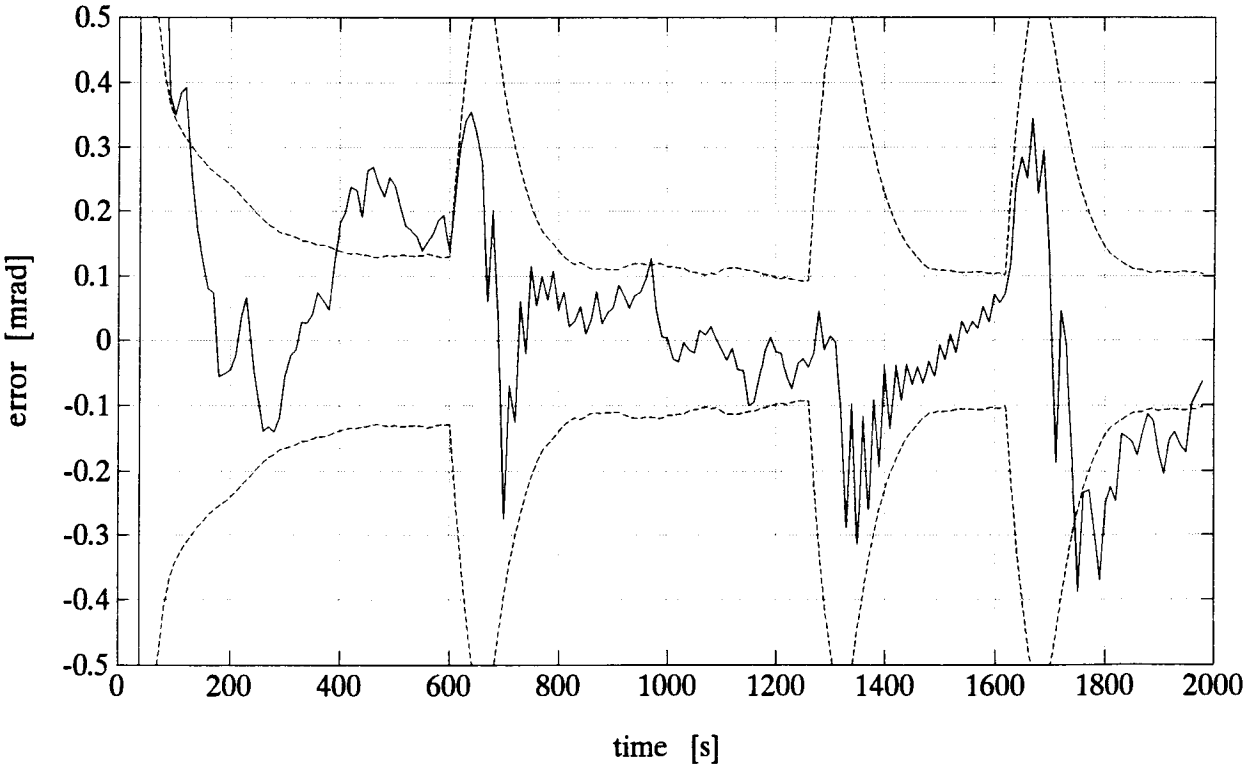
(4c-25) Error in Corrected y IMU Velocity



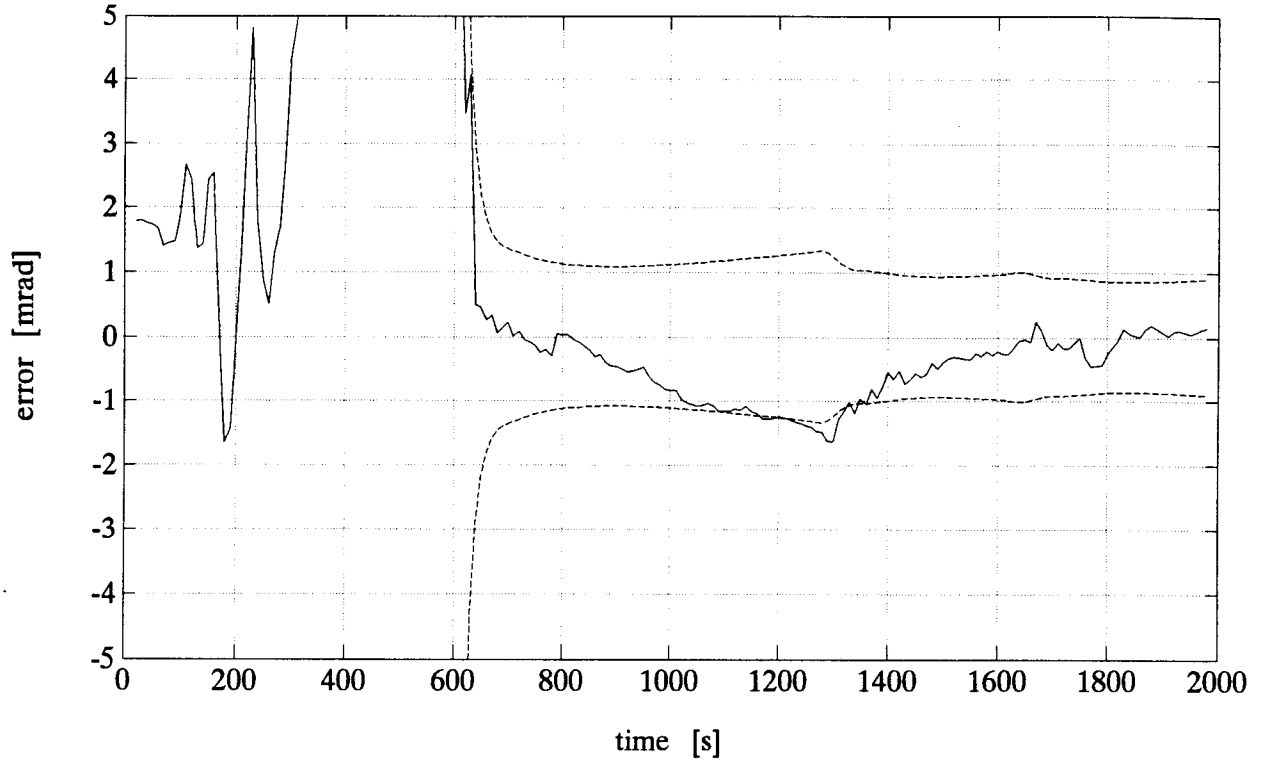
(4c-26) Error in Corrected x IMU Platform Alignment



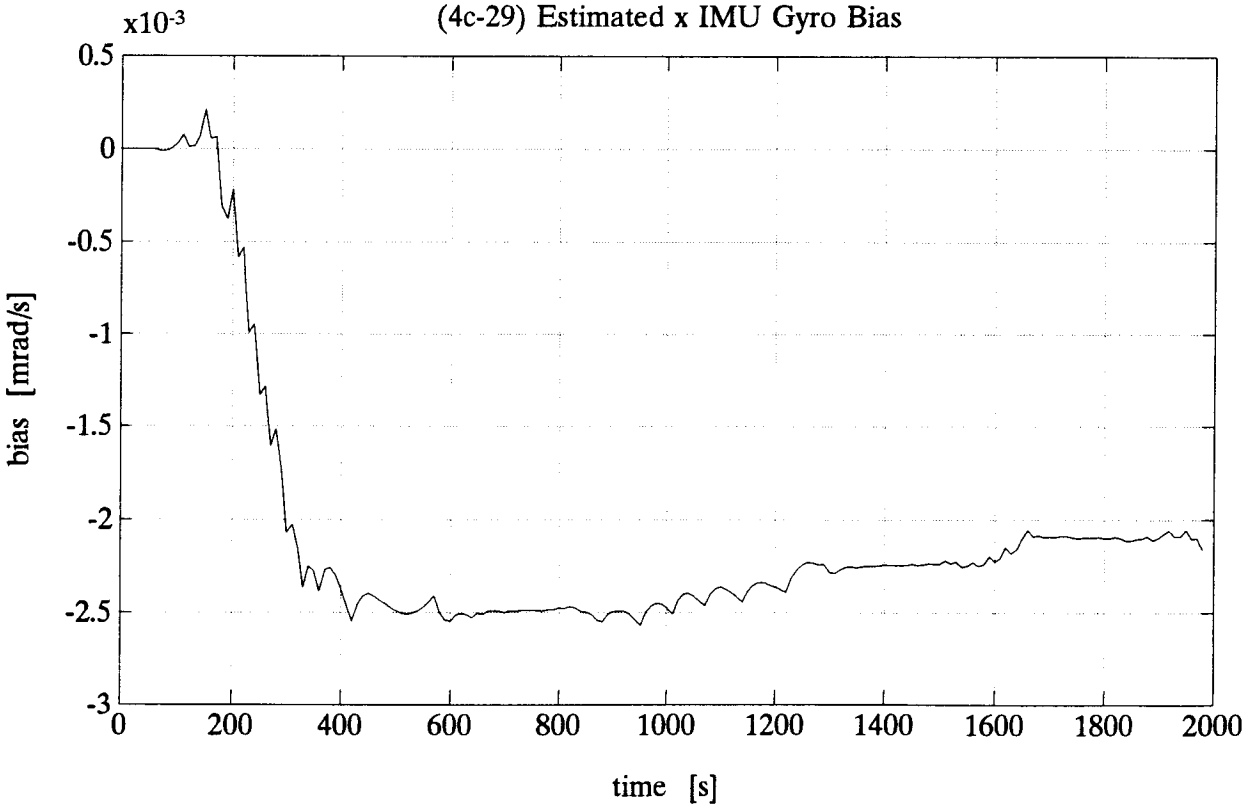
(4c-27) Error in Corrected y IMU Platform Alignment

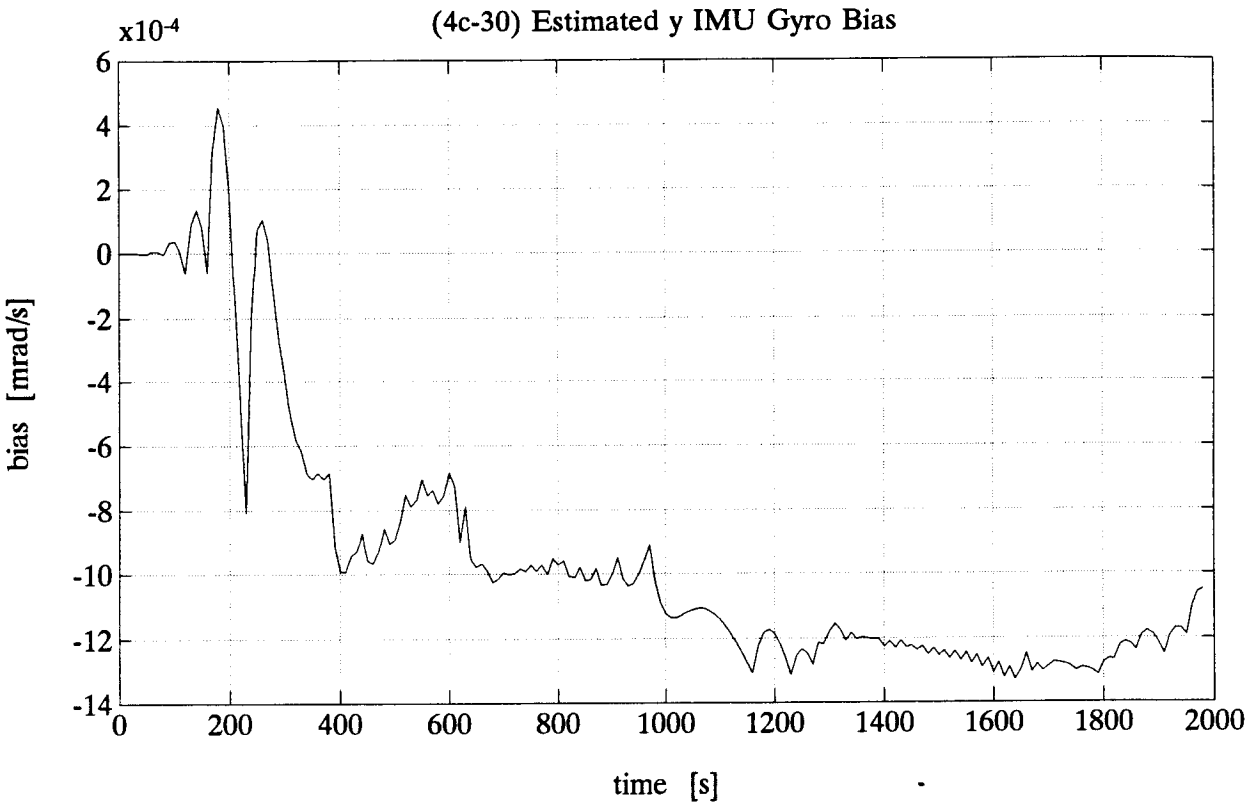


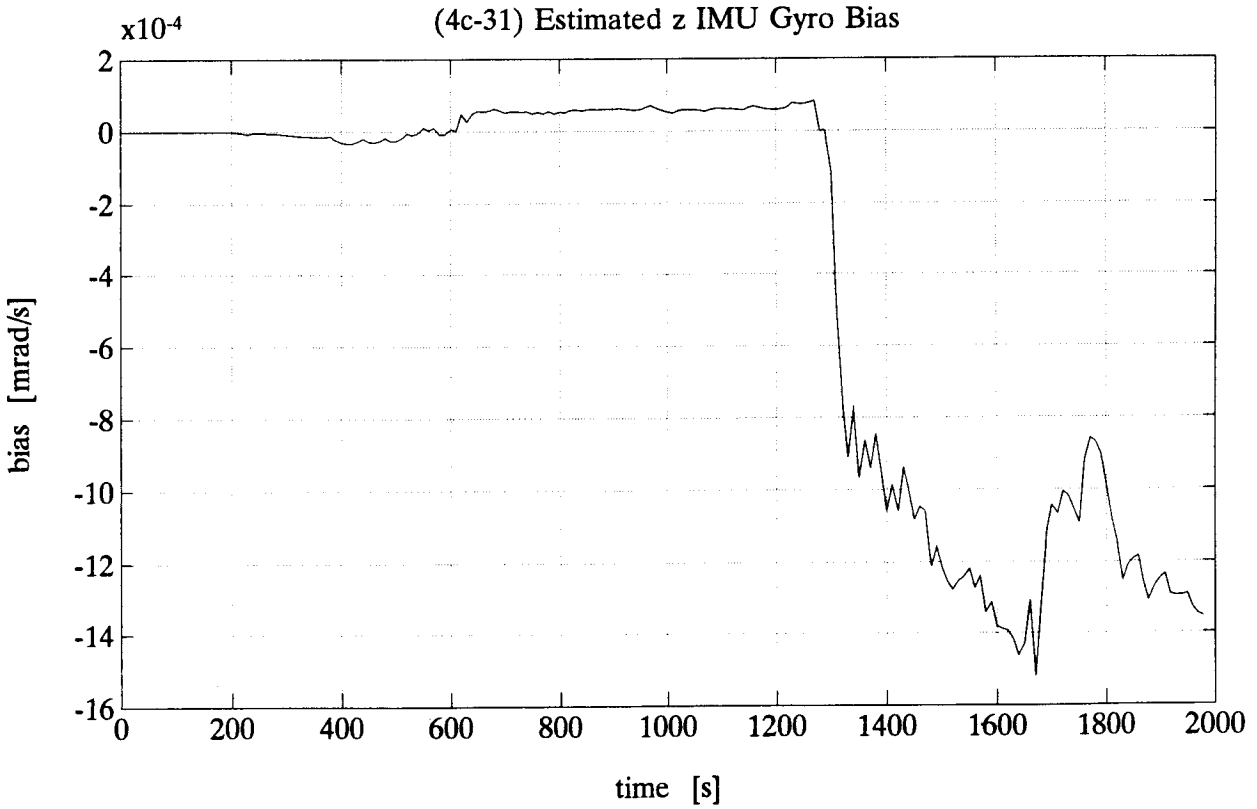
(4c-28) Error in Corrected z IMU Platform Alignment

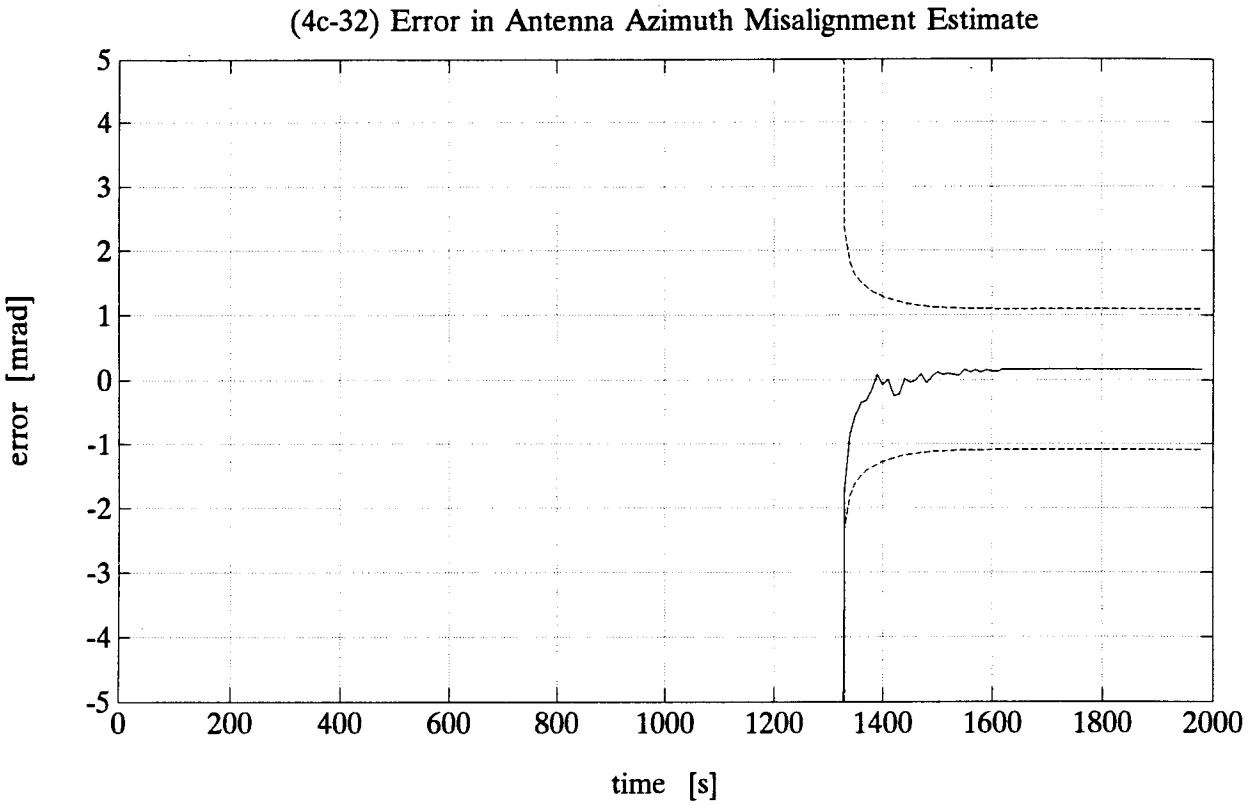


(4c-29) Estimated x IMU Gyro Bias

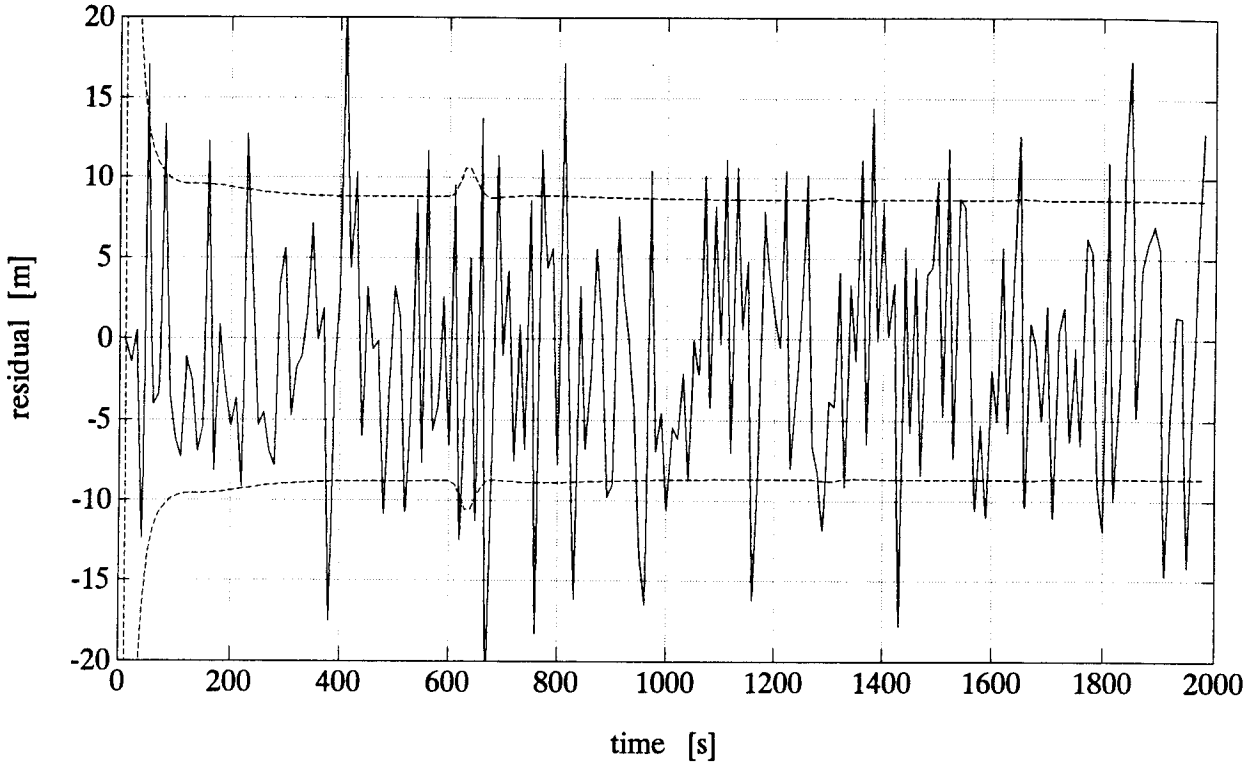



(4c-30) Estimated y IMU Gyro Bias
x10-4
6
4
2
0
-2
-4
-6
-8
-10
-12
-14
bias [mrad/s]
0
200
400
600
800
1000
1200
1400
1600
1800
2000
time [s]

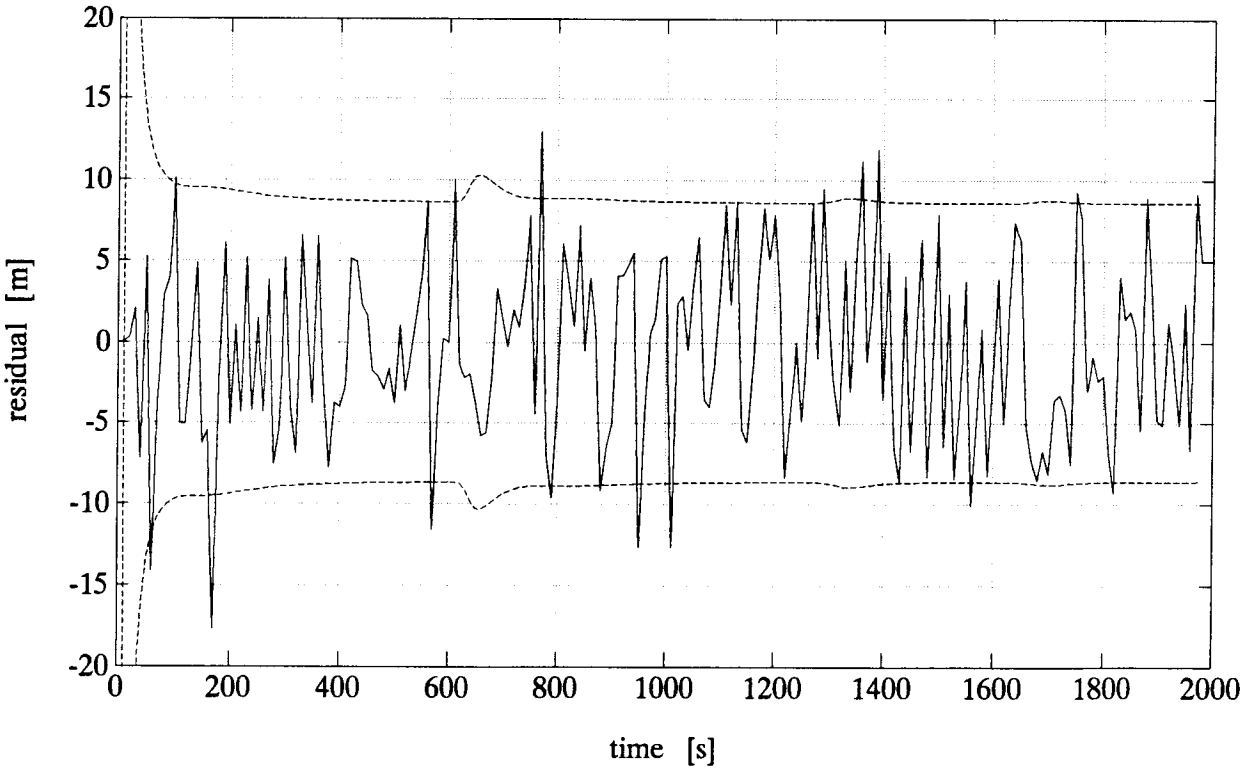

(4c-31) Estimated z IMU Gyro Bias
x10-4
2
0
-2
-4
-6
-8
-10
-12
-14
-16
bias [mrad/s]
0
200
400
600
800
1000
1200
1400
1600
1800
2000
time [s]


(4c-32) Error in Antenna Azimuth Misalignment Estimate
error [mrad]
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
200
400
600
800
1000
1200
1400
1600
1800
2000
time [s]

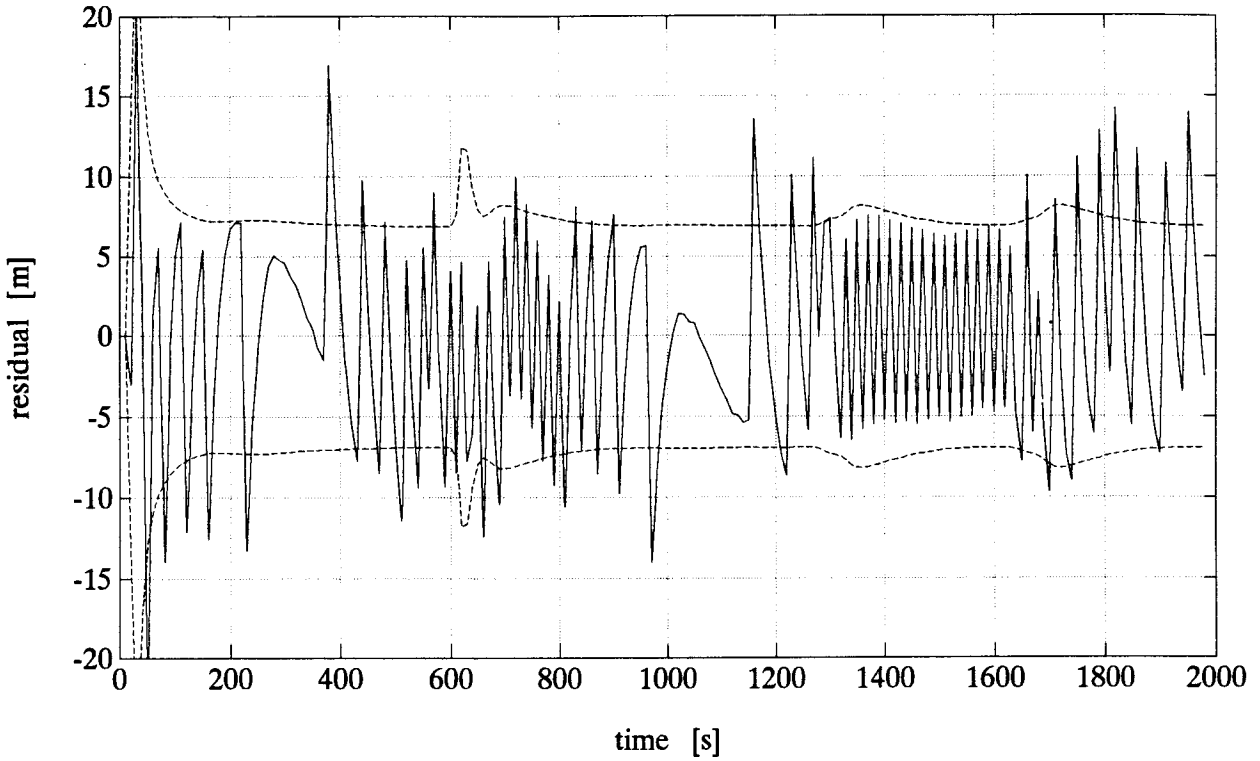
(4c-33) x INS/GPS Position Matching Measurement Residual



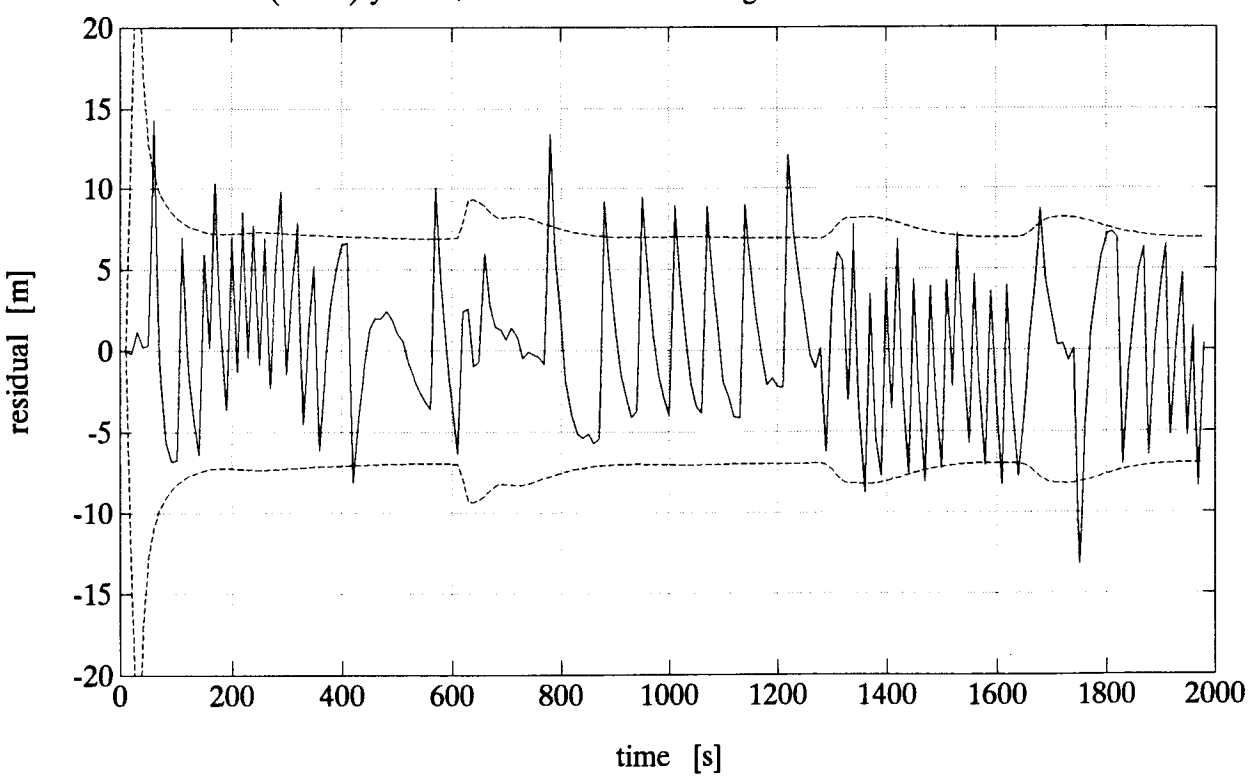
(4c-34) y INS/GPS Position Matching Measurement Residual



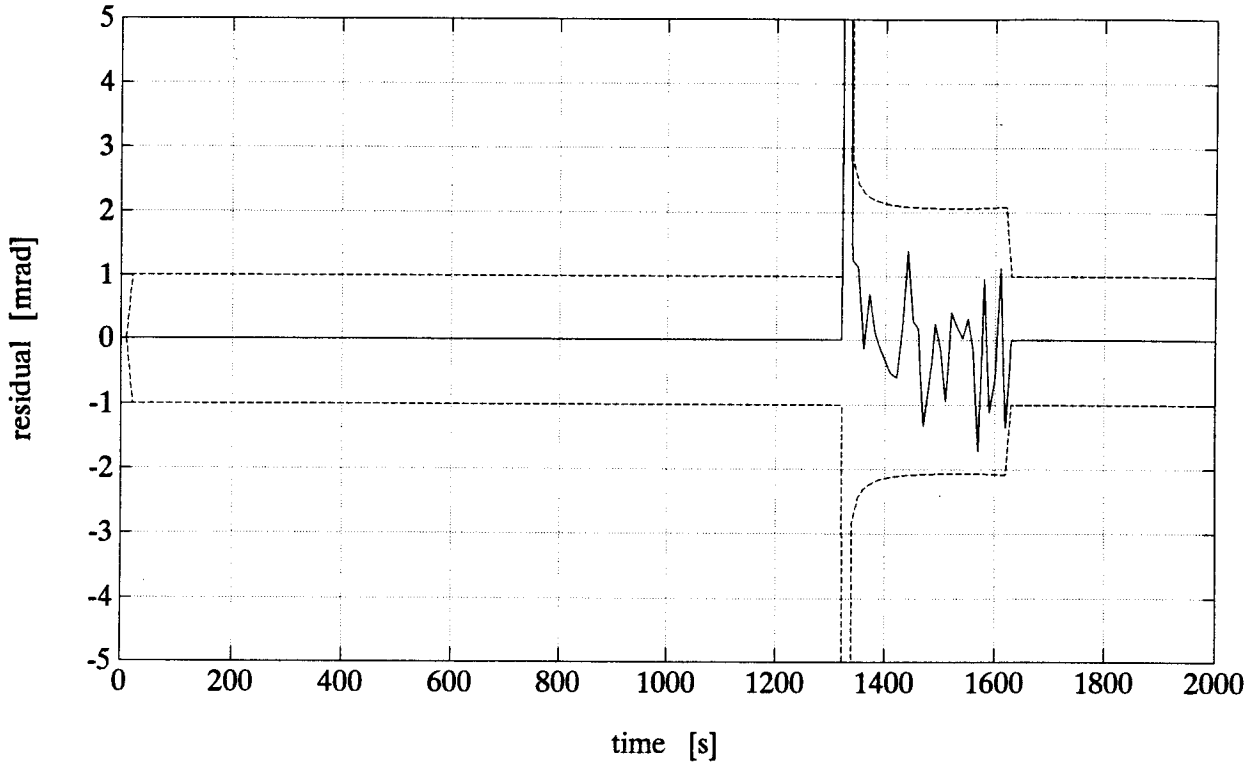
(4c-35) x IMU/INS Position Matching Measurement Residual



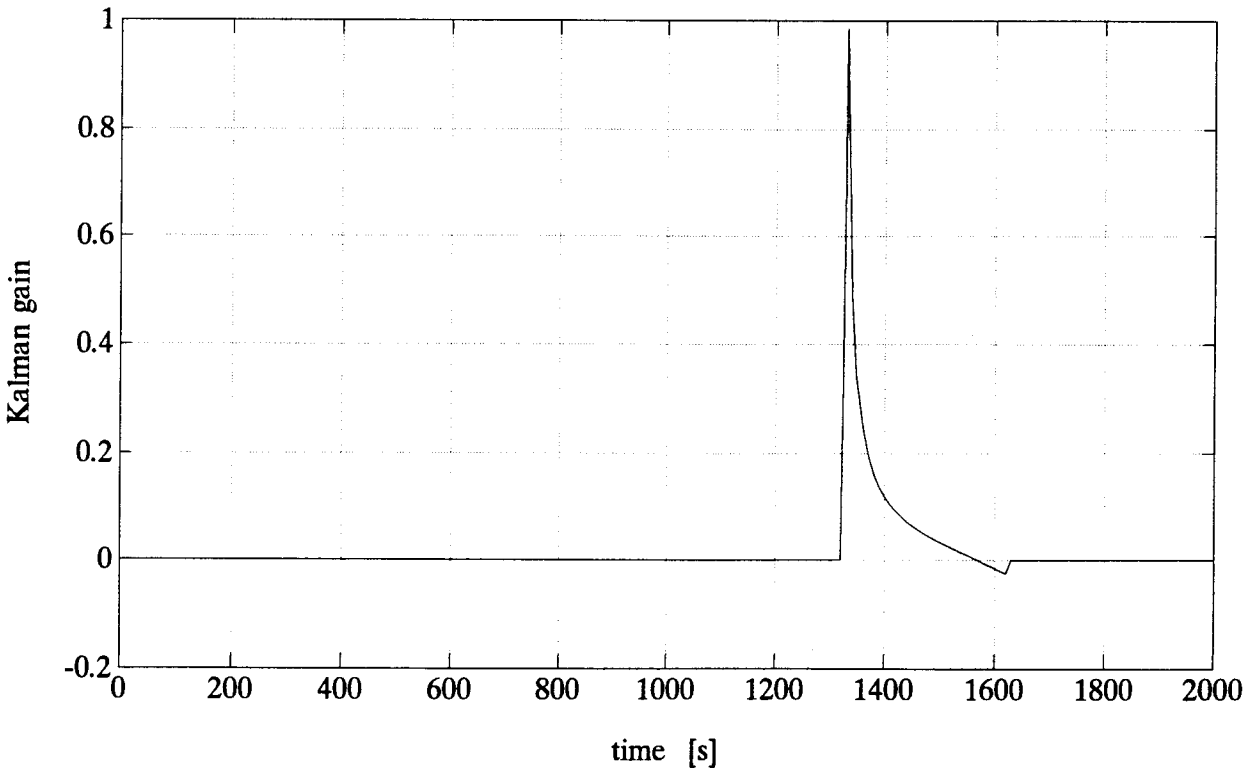
(4c-36) y IMU/INS Position Matching Measurement Residual



(4c-37) Antenna Azimuth Matching Measurement Residual



(4c-38) Antenna Azimuth Matching Kalman Gain



APPENDIX C

SOFTWARE LISTINGS

This appendix contains listings of the software modules comprising the data processing package of the simulator. The following modules are included:

- Antenna boresight calibration main program

  CALPRO.H
  CALPRO.CPP

- System model module

  CALMDL.H
  CALMDL.CPP

- Generic Kalman filter module

  KFLT.H
  PHIQD.CPP
  EXTRAP.CPP
  UPDAT.CPP

- Inertial navigation module

  NAVGTR.H
  NAVGTR.CPP

- Navigation library

  NAVLIB.H
  ERADII.CPP
  INVRADII.CPP
  GRAVITY.CPP
  RE2GEOD.CPP
  GEOD2RE.CPP
  EA2CBW.CPP
  EA2CWE.CPP
  EA2CEG.CPP
  EA2CGB.CPP
  CBW2EA.CPP
  CWE2EA.CPP
  DCM2Q.CPP
  QUPDT.CPP
  Q2DCM.CPP

```
// CALPRO.H

// IMU measurement record.
typedef struct {
  double time;
  double Dvel[3];
  double Dangl[3];
} imumes_record;

// INS measurement record.
typedef struct {
  double time;
  double latitude;
  double longitude;
  double gndspeed;
  double track;
  double heading;
  double pitch;
  double roll;
  double bdy_pitch_rate;
  double bdy_roll_rate;
  double bdy_yaw_rate;
  double plat_hdg;
  double altitude;
  double vertspeed;
  double N_velocity;
  double E_velocity;
} insmes_record;

// Baroaltitude measurement record;
typedef struct {
  double time;
  double baroalt;
} altmes_record;

// GPS measurement record.
typedef struct {
  double time;
  double latitude;
  double longitude;
  double altitude;
  double N_velocity;
  double E_velocity;
  double D_velocity;
} gpsmes_record;

// Azimuth encoder measurement record.
typedef struct {
  double time;
  double azimuth;
} azmmes_record;

// IMU navigation output record.
typedef struct {
  double time;
  double latitude;
  double longitude;
  double altitude;
  double gndspeed;
  double track;
  double heading;
  double azimuth;
  double pitch;
  double roll;
  double vertspeed;
```

```
} N00_record;

// State vector output record.
typedef struct {
  double time;
  double x[25];
} N01_record;

// State error variance output record.
typedef struct {
  double time;
  double Pdiag[25];
  double Poffd[7];
} N02_record;

// IMU gyro bias output record.
typedef struct {
  double time;
  double gb[3];
} N03_record;

// Measurement residual output record.
typedef struct {
  double time;
  double r[5];
} N04_record;

// Measurement residual variance output record.
typedef struct {
  double time;
  double Sdiag[5];
} N05_record;

// Kalman gain output record.
typedef struct {
  double time;
  double K[5];
} N06_record;

// Filtered acceleration and filtered heading rate output record.
typedef struct {
  double time;
  double gndspeed;
  double accel_f[3];
  double hdgrate_f;
} N07_record;
```

```
// CALPRO.CPP (main)
// Antenna azimuth calibration processing.
//
// CALPRO does not use any input or output parameters.  Instead, the required
// input files must be present in the default directory with the default
// names CALPRO.ext where ext is one of:
//
//   IMU = IMU measurement data.
//   INS = INS measurement data.
//   ALT = Baroaltitude measurement data.
//   GPS = GPS measurement data.
//   AZM = Azimuth encoder measurement data.
//
// Output files are produced with the default names CALPRO.ext where ext is
// one of:
//
//   $00 = corrected IMU navigation data.
//   $01 = state vector.
//   $02 = state error variances.
//   $03 = IMU gyro biases.
//   $04 = measurement residual vector.
//   $05 = measurement residual variances.
//   $06 = Kalman gain (elements directly affecting the antenna azimuth
//         misalignment estimate).
//   $07 = Filtered acceleration and filtered heading rate.


#include <fstream.h>
#include <stdlib.h>    // exit
#include <math.h>

#include "calpro.h"
#include "navgtr.h"
#include "navlib.h"
#include "calmdl.h"
#include "kflt.h"

#define TRUE 1
#define FALSE 0

struct inswork_record {
  double Rxinv;
  double Ryinv;
  double Tinv;
  double gravity;
  double wander;
  double Vw[3];
  double wiew[3];
  double weww[3];
  double fw[3];
  double Cbw[3][3];
  double Cwe[3][3];
};

void inscalc(long cycle_count, double Dtins, insmes_record insrec,
             inswork_record &inswrk, double accel_f[3], double &hdgrate_f);
void navcor(double naverr[10], double Dtimu,
            navgtr_record &navrec, navwork_record &navwrk,
    double gyro_bias[3]);

void main(void)
{
  const double target_lat=(44.0+52.0/60.0+6.0/3600.0)*M_PI/180.0;
  const double target_long=-(69.0+30.0/60.0+38.0/3600.0)*M_PI/180.0;
  const double target_alt=0.0*0.3048;
```

```
const double RRP[3]={6.0, -0.5, -0.1};
const double RSD[3]={0.0, 0.0, 0.0};
const double RGPS[3]={-3.0, -0.5, -2.0};

const double Dtimu=1.0/50.0;
const double Dtins=1.0/16.0;
const double Dtalt=1.0/2.0;
const double Dtgps=1.0;
const double Dtazm=10.0;

const double Dtupdt_min=0.75;
const double Dtupdt_max=1.5;

const int nTPHI=3;
const int nTQd=6;

ifstream imu_file;
ifstream ins_file;
ifstream alt_file;
ifstream gps_file;
ifstream azm_file;

ofstream N00_file;
ofstream N01_file;
ofstream N02_file;
ofstream N03_file;
ofstream N04_file;
ofstream N05_file;
ofstream N06_file;
ofstream N07_file;

imumes_record imurec;
insmes_record insrec;
altmes_record altrec;
gpsmes_record gpsrec;
azmmes_record azmrec;
navgtr_record navrec;
navwork_record navwrk;
inswork_record inswrk;
N00_record N00rec;
N01_record N01rec;
N02_record N02rec;
N03_record N03rec;
N04_record N04rec;
N05_record N05rec;
N06_record N06rec;
N07_record N07rec;

struct ready_record {
  int imu;
  int ins;
  int alt;
  int gps;
  int azm;
} rdyrec;

double TIME_lo;
double TIME_hi;
double start_time;
double end_time;
double cal_start_time;
double cal_end_time;
double cur_time;
double stamp_time;
double tau_time;
```

```
double kflt_time;
double Dtkflt;
long cycle_count;
long imu_cycle_count;
long ins_cycle_count;
long kflt_cycle_count;
int calibrate;
int calibrate_last=FALSE;
int gps_rdy;
int tau_rdy;
int kflt_go;

double altref;
double M;
double N;
double RM;
double RP;
double accel_f[3];
double hdgrate_f;
int insfw_count;
int imufw_count;
double insfw_avg[3];
double imufw_avg[3];
double Cgb[3][3];
double naverr[10];
double gyro_bias[3];

int nsys;
int mmes;
Kflt_vector x;
Kflt_vector z;
Kflt_vector r;
Kflt_matrix F;
Kflt_matrix Q;
Kflt_matrix PHI;
Kflt_matrix Qd;
Kflt_matrix H;
Kflt_matrix R;
Kflt_matrix S;
Kflt_matrix P;
Kflt_matrix K;
Kflt_tmp tmp;

int open_fail;
double a;
double b;
int i;
int j;

// Open the input files.
open_fail=FALSE;
imu_file.open("CALPRO.IMU", ios::nocreate|ios::binary);
if (!imu_file) {
  cerr << "Unable to open imu measurements file - CALPRO.IMU\n";
  open_fail=TRUE;
}
ins_file.open("CALPRO.INS", ios::nocreate|ios::binary);
if (!ins_file) {
  cerr << "Unable to open ins measurements file - CALPRO.INS\n";
  open_fail=TRUE;
}
alt_file.open("CALPRO.ALT", ios::nocreate|ios::binary);
if (!alt_file) {
  cerr << "Unable to open baroaltitude measurements file - CALPRO.ALT\n";
  open_fail=TRUE;
```

```
}
gps_file.open("CALPRO.GPS", ios::nocreate|ios::binary);
if (!gps_file) {
  cerr << "Unable to open gps measurements file - CALPRO.GPS\n";
  open_fail=TRUE;
}
azm_file.open("CALPRO.AZM", ios::nocreate|ios::binary);
if (!azm_file) {
  cerr << "Unable to open azimuth encoder measurements file - CALPRO.AZM\n";
  open_fail=TRUE;
}
if (open_fail)
  exit(EXIT_FAILURE);

// Create the output files.
N00_file.open("CALPRO.$00", ios::binary);
if (!N00_file) {
  cerr << "Unable to create output file - CALPRO.$00\n";
  exit(EXIT_FAILURE);
}
N01_file.open("CALPRO.$01", ios::binary);
if (!N01_file) {
  cerr << "Unable to create output file - CALPRO.$01\n";
  exit(EXIT_FAILURE);
}
N02_file.open("CALPRO.$02", ios::binary);
if (!N02_file) {
  cerr << "Unable to create output file - CALPRO.$02\n";
  exit(EXIT_FAILURE);
}
N03_file.open("CALPRO.$03", ios::binary);
if (!N03_file) {
  cerr << "Unable to create output file - CALPRO.$03\n";
  exit(EXIT_FAILURE);
}
N04_file.open("CALPRO.$04", ios::binary);
if (!N04_file) {
  cerr << "Unable to create output file - CALPRO.$04\n";
  exit(EXIT_FAILURE);
}
N05_file.open("CALPRO.$05", ios::binary);
if (!N05_file) {
  cerr << "Unable to create output file - CALPRO.$05\n";
  exit(EXIT_FAILURE);
}
N06_file.open("CALPRO.$06", ios::binary);
if (!N06_file) {
  cerr << "Unable to create output file - CALPRO.$06\n";
  exit(EXIT_FAILURE);
}
N07_file.open("CALPRO.$07", ios::binary);
if (!N07_file) {
  cerr << "Unable to create output file - CALPRO.$07\n";
  exit(EXIT_FAILURE);
}

// Get the start and end times.
azm_file.read((char *) &azmrec, sizeof(azmrec));
TIME_lo=azmrec.time-Dtazm;
azm_file.seekg(-1L*sizeof(azmrec), ios::end);
azm_file.read((char *) &azmrec, sizeof(azmrec));
TIME_hi=azmrec.time+Dtazm;
azm_file.clear();
azm_file.seekg(0L);
gps_file.read((char *) &gpsrec, sizeof(gpsrec));
```

```
 if (gpsrec.time-Dtgps>TIME_lo)
   TIME_lo=gpsrec.time-Dtgps;
 gps_file.seekg(-1L*sizeof(gpsrec), ios::end);
 gps_file.read((char *) &gpsrec, sizeof(gpsrec));
 if (gpsrec.time+Dtgps<TIME_hi)
   TIME_hi=gpsrec.time+Dtgps;
 gps_file.clear();
 gps_file.seekg(0L);
 alt_file.read((char *) &altrec, sizeof(altrec));
 if (altrec.time-Dtalt>TIME_lo)
   TIME_lo=altrec.time-Dtalt;
 alt_file.seekg(-1L*sizeof(altrec), ios::end);
 alt_file.read((char *) &altrec, sizeof(altrec));
 if (altrec.time+Dtalt<TIME_hi)
   TIME_hi=altrec.time+Dtalt;
 alt_file.clear();
 alt_file.seekg(0L);
 ins_file.read((char *) &insrec, sizeof(insrec));
 if (insrec.time-Dtins>TIME_lo)
   TIME_lo=insrec.time-Dtins;
 ins_file.seekg(-1L*sizeof(insrec), ios::end);
 ins_file.read((char *) &insrec, sizeof(insrec));
 if (insrec.time+Dtins<TIME_hi)
   TIME_hi=insrec.time+Dtins;
 ins_file.clear();
 ins_file.seekg(0L);
 imu_file.read((char *) &imurec, sizeof(imurec));
 if (imurec.time>TIME_lo)
   TIME_lo=imurec.time;
 imu_file.seekg(-1L*sizeof(imurec), ios::end);
 imu_file.read((char *) &imurec, sizeof(imurec));
 if (imurec.time<TIME_hi)
   TIME_hi=imurec.time;
 imu_file.clear();
 imu_file.seekg(0L);
 cout << "\n";
 do {
   cout << "Enter processing start time between "
        << TIME_lo << " and " << TIME_hi << ": ";
   cin >> start_time;
 } while (start_time<TIME_lo || start_time>=TIME_hi);
 cout << "\n";
 do {
   cout << "Enter processing end time between "
        << start_time << " and " << TIME_hi << ": ";
   cin >> end_time;
 } while (end_time<start_time || end_time>TIME_hi);
 cout << "\n";
 do {
   cout << "Enter calibration start time greater than or equal to "
        << start_time << ": ";
   cin >> cal_start_time;
 } while (cal_start_time<start_time);
 cout << "\n";
 if (cal_start_time<end_time) {
   do {
     cout << "Enter calibration end time between "
          << cal_start_time << " and " << end_time << ": ";
     cin >> cal_end_time;
   } while (cal_end_time<cal_start_time || cal_end_time>end_time);
   cout << "\n";
 }
 else {
   cal_start_time=end_time;
   cal_end_time=cal_start_time;
```

```
}

// Main loop.
imu_cycle_count=0L;
ins_cycle_count=0L;
kflt_cycle_count=0L;
cycle_count=0L;
cur_time=start_time;
stamp_time=cur_time;
calibrate=FALSE;
while (cur_time<end_time) {
  if (cur_time>=stamp_time) {
    cout << cur_time << "\tTime stamp\n";
    stamp_time=stamp_time+100.0;
  }

  // Get the sensor measurement data.
  if (cycle_count==0L) {
    rdyrec.imu=FALSE;
    rdyrec.ins=FALSE;
    rdyrec.alt=FALSE;
    rdyrec.gps=FALSE;
    rdyrec.azm=FALSE;
    do {
      ins_file.read((char *) &insrec, sizeof(insrec));
    } while (insrec.time<start_time);
    do {
      imu_file.read((char *) &imurec, sizeof(imurec));
    } while (imurec.time<insrec.time);
    start_time=imurec.time;
    do {
      alt_file.read((char *) &altrec, sizeof(altrec));
    } while (altrec.time<=start_time);
    do {
      gps_file.read((char *) &gpsrec, sizeof(gpsrec));
    } while (gpsrec.time<=start_time);
    do {
      azm_file.read((char *) &azmrec, sizeof(azmrec));
    } while (azmrec.time<=start_time);
  }
  else {
    if (rdyrec.imu) {
      rdyrec.imu=FALSE;
      imu_file.read((char *) &imurec, sizeof(imurec));
    }
    if (rdyrec.ins) {
      rdyrec.ins=FALSE;
      ins_file.read((char *) &insrec, sizeof(insrec));
    }
    if (rdyrec.alt) {
      rdyrec.alt=FALSE;
      alt_file.read((char *) &altrec, sizeof(altrec));
    }
    if (rdyrec.gps) {
      rdyrec.gps=FALSE;
      gps_file.read((char *) &gpsrec, sizeof(gpsrec));
    }
    if (rdyrec.azm) {
      rdyrec.azm=FALSE;
      azm_file.read((char *) &azmrec, sizeof(azmrec));
    }
  }
  rdyrec.imu=TRUE;
  cur_time=imurec.time;
  if (insrec.time<=cur_time)
```

```
    rdyrec.ins=TRUE;
  if (altrec.time<=cur_time)
    rdyrec.alt=TRUE;
  if (gpsrec.time<=cur_time)
    rdyrec.gps=TRUE;
  if (azmrec.time<=cur_time)
    rdyrec.azm=TRUE;

  // Process the INS measurements.
  if (rdyrec.ins) {
    inscalc(ins_cycle_count, Dtins, insrec, inswrk, accel_f, hdgrate_f);
    if (kflt_cycle_count>0L) {
      insfw_count=insfw_count+1;
      b=1.0/insfw_count;
      a=1.0-b;
      for (i=0; i<3; i++)
        insfw_avg[i]=a*insfw_avg[i]+b*inswrk.fw[i];
    }
    ins_cycle_count=ins_cycle_count+1L;
  }

  // Process the baroaltitude measurements.
  if (rdyrec.alt)
    altref=altrec.baroalt;

  // Process the IMU measurements.
  if (rdyrec.imu) {
    if (imu_cycle_count==0L) {
      Ea2Cgb(insrec.roll, insrec.pitch, insrec.heading, Cgb);
      eradii(insrec.latitude, M , N);
      RM=M+insrec.altitude;
      RP=(N+insrec.altitude)*cos(insrec.latitude);
      navrec.time=insrec.time;
      navrec.latitude=insrec.latitude
        +((Cgb[0][0]*RRP[0]+Cgb[1][0]*RRP[1]+Cgb[2][0]*RRP[2])
+(RSD[0]*cos(insrec.heading)-RSD[1]*sin(insrec.heading)))/RM;
      navrec.longitude=insrec.longitude
        +((Cgb[0][1]*RRP[0]+Cgb[1][1]*RRP[1]+Cgb[2][1]*RRP[2])
+(RSD[0]*sin(insrec.heading)+RSD[1]*cos(insrec.heading)))/RP;
      navrec.altitude=insrec.altitude
        -((Cgb[0][2]*RRP[0]+Cgb[1][2]*RRP[1]+Cgb[2][2]*RRP[2])+RSD[2]);
      navrec.gndspeed=insrec.gndspeed;
      navrec.track=insrec.track;
      navrec.heading=insrec.heading;
      navrec.azimuth=insrec.plat_hdg;
      navrec.pitch=0.0;
      navrec.roll=0.0;
      navrec.vertspeed=insrec.vertspeed;
      altref=navrec.altitude;
      for (i=0; i<3; i++)
        gyro_bias[i]=0.0;
    }
    for (i=0; i<3; i++)
      imurec.Dangl[i]=imurec.Dangl[i]-gyro_bias[i]*Dtimu;
    navgtr(imu_cycle_count, cur_time, Dtimu,
           imurec.Dvel, imurec.Dangl, altref, navrec, navwrk);
    if (kflt_cycle_count>0L) {
      imufw_count=imufw_count+1;
      b=1.0/imufw_count;
      a=1.0-b;
      for (i=0; i<3; i++)
        imufw_avg[i]=a*imufw_avg[i]+b*navwrk.fw[i];
    }
    imu_cycle_count=imu_cycle_count+1L;
  }
```

```
// Kalman filter timing.
// - Allow use of all azimuth measurements that occur between the
//   calibration start and end times.
// - If not calibrating, allow use of all GPS measurements.
// - If calibrating, ignore GPS measurements when an azimuth record is
//   expected.
// - Ensure Kalman filter updates are appropriately separated in time
//   (not too close together and not too far apart).
tau_rdy=FALSE;
if (rdyrec.azm)
  if (azmrec.time>=cal_start_time && azmrec.time<cal_end_time) {
    calibrate=TRUE;
    tau_rdy=TRUE;
    tau_time=azmrec.time;
  }
  else
    calibrate=FALSE;

gps_rdy=FALSE;
if (rdyrec.gps)
  if (!calibrate)
    gps_rdy=TRUE;
  else
    if ((gpsrec.time-tau_time)<=(Dtazm-0.75*Dtgps) || tau_rdy)
      gps_rdy=TRUE;

kflt_go=FALSE;
if (kflt_cycle_count==0L) {
  if (gps_rdy || tau_rdy) {
    kflt_go=TRUE;
    if (tau_rdy)
      kflt_time=azmrec.time;
    else
      kflt_time=gpsrec.time;
  }
}
else {
  if ((gps_rdy || tau_rdy || (cur_time-kflt_time)>=Dtupdt_max) &&
      (cur_time-kflt_time)>=Dtupdt_min) {
    kflt_go=TRUE;
    if (tau_rdy) {
      Dtkflt=azmrec.time-kflt_time;
      kflt_time=azmrec.time;
    }
    else if (gps_rdy) {
      Dtkflt=gpsrec.time-kflt_time;
      kflt_time=gpsrec.time;
    }
    else {
      Dtkflt=cur_time-kflt_time;
      kflt_time=cur_time;
    }
  }
}

// Perform the antenna azimuth calibration Kalman filtering.
if (kflt_go) {
  if (kflt_cycle_count==0L) {
    calmdl_wrk(nsys, mmes, x, z, r, F, Q, PHI, Qd, H, R, S, P, K, tmp);
    calmdl_x0P0();
  }
  else {
    if (!calibrate && calibrate_last) {
      for (i=0; i<14; i++)
        for (j=14; j<25; j++) {
```

```
            P[i][j]=0.0;
            P[j][i]=0.0;
          }
        for (i=0; i<24; i++) {
          P[i][24]=0.0;
          P[24][i]=0.0;
        }
      }
      calibrate_last=calibrate;
      naverr[0]=x[0]+x[14];
      naverr[1]=x[1]+x[15];
      naverr[2]=x[2]+x[16];
      naverr[3]=x[3]+x[17];
      naverr[4]=x[4]+x[18];
      naverr[5]=x[5]+x[19];
      naverr[6]=x[6]+x[20];
      naverr[7]=x[21];
      naverr[8]=x[22];
      naverr[9]=x[23];
      navcor(naverr, Dtimu, navrec, navwrk, gyro_bias);
      calmdl_mes(insrec.latitude, insrec.longitude, insrec.altitude,
            inswrk.Cwe, inswrk.Cbw,
         navrec.latitude, navrec.longitude, navrec.altitude,
         navrec.heading, navwrk.wander,
 navwrk.Cwe, navwrk.Cbw,
         gps_rdy, gpsrec.latitude, gpsrec.longitude,
 gpsrec.altitude,
                  tau_rdy, azmrec.azimuth,
 RRP, RSD, RGPS, target_lat, target_long, target_alt);
      calmdl_FQ(inswrk.Rxinv, inswrk.Ryinv,
            inswrk.Vw, inswrk.wiew, inswrk.weww,
                insfw_avg, inswrk.Cbw,
                navwrk.Rxinv, navwrk.Ryinv,
                navwrk.Vw, navwrk.wiew, navwrk.weww,
imufw_avg, navwrk.Cbw,
                insrec.gndspeed, accel_f, hdgrate_f);
      PHIQd(nsys, nTPHI, nTQd, Dtkflt, F, Q, PHI, Qd, tmp);
      extrap(nsys, PHI, Qd, x, P, tmp);
      x[14]=x[14]-naverr[0];
      x[15]=x[15]-naverr[1];
      x[16]=x[16]-naverr[2];
      x[17]=x[17]-naverr[3];
      x[18]=x[18]-naverr[4];
      x[19]=x[19]-naverr[5];
      x[20]=x[20]-naverr[6];
      x[21]=x[21]-naverr[7];
      x[22]=x[22]-naverr[8];
      x[23]=x[23]-naverr[9];
      updat(nsys, mmes, z, H, R, r, S, K, x, P, tmp);
    }

    // Output the results.
    NOOrec.time=kflt_time;
    NOOrec.latitude=navrec.latitude;
    NOOrec.longitude=navrec.longitude;
    NOOrec.altitude=navrec.altitude;
    NOOrec.gndspeed=navrec.gndspeed;
    NOOrec.track=navrec.track;
    NOOrec.heading=navrec.heading;
    NOOrec.azimuth=navrec.azimuth;
    NOOrec.pitch=navrec.pitch;
    NOOrec.roll=navrec.roll;
    NOOrec.vertspeed=navrec.vertspeed;
    if (!NOO_file.write((char *) &NOOrec, sizeof(NOOrec))) {
      cerr << "Write error on output file - CALPRO.$00\n";
```

```
        exit(EXIT_FAILURE);
      }
      N01rec.time=kflt_time;
      for (i=0; i<nsys; i++)
        N01rec.x[i]=x[i];
      if (!N01_file.write((char *) &N01rec, sizeof(N01rec))) {
        cerr << "Write error on output file - CALPRO.$01\n";
        exit(EXIT_FAILURE);
      }
      N02rec.time=kflt_time;
      for (i=0; i<nsys; i++)
        N02rec.Pdiag[i]=P[i][i];
      for (i=0; i<7; i++)
        N02rec.Poffd[i]=P[0+i][14+i];
      if (!N02_file.write((char *) &N02rec, sizeof(N02rec))) {
        cerr << "Write error on output file - CALPRO.$02\n";
        exit(EXIT_FAILURE);
      }
      N03rec.time=kflt_time;
      for (i=0; i<3; i++)
        N03rec.gb[i]=gyro_bias[i];
      if (!N03_file.write((char *) &N03rec, sizeof(N03rec))) {
        cerr << "Write error on output file - CALPRO.$03\n";
        exit(EXIT_FAILURE);
      }
      N04rec.time=kflt_time;
      for (i=0; i<mmes; i++)
        N04rec.r[i]=r[i];
      if (!N04_file.write((char *) &N04rec, sizeof(N04rec))) {
        cerr << "Write error on output file - CALPRO.$04\n";
        exit(EXIT_FAILURE);
      }
      N05rec.time=kflt_time;
      for (i=0; i<mmes; i++)
        N05rec.Sdiag[i]=S[i][i];
      if (!N05_file.write((char *) &N05rec, sizeof(N05rec))) {
        cerr << "Write error on output file - CALPRO.$05\n";
        exit(EXIT_FAILURE);
      }
      N06rec.time=kflt_time;
      for (i=0; i<mmes; i++)
        N06rec.K[i]=K[24][i];
      if (!N06_file.write((char *) &N06rec, sizeof(N06rec))) {
        cerr << "Write error on output file - CALPRO.$06\n";
        exit(EXIT_FAILURE);
      }
      N07rec.time=kflt_time;
      N07rec.gndspeed=insrec.gndspeed;
      for (i=0; i<3; i++)
        N07rec.accel_f[i]=accel_f[i];
      N07rec.hdgrate_f=hdgrate_f;
      if (!N07_file.write((char *) &N07rec, sizeof(N07rec))) {
        cerr << "Write error on output file - CALPRO.$07\n";
        exit(EXIT_FAILURE);
      }

      insfw_count=0;
      imufw_count=0;
      kflt_cycle_count=kflt_cycle_count+1L;
    }
    cycle_count=cycle_count+1L;
  }

  // Normal exit.
  imu_file.close();
```

```
  ins_file.close();
  alt_file.close();
  gps_file.close();
  azm_file.close();
  N00_file.close();
  N01_file.close();
  N02_file.close();
  N03_file.close();
  N04_file.close();
  N05_file.close();
  N06_file.close();
  N07_file.close();
  exit(EXIT_SUCCESS);
}


// Local function to calculate some required INS navigation quantities.
//
void inscalc(long cycle_count, double Dt, insmes_record n, inswork_record &w,
             double accel_f[3], double &hdgrate_f)
{
  const double tau=10.0;
  const double b=Dt/tau;
  const double a=1.0-b;

  double DVw[3];
  double vectmp[3];

  Ea2Cbw(n.roll, n.pitch, n.plat_hdg, w.Cbw);

  w.wander=n.plat_hdg-n.heading;
  if (w.wander>=M_PI)
    w.wander=w.wander-2.0*M_PI;
  else if (w.wander<-M_PI)
    w.wander=w.wander+2.0*M_PI;
  Ea2Cwe(n.latitude, n.longitude, w.wander, w.Cwe);

  vectmp[0]=w.Vw[0];
  vectmp[1]=w.Vw[1];
  vectmp[2]=w.Vw[2];
  w.Vw[0]=n.gndspeed*cos(n.track+w.wander);
  w.Vw[1]=-n.gndspeed*sin(n.track+w.wander);
  w.Vw[2]=n.vertspeed;
  if (cycle_count==0L) {
    DVw[0]=0.0;
    DVw[1]=0.0;
    DVw[2]=0.0;
  }
  else {
    DVw[0]=w.Vw[0]-vectmp[0];
    DVw[1]=w.Vw[1]-vectmp[1];
    DVw[2]=w.Vw[2]-vectmp[2];
  }

  invradii1(w.Cwe, n.altitude, w.Rxinv, w.Ryinv, w.Tinv);
  w.weww[0]=-w.Vw[1]*w.Ryinv+w.Vw[0]*w.Tinv;
  w.weww[1]=w.Vw[0]*w.Rxinv-w.Vw[1]*w.Tinv;
  w.weww[2]=0.0;

  w.wiew[0]=w.Cwe[0][0]*E_wie;
  w.wiew[1]=w.Cwe[0][1]*E_wie;
  w.wiew[2]=w.Cwe[0][2]*E_wie;

  w.gravity=gravity1(w.Cwe, n.altitude);
```

```
  vectmp[0]=2.0*w.wiew[0]+w.weww[0];
  vectmp[1]=2.0*w.wiew[1]+w.weww[1];
  vectmp[2]=2.0*w.wiew[2]+w.weww[2];
  w.fw[0]=DVw[0]/Dt+(vectmp[1]*w.Vw[2]-vectmp[2]*w.Vw[1]);
  w.fw[1]=DVw[1]/Dt+(vectmp[2]*w.Vw[0]-vectmp[0]*w.Vw[2]);
  w.fw[2]=DVw[2]/Dt+w.gravity+(vectmp[0]*w.Vw[1]-vectmp[1]*w.Vw[0]);

  if (cycle_count==0L) {
    accel_f[0]=w.fw[0];
    accel_f[1]=w.fw[1];
    accel_f[2]=w.fw[2]-w.gravity;
    hdgrate_f=(n.bdy_pitch_rate*sin(n.roll)
              +n.bdy_yaw_rate*cos(n.roll))/cos(n.pitch);
  }
  else {
    accel_f[0]=a*accel_f[0]+b*w.fw[0];
    accel_f[1]=a*accel_f[1]+b*w.fw[1];
    accel_f[2]=a*accel_f[2]+b*(w.fw[2]-w.gravity);
    hdgrate_f=a*hdgrate_f+b*(n.bdy_pitch_rate*sin(n.roll)
                            +n.bdy_yaw_rate*cos(n.roll))/cos(n.pitch);
  }

  return;
}


// Local function to correct the IMU navigation information using the
// estimated errors.
//
void navcor(double naverr[10], double Dt,
            navgtr_record &n, navwork_record &w, double gb[3])
{
  double q[4];
  double u[4];
  double vectmp[3];

  // Correct the wander to earth quaternion (and related quantities).
  // q=q+qu where q is the quaternion qwe and u is the quaternion
  // corresponding to the vector -0.5*delta_theta.
  q[0]=w.qwe[0];
  q[1]=w.qwe[1];
  q[2]=w.qwe[2];
  q[3]=w.qwe[3];
  u[0]=0.0;
  u[1]=0.5*naverr[1]*w.Ryinv;
  u[2]=-0.5*naverr[0]*w.Rxinv;
  u[3]=0.0;
  w.qwe[0]=q[0]-q[1]*u[1]-q[2]*u[2]-q[3]*u[3];
  w.qwe[1]=q[1]+q[0]*u[1]+q[2]*u[3]-q[3]*u[2];
  w.qwe[2]=q[2]+q[0]*u[2]-q[1]*u[3]+q[3]*u[1];
  w.qwe[3]=q[3]+q[0]*u[3]+q[1]*u[2]-q[2]*u[1];

  q2dcm(w.qwe, w.Cwe);

  w.wiew[0]=w.Cwe[0][0]*E_wie;
  w.wiew[1]=w.Cwe[0][1]*E_wie;
  w.wiew[2]=w.Cwe[0][2]*E_wie;

  invradii1(w.Cwe, n.altitude, w.Rxinv, w.Ryinv, w.Tinv);

  w.gravity=gravity1(w.Cwe, n.altitude);

  Cwe2Ea(w.Cwe, n.latitude, n.longitude, w.wander);

  // Correct the velocity (and related quantities).
```

```
  w.Vw[0]=w.Vw[0]-naverr[2];
  w.Vw[1]=w.Vw[1]-naverr[3];

  w.weww[0]=-w.Vw[1]*w.Ryinv+w.Vw[0]*w.Tinv;
  w.weww[1]=w.Vw[0]*w.Rxinv-w.Vw[1]*w.Tinv;
  w.weww[2]=0.0;

  n.gndspeed=sqrt(w.Vw[0]*w.Vw[0]+w.Vw[1]*w.Vw[1]);
  n.track=atan2(-w.Vw[1], w.Vw[0])-w.wander;
  if (n.track>=M_PI)
    n.track=n.track-2.0*M_PI;
  else if (n.track<-M_PI)
    n.track=n.track+2.0*M_PI;

  // Correct the body to wander quaternion (and related quantities).
  // q=q+uq where q is the quaternion qbw and u is the quaternion
  // corresponding to the vector 0.5*phi.
  q[0]=w.qbw[0];
  q[1]=w.qbw[1];
  q[2]=w.qbw[2];
  q[3]=w.qbw[3];
  u[0]=0.0;
  u[1]=(0.5*naverr[4])/1000.0;
  u[2]=(0.5*naverr[5])/1000.0;
  u[3]=(0.5*naverr[6])/1000.0;
  w.qbw[0]=q[0]-u[1]*q[1]-u[2]*q[2]-u[3]*q[3];
  w.qbw[1]=q[1]+u[1]*q[0]+u[2]*q[3]-u[3]*q[2];
  w.qbw[2]=q[2]-u[1]*q[3]+u[2]*q[0]+u[3]*q[1];
  w.qbw[3]=q[3]+u[1]*q[2]-u[2]*q[1]+u[3]*q[0];

  q2dcm(w.qbw, w.Cbw);

  vectmp[0]=(w.wiew[0]+w.weww[0])*Dt;
  vectmp[1]=(w.wiew[1]+w.weww[1])*Dt;
  vectmp[2]=(w.wiew[2]+w.weww[2])*Dt;
  w.Dthiwb[0]=w.Cbw[0][0]*vectmp[0]+w.Cbw[1][0]*vectmp[1]
              +w.Cbw[2][0]*vectmp[2];
  w.Dthiwb[1]=w.Cbw[0][1]*vectmp[0]+w.Cbw[1][1]*vectmp[1]
              +w.Cbw[2][1]*vectmp[2];
  w.Dthiwb[2]=w.Cbw[0][2]*vectmp[0]+w.Cbw[1][2]*vectmp[1]
              +w.Cbw[2][2]*vectmp[2];

  Cbw2Ea(w.Cbw, n.roll, n.pitch, n.azimuth);

  n.heading=n.azimuth-w.wander;
  if (n.heading>=M_PI)
    n.heading=n.heading-2.0*M_PI;
  else if (n.heading<-M_PI)
    n.heading=n.heading+2.0*M_PI;

  // Update the gyro bias.
  gb[0]=gb[0]+naverr[7]/1000.0;
  gb[1]=gb[1]+naverr[8]/1000.0;
  gb[2]=gb[2]+naverr[9]/1000.0;

  return;
}
```

```
// CALMDL.H

#ifndef __KFLT_H
#include "kflt.h"
#endif


// *****
// Provide access to the working data store.
// *****
void calmdl_wrk(int &nsys, int &mmes,
                Kflt_vector &x, Kflt_vector &z, Kflt_vector &r,
                Kflt_matrix &F, Kflt_matrix &Q,
        Kflt_matrix &PHI, Kflt_matrix &Qd,
        Kflt_matrix &H, Kflt_matrix &R, Kflt_matrix &S,
        Kflt_matrix &P, Kflt_matrix &K, Kflt_tmp &tmp);


// *****
// Initialize the state estimate vector (x) and the error covariance
// matrix (P).
// *****
//
void calmdl_x0P0(void);


// *****
// Construct the system dynamics matrix (F) and the process noise spectral
// density matrix (Q).
// *****
//
void calmdl_FQ(double insRxinv, double insRyinv,
       double insVw[3], double inswiew[3], double insweww[3],
       double insfw[3], double insCbw[3][3],
               double imuRxinv, double imuRyinv,
       double imuVw[3], double imuwiew[3], double imuweww[3],
       double imufw[3], double imuCbw[3][3],
       double gndspeed, double accel[3], double hdgrate);


// *****
// Construct the measurement vector (z), the measurement matrix (H) and the
// measurement noise covariance matrix (R).
// *****
//
void calmdl_mes(double inslat, double inslong, double insalt,
        double insCwe[3][3], double insCbw[3][3],
        double imulat, double imulong, double imualt,
double imuhdg, double imuwan,
double imuCwe[3][3], double imuCbw[3][3],
        int gpsrdy, double gpslat, double gpslong, double gpsalt,
int taurdy, double tauazm,
const double RRP[3], const double RSD[3],
const double RGPS[3],
double tgtlat, double tgtlong, double tgtalt);
```

```
// CALMDL.CPP
//
// The following variables are defined and allocated storage:
//
//     x = State estimate vector.
//     z = Measurement vector.
//     r = Residual vector.
//     F = System dynamics matrix.
//     Q = Process noise spectral density matrix.
//   PHI = State transistion matrix.
//    Qd = Process noise covariance matrix.
//     H = Measurement matrix.
//     R = Measurement noise covariance matrix.
//     S = Residual covariance matrix.
//     P = Error covariance matrix.
//     K = Kalman gain matrix.
//   tmp = Temporary storage required by the Kalman filter functions.
//
//
// The following functions are provided:
//
//    _wrk - Provides access to the working data store.
//   _x0P0 - Initializes x and P.
//     _FQ - Constructs F and Q.
//    _mes - Constructs z, H and R.


#include <math.h>

#include "calmdl.h"
#include "navlib.h"

#define NSYS 25  // Number of states.
#define MMES 5   // Maximum number of measurements.

static double x1[NSYS];

static double z1[MMES];

static double r1[MMES];

static double F2[NSYS][NSYS];
static double *F1[NSYS];

static double Q2[NSYS][NSYS];
static double *Q1[NSYS];

static double PHI2[NSYS][NSYS];
static double *PHI1[NSYS];

static double Qd2[NSYS][NSYS];
static double *Qd1[NSYS];

static double H2[MMES][NSYS];
static double *H1[MMES];

static double R2[MMES][MMES];
static double *R1[MMES];

static double S2[MMES][MMES];
static double *S1[MMES];

static double P2[NSYS][NSYS];
static double *P1[NSYS];
```

```
static double K2[NSYS][MMES];
static double *K1[NSYS];

#if NSYS>MMES
static double v1tmp1[NSYS];
static double v2tmp1[NSYS];
static double m1tmp2[NSYS][NSYS];
static double *m1tmp1[NSYS];
static double m2tmp2[NSYS][NSYS];
static double *m2tmp1[NSYS];
#else
static double v1tmp1[MMES];
static double v2tmp1[MMES];
static double m1tmp2[MMES][MMES];
static double *m1tmp1[MMES];
static double m2tmp2[MMES][MMES];
static double *m2tmp1[MMES];
#endif


// *****
// Provide access to the working data store.
// *****
void calmdl_wrk(int &nsys, int &mmes,
                Kflt_vector &x, Kflt_vector &z, Kflt_vector &r,
                Kflt_matrix &F, Kflt_matrix &Q,
        Kflt_matrix &PHI, Kflt_matrix &Qd,
        Kflt_matrix &H, Kflt_matrix &R, Kflt_matrix &S,
        Kflt_matrix &P, Kflt_matrix &K, Kflt_tmp &tmp)
{
  int i;

  nsys=NSYS;
  mmes=MMES;

  x=&x1[0];

  z=&z1[0];

  r=&r1[0];

  for (i=0; i<NSYS; i++)
    F1[i]=&F2[i][0];
  F=&F1[0];

  for (i=0; i<NSYS; i++)
    Q1[i]=&Q2[i][0];
  Q=&Q1[0];

  for (i=0; i<NSYS; i++)
    PHI1[i]=&PHI2[i][0];
  PHI=&PHI1[0];

  for (i=0; i<NSYS; i++)
    Qd1[i]=&Qd2[i][0];
  Qd=&Qd1[0];

  for (i=0; i<MMES; i++)
    H1[i]=&H2[i][0];
  H=&H1[0];

  for (i=0; i<MMES; i++)
    R1[i]=&R2[i][0];
  R=&R1[0];
```

```
  for (i=0; i<MMES; i++)
    S1[i]=&S2[i][0];
  S=&S1[0];

  for (i=0; i<NSYS; i++)
    P1[i]=&P2[i][0];
  P=&P1[0];

  for (i=0; i<NSYS; i++)
    K1[i]=&K2[i][0];
  K=&K1[0];

  tmp.vector1=&v1tmp1[0];
  tmp.vector2=&v2tmp1[0];
#if NSYS>MMES
  for (i=0; i<NSYS; i++)
    m1tmp1[i]=&m1tmp2[i][0];
  tmp.matrix1=&m1tmp1[0];
  for (i=0; i<NSYS; i++)
    m2tmp1[i]=&m2tmp2[i][0];
  tmp.matrix2=&m2tmp1[0];
#else
  for (i=0; i<MMES; i++)
    m1tmp1[i]=&m1tmp2[i][0];
  tmp.matrix1=&m1tmp1[0];
  for (i=0; i<MMES; i++)
    m2tmp1[i]=&m2tmp2[i][0];
  tmp.matrix2=&m2tmp1[0];
#endif

  return;
}


// *****
// Initialize the state estimate vector (x) and the error covariance
// matrix (P).
// *****
//
void calmdl_x0P0(void)
{
  Kflt_vector x;
  Kflt_matrix P;
  int i, j;

  x=&x1[0];
  P=&P1[0];

  for (i=0; i<NSYS; i++) {
    x[i]=0.0;
    for (j=0; j<NSYS; j++)
      P[i][j]=0.0;
  }

  // Initial INS inertial navigation error state covariance.
  P[0][0]=1.0e6;      // x position; 1000 m rms
  P[1][1]=1.0e6;      // y position; 1000 m rms
  P[2][2]=4.0;        // x velocity; 2 m/s rms
  P[3][3]=4.0;        // y velocity; 2 m/s rms
  P[4][4]=0.09;       // x misalignment; 0.3 mrad rms
  P[5][5]=0.09;       // y misalignment; 0.3 mrad rms
  P[6][6]=25.0;       // z misalignment; 5 mrad rms

  // Initial INS instrument error state covariance.
  P[7][7]=2.4e-7;     // x accelerometer bias; 50 micro_g (4.9e-4 m/s^2) rms
```

```
  P[8][8]=2.4e-7;     // y accelerometer bias; 50 micro_g (4.9e-4 m/s^2) rms
  P[9][9]=2.4e-9;     // x gyro bias; 0.01 deg/hr (4.8e-5 mrad/s) rms
  P[10][10]=2.4e-9;   // y gyro bias; 0.01 deg/hr (4.8e-5 mrad/s) rms
  P[11][11]=2.4e-9;   // z gyro bias; 0.01 deg/hr (4.8e-5 mrad/s) rms

  // Initial GPS position error state covariance.
  P[12][12]=400.0;    // x position; 20 m rms
  P[13][13]=400.0;    // y position; 20 m rms

  // Initial IMU inertial navigation error state covariance.
  P[14][14]=100.0;    // x position; 10 m rms
  P[15][15]=100.0;    // y position; 10 m rms
  P[16][16]=4.0e-4;   // x velocity; 0.02 m/s rms
  P[17][17]=4.0e-4;   // y velocity; 0.02 m/s rms
  P[18][18]=400.0;    // x misalignment; 20 mrad rms
  P[19][19]=400.0;    // y misalignment; 20 mrad rms
  P[20][20]=225.0;    // z misalignment; 15 mrad rms

  // Initial IMU instrument error state covariance.
  P[21][21]=2.1e-6;   // x gyro bias; 0.3 deg/hr (1.5e-3 mrad/s) rms
  P[22][22]=2.1e-6;   // y gyro bias; 0.3 deg/hr (1.5e-3 mrad/s) rms
  P[23][23]=2.1e-6;   // z gyro bias; 0.3 deg/hr (1.5e-3 mrad/s) rms

  // Initial antenna azimuth error state covariance.
  P[24][24]=400.0;   // antenna azimuth error; 20 mrad rms

  return;
}


// *****
// Construct the system dynamics matrix (F) and the process noise spectral
// density matrix (Q).
// *****
//
void calmdl_FQ(double insRxinv, double insRyinv,
       double insVw[3], double inswiew[3], double insweww[3],
       double insfw[3], double insCbw[3][3],
               double imuRxinv, double imuRyinv,
       double imuVw[3], double imuwiew[3], double imuweww[3],
       double imufw[3], double imuCbw[3][3],
       double gndspeed, double accel[3], double hdgrate)
{
  double Vgnd;
  double amag;
  double wzabs;
  double Rxinv;
  double Ryinv;
  double Vx;
  double Vy;
  double Vz;
  double wiex;
  double wiey;
  double wiez;
  double wewx;
  double wewy;
  double fx;
  double fy;
  double fz;

  Kflt_matrix F;
  Kflt_matrix Q;
  int I, j;

  F=&F1[0];
```

```
Q=&Q1[0];

for (i=0; i<NSYS; i++)
  for (j=0; j<NSYS; j++) {
    F[i][j]=0.0;
    Q[i][j]=0.0;
  }

// Compute some quatities used in constructing the spectral density matrix.
if (gndspeed>70.0)
  Vgnd=gndspeed;
else
  Vgnd=70.0;
amag=sqrt(pow(accel[0],2.0)+pow(accel[1],2.0)+pow(accel[2],2.0));
wzabs=fabs(hdgrate);

// INS inertial navigation error dynamics and random forcing function
// spectral density.
Rxinv=insRxinv;
Ryinv=insRyinv;
Vx=insVw[0];
Vy=insVw[1];
Vz=insVw[2];
wiex=inswiew[0];
wiey=inswiew[1];
wiez=inswiew[2];
wewx=insweww[0];
wewy=insweww[1];
fx=insfw[0];
fy=insfw[1];
fz=insfw[2];
F[0][0]=Vz*Rxinv;
F[0][2]=1.0;
F[1][1]=Vz*Ryinv;
F[1][3]=1.0;
F[2][0]=(2.0*wiex+wewx)*Vy*Rxinv;
F[2][1]=((2.0*wiey+wewy)*Vy+2.0*wiez*Vz)*Ryinv;
F[2][2]=-Vz*Rxinv;
F[2][3]=2.0*wiez;
F[2][5]=(-fz)/1000.0;
F[2][6]=(fy)/1000.0;
F[3][0]=-((2.0*wiex+wewx)*Vx+2.0*wiez*Vz)*Rxinv;
F[3][1]=-(2.0*wiey+wewy)*Vx*Ryinv;
F[3][2]=-2.0*wiez;
F[3][3]=-Vz*Ryinv;
F[3][4]=(fz)/1000.0;
F[3][6]=(-fx)/1000.0;
F[4][0]=(-wiez*Rxinv)*1000.0;
F[4][3]=(-Ryinv)*1000.0;
F[4][5]=wiez;
F[4][6]=-wiey-wewy;
F[5][1]=(-wiez*Ryinv)*1000.0;
F[5][2]=(Rxinv)*1000.0;
F[5][4]=-wiez;
F[5][6]=wiex+wewx;
F[6][0]=((wiex+wewx)*Rxinv)*1000.0;
F[6][1]=((wiey+wewy)*Ryinv)*1000.0;
F[6][4]=wiey+wewy;
F[6][5]=-wiex-wewx;

Q[2][2]=6.0e-3/Vgnd+3.2e-9*Vgnd*amag;
Q[3][3]=6.0e-3/Vgnd+3.2e-9*Vgnd*amag;
Q[4][4]=7.6e-7;
Q[5][5]=7.6e-7;
Q[6][6]=7.6e-7;
```

```
// INS instrument error dynamics and random forcing function spectral
// density.
F[7][7]=-1.0/360000.0;    // x accelerometer bias correlation time = 100 hr
F[8][8]=-1.0/360000.0;    // y accelerometer bias correlation time = 100 hr
F[9][9]=-1.0/360000.0;    // x gyro bias correlation time = 100 hr
F[10][10]=-1.0/360000.0;  // y gyro bias correlation time = 100 hr
F[11][11]=-1.0/360000.0;  // z gyro bias correlation time = 100 hr

Q[7][7]=2.0*2.4e-7/360000.0;    // x accel bias variance = 2.4e-7 (m/s^2)^2
Q[8][8]=2.0*2.4e-7/360000.0;    // y accel bias variance = 2.4e-7 (m/s^2)^2
Q[9][9]=2.0*2.4e-9/360000.0;    // x gyro bias variance = 2.4e-9 (mrad/s)^2
Q[10][10]=2.0*2.4e-9/360000.0;  // y gyro bias variance = 2.4e-9 (mrad/s)^2
Q[11][11]=2.0*2.4e-9/360000.0;  // z gyro bias variance = 2.4e-9 (mrad/s)^2

// INS inertial/instrument coupling.
F[2][7]=insCbw[0][0];
F[2][8]=insCbw[0][1];
F[3][7]=insCbw[1][0];
F[3][8]=insCbw[1][1];
F[4][9]=-insCbw[0][0];
F[4][10]=-insCbw[0][1];
F[4][11]=-insCbw[0][2];
F[5][9]=-insCbw[1][0];
F[5][10]=-insCbw[1][1];
F[5][11]=-insCbw[1][2];
F[6][9]=-insCbw[2][0];
F[6][10]=-insCbw[2][1];
F[6][11]=-insCbw[2][2];

// GPS position error dynamics and random forcing function spectral
// density.
F[12][12]=-1.0/10800.0;  // x position error correlation time = 3 hr
F[13][13]=-1.0/10800.0;  // y position error correlation time = 3 hr

Q[12][12]=2.0*400.0/10800.0;  // x position error variance = 400 m^2
Q[13][13]=2.0*400.0/10800.0;  // y position error variance = 400 m^2

// IMU inertial navigation error dynamics and random forcing function
// spectral density.
Rxinv=imuRxinv;
Ryinv=imuRyinv;
Vx=imuVw[0];
Vy=imuVw[1];
Vz=imuVw[2];
wiex=imuwiew[0];
wiey=imuwiew[1];
wiez=imuwiew[2];
wewx=imuweww[0];
wewy=imuweww[1];
fx=imufw[0];
fy=imufw[1];
fz=imufw[2];
F[14][14]=Vz*Rxinv;
F[14][16]=1.0;
F[15][15]=Vz*Ryinv;
F[15][17]=1.0;
F[16][14]=(2.0*wiex+wewx)*Vy*Rxinv;
F[16][15]=((2.0*wiey+wewy)*Vy+2.0*wiez*Vz)*Ryinv;
F[16][16]=-Vz*Rxinv;
F[16][17]=2.0*wiez;
F[16][19]=(-fz)/1000.0;
F[16][20]=(fy)/1000.0;
F[17][14]=-((2.0*wiex+wewx)*Vx+2.0*wiez*Vz)*Rxinv;
F[17][15]=-(2.0*wiey+wewy)*Vx*Ryinv;
F[17][16]=-2.0*wiez;
```

```
  F[17][17]=-Vz*Ryinv;
  F[17][18]=(fz)/1000.0;
  F[17][20]=(-fx)/1000.0;
  F[18][14]=(-wiez*Rxinv)*1000.0;
  F[18][17]=(-Ryinv)*1000.0;
  F[18][19]=wiez;
  F[18][20]=-wiey-wewy;
  F[19][15]=(-wiez*Ryinv)*1000.0;
  F[19][16]=(Rxinv)*1000.0;
  F[19][18]=-wiez;
  F[19][20]=wiex+wewx;
  F[20][14]=((wiex+wewx)*Rxinv)*1000.0;
  F[20][15]=((wiey+wewy)*Ryinv)*1000.0;
  F[20][18]=wiey+wewy;
  F[20][19]=-wiex-wewx;

  Q[16][16]=(5.1e-8+3.2e-9)*Vgnd*amag;
  Q[17][17]=(5.1e-8+3.2e-9)*Vgnd*amag;
  Q[18][18]=1.2e-8*Vgnd*amag+(1.2e-1+1.2e-2)*wzabs+7.6e-7;
  Q[19][19]=1.2e-8*Vgnd*amag+(1.2e-1+1.2e-2)*wzabs+7.6e-7;
  Q[20][20]=1.2e-8*Vgnd*amag+7.1e-2*wzabs+7.6e-7;

  // IMU instrument error dynamics and random forcing function spectral
  // density.
  F[21][21]=-1.0/36000.0;  // x gyro bias correlation time = 10 hr
  F[22][22]=-1.0/36000.0;  // y gyro bias correlation time = 10 hr
  F[23][23]=-1.0/36000.0;  // z gyro bias correlation time = 10 hr

  Q[21][21]=2.0*2.1e-6/36000.0;  // x gyro bias variance = 2.1e-6 (mrad/s)^2
  Q[22][22]=2.0*2.1e-6/36000.0;  // y gyro bias variance = 2.1e-6 (mrad/s)^2
  Q[23][23]=2.0*2.1e-6/36000.0;  // z gyro bias variance = 2.1e-6 (mrad/s)^2

  // IMU inertial/instrument coupling.
  F[18][21]=-imuCbw[0][0];
  F[18][22]=-imuCbw[0][1];
  F[18][23]=-imuCbw[0][2];
  F[19][21]=-imuCbw[1][0];
  F[19][22]=-imuCbw[1][1];
  F[19][23]=-imuCbw[1][2];
  F[20][21]=-imuCbw[2][0];
  F[20][22]=-imuCbw[2][1];
  F[20][23]=-imuCbw[2][2];

  return;
}


// *****
// Construct the measurement vector (z), the measurement matrix (H) and the
// measurement noise covariance matrix (R).
// *****
//
void calmdl_mes(double inslat, double inslong, double insalt,
        double insCwe[3][3], double insCbw[3][3],
        double imulat, double imulong, double imualt,
double imuhdg, double imuwan,
double imuCwe[3][3], double imuCbw[3][3],
        int gpsrdy, double gpslat, double gpslong, double gpsalt,
int taurdy, double tauazm,
const double RRP[3], const double RSD[3],
const double RGPS[3],
double tgtlat, double tgtlong, double tgtalt)
{
  double insRe[3];
  double imuRe[3];
```

```
double gpsRe[3];
double RRPe[3];
double RSDe[3];
double Cbe[3][3];
double rplat;
double rplong;
double rpalt;
double Dlat;
double Dlong;
double slat;
double clat;
double slat1;
double clat1;
double sDlat;
double sDlong;
double cDlong;
double swan;
double cwan;
double shdg;
double chdg;
double M;
double N;
double N1;
double RN;
double RN1;
double RM;
double RP;
double D;
double rg[2];
double rg2;
double brg;
double vectmp[3];

Kflt_vector z;
Kflt_matrix H;
Kflt_matrix R;
double sum;
int i, j, k;

z=&z1[0];
H=&H1[0];
R=&R1[0];

for (i=0; i<MMES; i++)
  z[i]=0.0;

for (i=0; i<MMES; i++)
  for (j=0; j<NSYS; j++)
    H[i][j]=0.0;

for (i=0; i<MMES; i++) {
  for (j=0; j<MMES; j++)
    R[i][j]=0.0;
  R[i][i]=1.0;
}

geod2Re(inslat, inslong, insalt, insRe);
geod2Re(imulat, imulong, imualt, imuRe);
geod2Re(gpslat, gpslong, gpsalt, gpsRe);

// INS/GPS position matching measurements.
if (gpsrdy) {
  vectmp[0]=insRe[0]-gpsRe[0];
  vectmp[1]=insRe[1]-gpsRe[1];
  vectmp[2]=insRe[2]-gpsRe[2];
```

```
  z[0]=insCwe[0][0]*vectmp[0]+insCwe[1][0]*vectmp[1]+insCwe[2][0]*vectmp[2]
      +insCbw[0][0]*RGPS[0]+insCbw[0][1]*RGPS[1]+insCbw[0][2]*RGPS[2];
  z[1]=insCwe[0][1]*vectmp[0]+insCwe[1][1]*vectmp[1]+insCwe[2][1]*vectmp[2]
      +insCbw[1][0]*RGPS[0]+insCbw[1][1]*RGPS[1]+insCbw[1][2]*RGPS[2];

  H[0][0]=1.0;
  H[0][12]=-1.0;
  H[1][1]=1.0;
  H[1][13]=-1.0;

  R[0][0]=25.0;  // x INS/GPS position matching measurement noise; 5 m rms
  R[1][1]=25.0;  // y INS/GPS position matching measurement noise; 5 m rms
}

// IMU/INS position matching measurements.
for (j=0; j<3; j++)
  for (i=0; i<3; i++) {
    sum=0.0;
    for (k=0; k<3; k++)
      sum=sum+insCwe[i][k]*insCbw[k][j];
    Cbe[i][j]=sum;
  }
RRPe[0]=Cbe[0][0]*RRP[0]+Cbe[0][1]*RRP[1]+Cbe[0][2]*RRP[2];
RRPe[1]=Cbe[1][0]*RRP[0]+Cbe[1][1]*RRP[1]+Cbe[1][2]*RRP[2];
RRPe[2]=Cbe[2][0]*RRP[0]+Cbe[2][1]*RRP[1]+Cbe[2][2]*RRP[2];
for (j=0; j<3; j++)
  for (i=0; i<3; i++) {
    sum=0.0;
    for (k=0; k<3; k++)
      sum=sum+imuCwe[i][k]*imuCbw[k][j];
    Cbe[i][j]=sum;
  }
RSDe[0]=Cbe[0][0]*RSD[0]+Cbe[0][1]*RSD[1]+Cbe[0][2]*RSD[2];
RSDe[1]=Cbe[1][0]*RSD[0]+Cbe[1][1]*RSD[1]+Cbe[1][2]*RSD[2];
RSDe[2]=Cbe[2][0]*RSD[0]+Cbe[2][1]*RSD[1]+Cbe[2][2]*RSD[2];
vectmp[0]=imuRe[0]-(insRe[0]+RRPe[0]+RSDe[0]);
vectmp[1]=imuRe[1]-(insRe[1]+RRPe[1]+RSDe[1]);
vectmp[2]=imuRe[2]-(insRe[2]+RRPe[2]+RSDe[2]);
z[2]=imuCwe[0][0]*vectmp[0]+imuCwe[1][0]*vectmp[1]+imuCwe[2][0]*vectmp[2];
z[3]=imuCwe[0][1]*vectmp[0]+imuCwe[1][1]*vectmp[1]+imuCwe[2][1]*vectmp[2];

H[2][14]=1.0;
H[3][15]=1.0;

R[2][2]=4.0;  // x IMU/INS position matching measurement noise; 2 m rms
R[3][3]=4.0;  // y IMU/INS position matching measurement noise; 2 m rms

// Antenna azimuth error measurement.
if (taurdy) {
  eradii(imulat, M, N);
  RM=M+imualt;
  RP=(N+imualt)*cos(imulat);
  shdg=sin(imuhdg);
  chdg=cos(imuhdg);
  rplat=imulat-(RSD[0]*chdg-RSD[1]*shdg)/RM;
  rplong=imulong-(RSD[0]*shdg+RSD[1]*chdg)/RP;
  rpalt=imualt+RSD[2];
  Dlat=tgtlat-rplat;
  Dlong=tgtlong-rplong;
  slat=sin(rplat);
  clat=cos(rplat);
  slat1=sin(tgtlat);
  clat1=cos(tgtlat);
  sDlat=sin(Dlat);
  sDlong=sin(Dlong);
```

```
      cDlong=cos(Dlong);
      swan=sin(imuwan);
      cwan=cos(imuwan);
      eradii(rplat, M, N);
      eradii(tgtlat, M, N1);
      RN=N+rpalt;
      RN1=N1+tgtalt;
      D=E_e2*(N1*slat1-N*slat);
      rg[0]=RN1*clat1*sDlong;
      rg[1]=RN1*(sDlat+clat1*slat*(1.0-cDlong))-D*clat;
      rg2=rg[0]*rg[0]+rg[1]*rg[1];
      brg=atan2(rg[0],rg[1]);

      z[4]=tauazm-(brg-imuhdg);
      if (z[4]>=M_PI)
        z[4]=z[4]-2.0*M_PI;
      else if (z[4]<-M_PI)
        z[4]=z[4]+2.0*M_PI;
      z[4]=z[4]*1000.0;

      H[4][0]=(-(rg[0]*cwan+rg[1]*swan)/rg2-(slat*swan)/(RN*clat))*1000.0;
      H[4][14]=H[4][0];
      H[4][1]=((rg[0]*swan-rg[1]*cwan)/rg2-(slat*cwan)/(RN*clat))*1000.0;
      H[4][15]=H[4][1];
      H[4][6]=1.0;
      H[4][20]=H[4][6];
      H[4][24]=1.0;

      R[4][4]=4.0;  // Antenna azimuth error measurement noise; 2 mrad rms
    }

  return;
}
```

```
// KFLT.H

#ifndef __KFLT_H
#define __KFLT_H

typedef double *Kflt_vector;
typedef double *const *Kflt_matrix;
typedef struct {
          double *vector1;
          double *vector2;
          double **matrix1;
          double **matrix2;
        } Kflt_tmp;


// *****
// Calculate the state transition matrix and the process noise covariance
// matrix.
// *****
//
void PHIQd(int nsys, int nTPHI, int nTQd, double Dt,
           Kflt_matrix F, Kflt_matrix Q, Kflt_matrix PHI, Kflt_matrix Qd,
   Kflt_tmp tmp);


// *****
// Perform the Kalman filter state estimate and error covariance
// extrapolation.
// *****
//
void extrap(int nsys, Kflt_matrix PHI, Kflt_matrix Qd,
            Kflt_vector x, Kflt_matrix P, Kflt_tmp tmp);


// *****
// Perform the Kalman filter state estimate and error covariance update.
// *****
//
void updat(int nsys, int mmes, Kflt_vector z, Kflt_matrix H, Kflt_matrix R,
           Kflt_vector r, Kflt_matrix S, Kflt_matrix K,
   Kflt_vector x, Kflt_matrix P, Kflt_tmp tmp);


#endif
```

```
// PHIQD.CPP
//
// Function to compute the state transistion matrix (PHI) and the process
// noise covariance matrix (Qd) by Taylor series expansion.  The system
// dynamics matrix (F) and the process noise spectral density matrix (Q) are
// assumed to be constant over the interval (Dt).  nTPHI and nTQd are the
// degrees of the truncated series expansions for PHI and Qd respectively.


#include "kflt.h"


// *****
// Calculate the state transition matrix and the process noise covariance
// matrix.
// *****
//
void PHIQd(int nsys, int nTPHI, int nTQd, double Dt,
           Kflt_matrix F, Kflt_matrix Q, Kflt_matrix PHI, Kflt_matrix Qd,
   Kflt_tmp tmp)
{
  double sum;
  int n;
  double Dtn;
  int i, j, k;

  // Calculate the state transition matrix.
  for (j=0; j<nsys; j++) {
    for (i=0; i<nsys; i++) {
      tmp.matrix1[i][j]=F[i][j]*Dt;
      PHI[i][j]=tmp.matrix1[i][j];
    }
    PHI[j][j]=1.0+PHI[j][j];
  }
  for (n=2; n<=nTPHI; n++) {
    for (j=0; j<nsys; j++)
      for (i=0; i<nsys; i++) {
        sum=0.0;
        for (k=0; k<nsys; k++)
          sum=sum+F[i][k]*tmp.matrix1[k][j];
        tmp.matrix2[i][j]=sum;
      }
    Dtn=Dt/n;
    for (j=0; j<nsys; j++)
      for (i=0; i<nsys; i++) {
        tmp.matrix1[i][j]=tmp.matrix2[i][j]*Dtn;
        PHI[i][j]=PHI[i][j]+tmp.matrix1[i][j];
      }
  }

  // Calculate the process noise covariance matrix.
  for (j=0; j<nsys; j++)
    for (i=0; i<nsys; i++) {
      tmp.matrix1[i][j]=Q[i][j]*Dt;
      Qd[i][j]=tmp.matrix1[i][j];
    }
  for (n=2; n<=nTQd; n++) {
    for (j=0; j<nsys; j++)
      for (i=0; i<nsys; i++) {
        sum=0.0;
        for (k=0; k<nsys; k++)
          sum=sum+F[i][k]*tmp.matrix1[k][j];
        tmp.matrix2[i][j]=sum;
      }
    Dtn=Dt/n;
```

```
    for (j=0; j<nsys; j++)
      for (i=0; i<=j; i++) {
        sum=(tmp.matrix2[i][j]+tmp.matrix2[j][i])*Dtn;
        tmp.matrix1[i][j]=sum;
        tmp.matrix1[j][i]=sum;
        sum=Qd[i][j]+sum;
        Qd[i][j]=sum;
        Qd[j][i]=sum;
      }
  }

  return;
}
```

```
// EXTRAP.CPP
//
// Function to extrapolate the Kalman filter state estimate vector (x) and
// the error covariance matrix (P) given the state transition matrix (PHI)
// and the process noise covariance matrix (Qd).  The Kalman filter
// extrapolation equations are directly implemented except that only the
// upper triangular portion of P is computed.  The symmetry property of P is
// used to determine the lower triangular portion.


#include "kflt.h"


// *****
// Perform the Kalman filter state estimate and error covariance
// extrapolation.
// *****
//
void extrap(int nsys, Kflt_matrix PHI, Kflt_matrix Qd,
            Kflt_vector x, Kflt_matrix P, Kflt_tmp tmp)
{
  double sum;
  int i, j, k;

  // Extrapolate the state estimate.
  for (i=0; i<nsys; i++) {
    sum=0.0;
    for (k=0; k<nsys; k++)
      sum=sum+PHI[i][k]*x[k];
  tmp.vector1[i]=sum;
  }
  for (i=0; i<nsys; i++)
    x[i]=tmp.vector1[i];

  // Extrapolate the error covariance.
  for (j=0; j<nsys; j++)
    for (i=0; i<nsys; i++) {
      sum=0.0;
      for (k=0; k<nsys; k++)
        sum=sum+PHI[i][k]*P[k][j];
      tmp.matrix1[i][j]=sum;
    }
  for (j=0; j<nsys; j++)
    for (i=0; i<=j; i++) {
      sum=0.0;
      for (k=0; k<nsys; k++)
        sum=sum+tmp.matrix1[i][k]*PHI[j][k];
      sum=sum+Qd[i][j];
      P[i][j]=sum;
      P[j][i]=sum;
    }

  return;
}
```

```
// UPDAT.CPP
//
// Function to update the Kalman filter state estimate vector (x) and
// the error covariance matrix (P) given the measurement vector (z), the
// measurement matrix (H) and the measurement noise covariance matrix (R).
// The Kalman filter update equations are directly implemented except that
// only the upper triangular portion of P is computed.  The symmetry property
// of P is used to determine the lower triangular portion.  The measurement
// residual vector (r), the residual covariance matrix (S) and the Kalman
// gain matrix (K) are also output.


#include <stdlib.h>
#include <iostream.h>
#include <math.h>

#include "kflt.h"

void inv(int n, Kflt_tmp tmp);


// *****
// Perform the Kalman filter state estimate and error covariance update.
// *****
//
void updat(int nsys, int mmes, Kflt_vector z, Kflt_matrix H, Kflt_matrix R,
           Kflt_vector r, Kflt_matrix S, Kflt_matrix K,
   Kflt_vector x, Kflt_matrix P, Kflt_tmp tmp)
{
  double sum;
  int i, j, k;

  // Compute the Kalman gain matrix.
  for (j=0; j<nsys; j++)
    for (i=0; i<mmes; i++) {
      sum=0.0;
      for (k=0; k<nsys; k++)
        sum=sum+H[i][k]*P[k][j];
      tmp.matrix2[i][j]=sum;
    }
  for (j=0; j<mmes; j++)
    for (i=0; i<mmes; i++) {
      sum=0.0;
      for (k=0; k<nsys; k++)
        sum=sum+tmp.matrix2[i][k]*H[j][k];
      S[i][j]=sum+R[i][j];
      tmp.matrix1[i][j]=S[i][j];
    }
  inv(mmes, tmp);
  for (j=0; j<mmes; j++)
    for (i=0; i<nsys; i++) {
      sum=0.0;
      for (k=0; k<mmes; k++)
        sum=sum+H[k][i]*tmp.matrix1[k][j];
      tmp.matrix2[i][j]=sum;
    }
  for (j=0; j<mmes; j++)
    for (i=0; i<nsys; i++) {
      sum=0.0;
      for (k=0; k<nsys; k++)
        sum=sum+P[i][k]*tmp.matrix2[k][j];
      K[i][j]=sum;
    }

  // Update the state estimate.
```

```
  for (i=0; i<mmes; i++) {
    sum=0.0;
    for (k=0; k<nsys; k++)
      sum=sum+H[i][k]*x[k];
    r[i]=z[i]-sum;
  }
  for (i=0; i<nsys; i++) {
    sum=0.0;
    for (k=0; k<mmes; k++)
      sum=sum+K[i][k]*r[k];
    x[i]=x[i]+sum;
  }

  // Update the error covariance.
  for (j=0; j<nsys; j++) {
    for (i=0; i<nsys; i++) {
      sum=0.0;
      for (k=0; k<mmes; k++)
        sum=sum-K[i][k]*H[k][j];
      tmp.matrix1[i][j]=sum;
      tmp.matrix2[i][j]=P[i][j];
    }
    tmp.matrix1[j][j]=1.0+tmp.matrix1[j][j];
  }
  for (j=0; j<nsys; j++)
    for (i=0; i<=j; i++) {
      sum=0.0;
      for (k=0; k<nsys; k++)
        sum=sum+tmp.matrix1[i][k]*tmp.matrix2[k][j];
      P[i][j]=sum;
      P[j][i]=sum;
    }

  return;
}


// Local function to perform a matrix inversion using LU decomposition.
//   On entry, an mxm matrix is supplied in tmp.matrix1.
//   On exit, the matrix inverse is provided in tmp.matrix1.
//   The original matrix is destroyed as are the contents of tmp.vector1,
//   tmp.vector2 and tmp.matrix2.
//
void inv(int m, Kflt_tmp tmp)
{
  Kflt_matrix A;
  Kflt_matrix LU;
  double *s;
  int *ix;
  double amax;
  double sum;
  double atmp;
  int imax;
  int ii;
  int i, j, k;

  A=tmp.matrix1;
  LU=tmp.matrix2;
  s=tmp.vector1;
  ix=(int *) tmp.vector2;

  // Perform the LU decomposition.
  for (i=0; i<m; i++) {
    amax=0.0;
    for (j=0; j<m; j++) {
```

```
      if (fabs(A[i][j])>amax)
        amax=fabs(A[i][j]);
      LU[i][j]=A[i][j];
    }
    if (amax==0.0) {
      cerr << "Singular matrix in function inv\n";
      exit(EXIT_FAILURE);
    }
    s[i]=1.0/amax;
  }
  for (j=0; j<m; j++) {
    for (i=0; i<j; i++) {
      sum=LU[i][j];
      for (k=0; k<i; k++)
        sum=sum-LU[i][k]*LU[k][j];
      LU[i][j]=sum;
    }
    amax=0.0;
    for (i=j; i<m; i++) {
      sum=LU[i][j];
      for (k=0; k<j; k++)
        sum=sum-LU[i][k]*LU[k][j];
      LU[i][j]=sum;
      atmp=s[i]*fabs(sum);
      if (atmp>=amax) {
        imax=i;
        amax=atmp;
      }
    }
    if (j!=imax) {
      for (k=0; k<m; k++) {
        atmp=LU[imax][k];
        LU[imax][k]=LU[j][k];
        LU[j][k]=atmp;
      }
      s[imax]=s[j];
    }
    ix[j]=imax;
    if (LU[j][j]==0.0) {
      cerr << "Singular matrix in function inv\n";
      exit(EXIT_FAILURE);
    }
    if (j!=m-1) {
      atmp=1.0/LU[j][j];
      for (i=j+1; i<m; i++)
        LU[i][j]=LU[i][j]*atmp;
    }
  }

  // Perform the matrix inversion column by column.
  for (j=0; j<m; j++) {
    for (i=0; i<m; i++)
      A[i][j]=0.0;
    A[j][j]=1.0;
  }
  for (j=0; j<m; j++) {
    ii=-1;
    for (i=0; i<m; i++) {
      imax=ix[i];
      sum=A[imax][j];
      A[imax][j]=A[i][j];
      if (ii!=-1)
        for (k=ii; k<i; k++)
          sum=sum-LU[i][k]*A[k][j];
      else if (sum!=0.0)
```

```
        ii=i;
      A[i][j]=sum;
    }
    for (i=m-1; i>=0; i--) {
      sum=A[i][j];
      if (i!=m-1)
        for (k=i+1; k<m; k++)
          sum=sum-LU[i][k]*A[k][j];
      A[i][j]=sum/LU[i][i];
    }
  }

  return;
}
```

```
// NAVGTR.H

struct navgtr_record {
  double time;
  double latitude;
  double longitude;
  double altitude;
  double gndspeed;
  double track;
  double heading;
  double azimuth;
  double pitch;
  double roll;
  double vertspeed;
};

struct navwork_record {
  double vertaccelbias;
  double vertaccelfb;
  double vertvelfb;
  double Rxinv;
  double Ryinv;
  double Tinv;
  double gravity;
  double wander;
  double Dthiwb[3];
  double Dthwbb[3];
  double Dtheww[3];
  double Vw[3];
  double wiew[3];
  double weww[3];
  double fw[3];
  double qbw[4];
  double qwe[4];
  double Cbw[3][3];
  double Cwe[3][3];
};


// *****
// Perform inertial navigation calculations.
// *****
//
void navgtr(long cycle_count, double cur_time, double Dt,
            double DVelb[3], double Dthibb[3], double altref,
            navgtr_record &navrec, navwork_record &wrkrec);
```

```
// NAVGTR.CPP
// Perform inertial navigation calculations.

#include <math.h>

#include "navlib.h"
#include "navgtr.h"

static void vertfb(double Dt, double altitude, double altref,
                   double &vertaccelbias,
   double &vertaccelfb, double &vertvelfb);


// *****
// Perform inertial navigation calculations.
// *****
//
void navgtr(long cycle_count, double cur_time, double Dt,
            double Dvelb[3], double Dthibb[3], double altref,
            navgtr_record &n, navwork_record &w)
{
  double roll;
  double pitch;
  double azimuth;
  double heading;
  double latitude;
  double longitude;
  double wander;
  double altitude;
  double gndspeed;
  double vertspeed;
  double track;
  double Dvelw[3];
  double DVw[3];
  double Dthiwb[3];
  double Dthwbb[3];
  double Dtheww[3];
  double Vw[3];
  double wiew[3];
  double weww[3];
  double qbw[4];
  double qwe[4];
  double Cbw[3][3];
  double Cwe[3][3];

  double vectmp[3];
  int i, j;

  if (cycle_count==0L) {
    Ea2Cbw(n.roll, n.pitch, n.azimuth, w.Cbw);
    dcm2q(w.Cbw, w.qbw);

    Dvelw[0]=w.Cbw[0][0]*Dvelb[0]+w.Cbw[0][1]*Dvelb[1]+w.Cbw[0][2]*Dvelb[2];
    Dvelw[1]=w.Cbw[1][0]*Dvelb[0]+w.Cbw[1][1]*Dvelb[1]+w.Cbw[1][2]*Dvelb[2];
    Dvelw[2]=w.Cbw[2][0]*Dvelb[0]+w.Cbw[2][1]*Dvelb[1]+w.Cbw[2][2]*Dvelb[2];
    w.fw[0]=Dvelw[0]/Dt;
    w.fw[1]=Dvelw[1]/Dt;
    w.fw[2]=Dvelw[2]/Dt;

    w.wander=n.azimuth-n.heading;
    Ea2Cwe(n.latitude, n.longitude, w.wander, w.Cwe);
    dcm2q(w.Cwe, w.qwe);

    w.Vw[0]=n.gndspeed*cos(n.track+w.wander);
    w.Vw[1]=-n.gndspeed*sin(n.track+w.wander);
```

```
    w.Vw[2]=n.vertspeed;

    invradii1(w.Cwe, n.altitude, w.Rxinv, w.Ryinv, w.Tinv);
    w.weww[0]=-w.Vw[1]*w.Ryinv+w.Vw[0]*w.Tinv;
    w.weww[1]=w.Vw[0]*w.Rxinv-w.Vw[1]*w.Tinv;
    w.weww[2]=0.0;

    w.Dtheww[0]=w.weww[0]*Dt;
    w.Dtheww[1]=w.weww[1]*Dt;
    w.Dtheww[2]=w.weww[2]*Dt;

    w.wiew[0]=w.Cwe[0][0]*E_wie;
    w.wiew[1]=w.Cwe[0][1]*E_wie;
    w.wiew[2]=w.Cwe[0][2]*E_wie;

    vectmp[0]=(w.wiew[0]+w.weww[0])*Dt;
    vectmp[1]=(w.wiew[1]+w.weww[1])*Dt;
    vectmp[2]=(w.wiew[2]+w.weww[2])*Dt;
    w.Dthiwb[0]=w.Cbw[0][0]*vectmp[0]+w.Cbw[1][0]*vectmp[1]
                +w.Cbw[2][0]*vectmp[2];
    w.Dthiwb[1]=w.Cbw[0][1]*vectmp[0]+w.Cbw[1][1]*vectmp[1]
                +w.Cbw[2][1]*vectmp[2];
    w.Dthiwb[2]=w.Cbw[0][2]*vectmp[0]+w.Cbw[1][2]*vectmp[1]
                +w.Cbw[2][2]*vectmp[2];

    w.Dthwbb[0]=Dthibb[0]-w.Dthiwb[0];
    w.Dthwbb[1]=Dthibb[1]-w.Dthiwb[1];
    w.Dthwbb[2]=Dthibb[2]-w.Dthiwb[2];

    w.gravity=gravity1(w.Cwe, n.altitude);

    w.vertaccelbias=0.0;
    w.vertaccelfb=0.0;
    w.vertvelfb=0.0;
  }
  else {
    Dthwbb[0]=Dthibb[0]-w.Dthiwb[0];
    Dthwbb[1]=Dthibb[1]-w.Dthiwb[1];
    Dthwbb[2]=Dthibb[2]-w.Dthiwb[2];

    qbw[0]=w.qbw[0];
    qbw[1]=w.qbw[1];
    qbw[2]=w.qbw[2];
    qbw[3]=w.qbw[3];
    qupdt(Dthwbb, w.Dthwbb, qbw);
    q2dcm(qbw, Cbw);
    Cbw2Ea(Cbw, roll, pitch, azimuth);

    Dvelw[0]=Cbw[0][0]*Dvelb[0]+Cbw[0][1]*Dvelb[1]+Cbw[0][2]*Dvelb[2];
    Dvelw[1]=Cbw[1][0]*Dvelb[0]+Cbw[1][1]*Dvelb[1]+Cbw[1][2]*Dvelb[2];
    Dvelw[2]=Cbw[2][0]*Dvelb[0]+Cbw[2][1]*Dvelb[1]+Cbw[2][2]*Dvelb[2];

    vectmp[0]=2.0*w.wiew[0]+w.weww[0];
    vectmp[1]=2.0*w.wiew[1]+w.weww[1];
    vectmp[2]=2.0*w.wiew[2]+w.weww[2];
    DVw[0]=Dvelw[0]-(vectmp[1]*w.Vw[2]-vectmp[2]*w.Vw[1])*Dt;
    DVw[1]=Dvelw[1]-(vectmp[2]*w.Vw[0]-vectmp[0]*w.Vw[2])*Dt;
    DVw[2]=Dvelw[2]-(vectmp[0]*w.Vw[1]-vectmp[1]*w.Vw[0])*Dt
          -(w.gravity+w.vertaccelbias+w.vertaccelfb)*Dt;

    Vw[0]=w.Vw[0]+DVw[0];
    Vw[1]=w.Vw[1]+DVw[1];
    Vw[2]=w.Vw[2]+DVw[2];

    weww[0]=-Vw[1]*w.Ryinv+Vw[0]*w.Tinv;
```

```
weww[1]=Vw[0]*w.Rxinv-Vw[1]*w.Tinv;
weww[2]=0.0;

Dtheww[0]=0.5*(weww[0]+w.weww[0])*Dt;
Dtheww[1]=0.5*(weww[1]+w.weww[1])*Dt;
Dtheww[2]=0.5*(weww[2]+w.weww[2])*Dt;

qwe[0]=w.qwe[0];
qwe[1]=w.qwe[1];
qwe[2]=w.qwe[2];
qwe[3]=w.qwe[3];
qupdt(Dtheww, w.Dtheww, qwe);
q2dcm(qwe, Cwe);
Cwe2Ea(Cwe, latitude, longitude, wander);

altitude=n.altitude+(0.5*(Vw[2]+w.Vw[2])-w.vertvelfb)*Dt;

wiew[0]=Cwe[0][0]*E_wie;
wiew[1]=Cwe[0][1]*E_wie;
wiew[2]=Cwe[0][2]*E_wie;

vectmp[0]=(wiew[0]+weww[0])*Dt;
vectmp[1]=(wiew[1]+weww[1])*Dt;
vectmp[2]=(wiew[2]+weww[2])*Dt;
Dthiwb[0]=Cbw[0][0]*vectmp[0]+Cbw[1][0]*vectmp[1]+Cbw[2][0]*vectmp[2];
Dthiwb[1]=Cbw[0][1]*vectmp[0]+Cbw[1][1]*vectmp[1]+Cbw[2][1]*vectmp[2];
Dthiwb[2]=Cbw[0][2]*vectmp[0]+Cbw[1][2]*vectmp[1]+Cbw[2][2]*vectmp[2];

gndspeed=sqrt(Vw[0]*Vw[0]+Vw[1]*Vw[1]);
vertspeed=Vw[2];
track=atan2(-Vw[1],Vw[0])-wander;
if (track>=M_PI)
  track=track-2.0*M_PI;
else if (track<-M_PI)
  track=track+2.0*M_PI;
heading=azimuth-wander;
if (heading>=M_PI)
  heading=heading-2.0*M_PI;
else if (heading<-M_PI)
  heading=heading+2.0*M_PI;

n.time=cur_time;
n.latitude=latitude;
n.longitude=longitude;
n.altitude=altitude;
n.gndspeed=gndspeed;
n.track=track;
n.heading=heading;
n.azimuth=azimuth;
n.pitch=pitch;
n.roll=roll;
n.vertspeed=vertspeed;

vertfb(Dt, altitude, altref,
       w.vertaccelbias, w.vertaccelfb, w.vertvelfb);
invradii1(Cwe, altitude, w.Rxinv, w.Ryinv, w.Tinv);
w.gravity=gravity1(Cwe, altitude);
w.wander=wander;
for (i=0; i<=2; i++) {
  w.Dthiwb[i]=Dthiwb[i];
  w.Dthwbb[i]=Dthwbb[i];
  w.Dtheww[i]=Dtheww[i];
  w.Vw[i]=Vw[i];
  w.wiew[i]=wiew[i];
  w.weww[i]=weww[i];
```

```
      w.fw[i]=Dvelw[i]/Dt;
    }
    for (i=0; i<=3; i++) {
      w.qbw[i]=qbw[i];
      w.qwe[i]=qwe[i];
    }
    for (i=0; i<=2; i++)
      for (j=0; j<=2; j++) {
        w.Cbw[i][j]=Cbw[i][j];
        w.Cwe[i][j]=Cwe[i][j];
      }
  }
  return;
}


// Local function to compute vertical channel error control feedback.
//
static void vertfb(double Dt, double altitude, double altref,
                   double &vertaccelbias,
   double &vertaccelfb, double &vertvelfb)
{
  static const double k1=3.0e-2;
  static const double k2=3.0e-4;
  static const double k3=1.0e-6;

  double altdiff;

  altdiff=altitude-altref;
  vertvelfb=k1*altdiff;
  vertaccelfb=k2*altdiff;
  vertaccelbias=vertaccelbias+k3*altdiff*Dt;

  return;
}
```

```
// NAVLIB.H
// Definitions for the navigation library.
//
// The functions defined in this library use the following coordinate
// systems:
//
// Earth-fixed coordinates:
//   x-axis through North pole
//   y-axis for right-hand system
//   z-axis through Greenwich meridian
//
// Geographic coordinates:
//   x-axis north
//   y-axis east
//   z-axis down
//
// Wander azimuth coordinates:
//   x-axis displaced counterclockwise from north through wander angle
//   y-axis for right-hand system
//   z-axis up
//
// Body coordinates:
//   x-axis forward
//   y-axis right
//   z-axis down


#ifndef __NAVLIB_H
#define __NAVLIB_H


// *****
// Compute the radius of curvature in the plane of the meridian (M) and the
// radius of curvature in the plane of the prime vertical (N) given the
// latitude.
// *****
//
void eradii(double latitude, double &M, double &N);


// *****
// Compute the radius of curvature in the plane of the meridian (M) and the
// radius of curvature in the plane of the prime vertical (N) given the
// wander azimuth to Earth DCM.
// *****
//
void eradii1(const double Cwe[3][3], double &M, double &N);


// *****
// Compute the inverse radii of curvature given the latitude, altitude and
// wander angle.
// *****
//
void invradii(double latitude, double altitude, double wander,
              double &Rxinv, double &Ryinv, double &Tinv);


// *****
// Compute the inverse radii of curvature given the wander azimuth to Earth
// DCM and the altitude.
// *****
//
void invradii1(const double Cwe[3][3], double altitude,
              double &Rxinv, double &Ryinv, double &Tinv);
```

```
// *****
// Compute the normal gravity given the latitude and altitude.
// *****
//
double gravity(double latitude, double altitude);


// *****
// Compute the normal gravity given the wander azimuth to Earth DCM and the
// altitude.
// *****
//
double gravity1(const double Cwe[3][3], double altitude);


// *****
// Compute latitude, longitude and altitude from Earth-fixed coordinates.
// *****
//
void Re2geod(const double Re[3],
             double &latitude, double &longitude, double &altitude);


// *****
// Compute Earth-fixed coordinates from latitude, longitude and altitude.
// *****
//
void geod2Re(double latitude, double longitude, double altitude,
             double Re[3]);


// *****
// Compute the body to wander azimuth DCM from roll, pitch and platform
// azimuth (Euler angles).
// *****
//
void Ea2Cbw(double roll, double pitch, double azimuth,
            double Cbw[3][3]);


// *****
// Compute the wander azimuth to Earth DCM from latitude, longitude and
// wander angle (Euler angles).
// *****
//
void Ea2Cwe(double latitude, double longitude, double wander,
            double Cwe[3][3]);


// *****
// Compute the Earth to geographic (NED) DCM from latitude and longitude.
// *****
//
void Ea2Ceg(double latitude, double longitude, double Ceg[3][3]);


// *****
// Compute the geographic (NED) to body DCM from roll, pitch and heading
// (Euler angles).
// *****
//
void Ea2Cgb(double roll, double pitch, double heading, double Cgb[3][3]);
```

```
// *****
// Compute roll, pitch and platform azimuth (Euler angles) from the
// body to wander azimuth DCM.
// *****
//
void Cbw2Ea(const double Cbw[3][3],
            double &roll, double &pitch, double &azimuth);


// *****
// Compute latitude, longitude and wander angle (Euler angles) from the
// wander azimuth to Earth DCM.
// *****
//
void Cwe2Ea(const double Cwe[3][3],
            double &latitude, double &longitude, double &wander);


// *****
// Obtain a quaternion from a direction cosine matrix.
// *****
//
void dcm2q(const double C[3][3], double q[4]);


// *****
// Update a quaternion using third order quaternion integration.
// *****
//
void qupdt(const double Dangl[3], const double Dangl_L[3], double q[4]);


// *****
// Obtain a direction cosine matrix from a quaternion.
// *****
//
void q2dcm(const double q[4], double C[3][3]);


// *****
// Earth constants (WGS 84).
//   a    semi-major axis [m]
//   f    flattening
//   e2   first eccentricity squared
//   wie  angular velocity [rad/s]
//   GM   gravitational constant [m^3/s^2]
//   ge   normal gravity at equator [m/s^2]
//   k    constant used in gravity formula
// *****
//
#define E_a   6.378137e6
#define E_f   3.35281066474e-3
#define E_e2  6.69437999013e-3
#define E_wie 7.292115e-5
#define E_GM  3.986005e14
#define E_ge  9.7803267714
#define E_k   1.93185138639e-3


#endif
```

```
// ERADII.CPP
// Functions to compute the radius of curvature in the plane of the meridian
// and the radius of curvature in the plane of the prime vertical.

#include <math.h>

#include "navlib.h"


// *****
// Compute the radius of curvature in the plane of the meridian (M) and the
// radius of curvature in the plane of the prime vertical (N) given the
// latitude.
// *****
//
void eradii(double latitude, double &M, double &N)
{
  double slat2;

  slat2=pow(sin(latitude), 2.0);
  M=E_a*(1.0-E_e2)/pow(1.0-E_e2*slat2, 1.5);
  N=E_a/sqrt(1.0-E_e2*slat2);
  return;
}


// *****
// Compute the radius of curvature in the plane of the meridian (M) and the
// radius of curvature in the plane of the prime vertical (N) given the
// wander azimuth to Earth DCM.
// *****
//
void eradii1(const double Cwe[3][3], double &M, double &N)
{
  double slat2;

  slat2=Cwe[0][2]*Cwe[0][2];
  M=E_a*(1.0-E_e2)/pow(1.0-E_e2*slat2, 1.5);
  N=E_a/sqrt(1.0-E_e2*slat2);
  return;
}
```

```
// INVRADII.CPP
// Functions to compute the inverse radii of curvature.

#include <math.h>

#include "navlib.h"


// *****
// Compute the inverse radii of curvature given the latitude, altitude and
// wander angle.
// *****
//
void invradii(double latitude, double altitude, double wander,
              double &Rxinv, double &Ryinv, double &Tinv)
{
  double swan;
  double cwan;
  double swan2;
  double cwan2;
  double M;
  double N;
  double R1;
  double R2;
  double R1R2;

  swan=sin(wander);
  cwan=cos(wander);
  swan2=swan*swan;
  cwan2=cwan*cwan;
  eradii(latitude, M, N);
  R1=M+altitude;
  R2=N+altitude;
  R1R2=R1*R2;
  Rxinv=(R1*swan2+R2*cwan2)/R1R2;
  Ryinv=(R1*cwan2+R2*swan2)/R1R2;
  Tinv=swan*cwan*(R2-R1)/R1R2;
  return;
}


// *****
// Compute the inverse radii of curvature given the wander azimuth to Earth
// DCM and the altitude.
// *****
//
void invradii1(const double Cwe[3][3], double altitude,
              double &Rxinv, double &Ryinv, double &Tinv)
{
  double u;
  double v;
  double u2;
  double v2;
  double w2;
  double swan2;
  double cwan2;
  double swancwan;
  double M;
  double N;
  double R1;
  double R2;
  double R1R2;

  u=Cwe[0][0];
  v=-Cwe[0][1];
```

```
  u2=u*u;
  v2=v*v;
  w2=u2+v2;
  swan2=v2/w2;
  cwan2=u2/w2;
  swancwan=u*v/w2;
  eradii1(Cwe, M, N);
  R1=M+altitude;
  R2=N+altitude;
  R1R2=R1*R2;
  Rxinv=(R1*swan2+R2*cwan2)/R1R2;
  Ryinv=(R1*cwan2+R2*swan2)/R1R2;
  Tinv=swancwan*(R2-R1)/R1R2;
  return;
}
```

```
// GRAVITY.CPP
// Functions to compute normal gravity.

#include <math.h>

#include "navlib.h"

static const double m=E_wie*E_wie*E_a*E_a*E_a*(1.0-E_f)/E_GM;
static const double c1=(2.0/E_a)*(1.0+E_f+m);
static const double c2=4.0*E_f/E_a;
static const double c3=3.0/(E_a*E_a);


// *****
// Compute the normal gravity given the latitude and altitude.
// *****
//
double gravity(double latitude, double altitude)
{
  double slat2;
  double g;

  slat2=pow(sin(latitude), 2.0);
  g=E_ge*(1.0+E_k*slat2)/sqrt(1.0-E_e2*slat2);
  g=g*(1.0-(c1-c2*slat2)*altitude+c3*altitude*altitude);
  return g;
}


// *****
// Compute the normal gravity given the wander azimuth to Earth DCM and the
// altitude.
// *****
//
double gravity1(const double Cwe[3][3], double altitude)
{
  double slat2;
  double g;

  slat2=Cwe[0][2]*Cwe[0][2];
  g=E_ge*(1.0+E_k*slat2)/sqrt(1.0-E_e2*slat2);
  g=g*(1.0-(c1-c2*slat2)*altitude+c3*altitude*altitude);
  return g;
}
```

```
// RE2GEOD.CPP
// Function to compute geodetic coordinates (latitude, longitude and
// altitude) given the Earth-fixed coordinates.
//
// Earth-fixed coordinates:
//
//   x-axis through North pole
//   y-axis for right-hand system
//   z-axis through Greenwich meridian

#include <math.h>

#include "navlib.h"

static const double c=1.0-E_e2;


// *****
// Compute latitude, longitude and altitude from Earth-fixed coordinates.
// *****
//
void Re2geod(const double Re[3],
             double &latitude, double &longitude, double &altitude)
{
  double x, y, z;
  double w;
  double N;
  double slat;

  x=Re[0];
  y=Re[1];
  z=Re[2];
  longitude=atan2(-y,z);
  w=sqrt(y*y+z*z);
  latitude=atan2(x,c*w);
  if (fabs(latitude)<M_PI_4)
    for (int i=1; i<=6; i++) {
      slat=sin(latitude);
      N=E_a/sqrt(1.0-E_e2*slat*slat);
      altitude=w/cos(latitude)-N;
      latitude=atan2(x*(N+altitude),w*(c*N+altitude));
    }
  else
    for (int i=1; i<=6; i++) {
      slat=sin(latitude);
      N=E_a/sqrt(1.0-E_e2*slat*slat);
      altitude=x/slat-c*N;
      latitude=atan2(x*(N+altitude),w*(c*N+altitude));
    }
  return;
}
```

```
// GEOD2RE.CPP
// Function to compute Earth-fixed coordinates given the geodetic coordinates
// (latitude, longitude and altitude).
//
// Earth-fixed coordinates:
//
//   x-axis through North pole
//   y-axis for right-hand system
//   z-axis through Greenwich meridian

#include <math.h>

#include "navlib.h"


// *****
// Compute Earth-fixed coordinates from latitude, longitude and altitude.
// *****
//
void geod2Re(double latitude, double longitude, double altitude,
             double Re[3])
{
  double slat;
  double clat;
  double N;

  slat=sin(latitude);
  clat=cos(latitude);
  N=E_a/sqrt(1.0-E_e2*slat*slat);
  Re[0]=(N*(1.0-E_e2)+altitude)*slat;
  Re[1]=-(N+altitude)*clat*sin(longitude);
  Re[2]=(N+altitude)*clat*cos(longitude);
  return;
}
```

```
// EA2CBW.CPP
// Function to compute the DCM which transforms a vector in body coordinates
// to wander azimuth coordinates.
//
// Body coordinates:
//
//   x-axis forward
//   y-axis right
//   z-axis down
//
// Wander azimuth coordinates:
//
//   x-axis displaced counterclockwise from north through wander angle
//   y-axis for right-hand system
//   z-axis up

#include <math.h>


// *****
// Compute the body to wander azimuth DCM from roll, pitch and platform
// azimuth (Euler angles).
// *****
//
void Ea2Cbw(double roll, double pitch, double azimuth,
            double Cbw[3][3])
{
  double sroll;
  double croll;
  double sptch;
  double cptch;
  double saz;
  double caz;

  sroll=sin(roll);
  croll=cos(roll);
  sptch=sin(pitch);
  cptch=cos(pitch);
  saz=sin(azimuth);
  caz=cos(azimuth);
  Cbw[0][0]=cptch*caz;
  Cbw[0][1]=-croll*saz+sroll*sptch*caz;
  Cbw[0][2]=sroll*saz+croll*sptch*caz;
  Cbw[1][0]=-cptch*saz;
  Cbw[1][1]=-croll*caz-sroll*sptch*saz;
  Cbw[1][2]=sroll*caz-croll*sptch*saz;
  Cbw[2][0]=sptch;
  Cbw[2][1]=-sroll*cptch;
  Cbw[2][2]=-croll*cptch;
  return;
}
```

```
// EA2CWE.CPP
// Function to compute the DCM which transforms a vector in wander azimuth
// coordinates to Earth-fixed coordinates.
//
// Wander azimuth coordinates:
//
//   x-axis displaced counterclockwise from north through wander angle
//   y-axis for right-hand system
//   z-axis up
//
// Earth-fixed coordinates:
//
//   x-axis through North pole
//   y-axis for right-hand system
//   z-axis through Greenwich meridian

#include <math.h>


// *****
// Compute the wander azimuth to Earth DCM from latitude, longitude and
// wander angle (Euler angles).
// *****
//
void Ea2Cwe(double latitude, double longitude, double wander,
            double Cwe[3][3])
{
  double slat;
  double clat;
  double slong;
  double clong;
  double swan;
  double cwan;

  slat=sin(latitude);
  clat=cos(latitude);
  slong=sin(longitude);
  clong=cos(longitude);
  swan=sin(wander);
  cwan=cos(wander);
  Cwe[0][0]=clat*cwan;
  Cwe[0][1]=-clat*swan;
  Cwe[0][2]=slat;
  Cwe[1][0]=clong*swan+slat*slong*cwan;
  Cwe[1][1]=clong*cwan-slat*slong*swan;
  Cwe[1][2]=-clat*slong;
  Cwe[2][0]=slong*swan-slat*clong*cwan;
  Cwe[2][1]=slong*cwan+slat*clong*swan;
  Cwe[2][2]=clat*clong;
  return;
}
```

```
// EA2CEG.CPP
// Function to compute the DCM which transforms a vector in Earth-fixed
// coordinates to geographic coordinates.
//
// Earth-fixed coordinates:
//
//   x-axis through North pole
//   y-axis for right-hand system
//   z-axis through Greenwich meridian
//
// Geographic coordinates:
//
//   x-axis north
//   y-axis east
//   z-axis down

#include <math.h>


// *****
// Compute the Earth to geographic (NED) DCM from latitude and longitude.
// *****
//
void Ea2Ceg(double latitude, double longitude, double Ceg[3][3])
{
  double slat;
  double clat;
  double slong;
  double clong;

  slat=sin(latitude);
  clat=cos(latitude);
  slong=sin(longitude);
  clong=cos(longitude);
  Ceg[0][0]=clat;
  Ceg[0][1]=slat*slong;
  Ceg[0][2]=-slat*clong;
  Ceg[1][0]=0.0;
  Ceg[1][1]=-clong;
  Ceg[1][2]=-slong;
  Ceg[2][0]=-slat;
  Ceg[2][1]=clat*slong;
  Ceg[2][2]=-clat*clong;
  return;
}
```

```
// EA2CGB.CPP
// Function to compute the DCM which transforms a vector in geographic (NED)
// coordinates to body coordinates.
//
// Geographic coordinates:
//
//   x-axis north
//   y-axis east
//   z-axis down
//
// Body coordinates:
//
//   x-axis forward
//   y-axis right
//   z-axis down

#include <math.h>


// *****
// Compute the geographic (NED) to body DCM from roll, pitch and heading.
// *****
//
void Ea2Cgb(double roll, double pitch, double heading, double Cgb[3][3])
{
  double sroll;
  double croll;
  double sptch;
  double cptch;
  double shdg;
  double chdg;

  sroll=sin(roll);
  croll=cos(roll);
  sptch=sin(pitch);
  cptch=cos(pitch);
  shdg=sin(heading);
  chdg=cos(heading);
  Cgb[0][0]=cptch*chdg;
  Cgb[0][1]=cptch*shdg;
  Cgb[0][2]=-sptch;
  Cgb[1][0]=sroll*sptch*chdg-croll*shdg;
  Cgb[1][1]=sroll*sptch*shdg+croll*chdg;
  Cgb[1][2]=sroll*cptch;
  Cgb[2][0]=croll*sptch*chdg+sroll*shdg;
  Cgb[2][1]=croll*sptch*shdg-sroll*chdg;
  Cgb[2][2]=croll*cptch;
  return;
}
```

```
// CBW2EA.CPP
// Function to compute the Euler angles (roll, pitch and platform azimuth)
// from the body to wander azimuth DCM.
//
// Body coordinates:
//
//   x-axis forward
//   y-axis right
//   z-axis down
//
// Wander azimuth coordinates:
//
//   x-axis displaced counterclockwise from north through wander angle
//   y-axis for right-hand system
//   z-axis up

#include <math.h>


// *****
// Compute roll, pitch and platform azimuth (Euler angles) from the
// body to wander azimuth DCM.
// *****
//
void Cbw2Ea(const double Cbw[3][3],
            double &roll, double &pitch, double &azimuth)
{
  roll=atan2(-Cbw[2][1],-Cbw[2][2]);
  pitch=atan2(Cbw[2][0],sqrt(pow(Cbw[0][0],2.0)+pow(Cbw[1][0],2.0)));
  azimuth=atan2(-Cbw[1][0],Cbw[0][0]);
  return;
}
```

```
// CWE2EA.CPP
// Function to compute the Euler angles (latitude, longitude and wander
// angle) from the wander azimuth to Earth DCM.
//
// Wander azimuth coordinates:
//
//   x-axis displaced counterclockwise from north through wander angle
//   y-axis for right-hand system
//   z-axis up
//
// Earth-fixed coordinates:
//
//   x-axis through North pole
//   y-axis for right-hand system
//   z-axis through Greenwich meridian

#include <math.h>


// *****
// Compute latitude, longitude and wander angle (Euler angles) from the
// wander azimuth to Earth DCM.
// *****
//
void Cwe2Ea(const double Cwe[3][3],
            double &latitude, double &longitude, double &wander)
{
  latitude=atan2(Cwe[0][2],sqrt(pow(Cwe[0][0],2.0)+pow(Cwe[0][1],2.0)));
  longitude=atan2(-Cwe[1][2],Cwe[2][2]);
  wander=atan2(-Cwe[0][1],Cwe[0][0]);
  return;
}
```

```
// DCM2Q.CPP
// Function to compute a quaternion from a direction cosine matrix.

#include <math.h>


// *****
// Obtain a quaternion from a direction cosine matrix.
// *****
//
void dcm2q(const double C[3][3], double q[4])
{
  q[0]=0.5*sqrt(1.0+C[0][0]+C[1][1]+C[2][2]);
  if (q[0]>=0.0002) {
    q[1]=0.25*(C[2][1]-C[1][2])/q[0];
    q[2]=0.25*(C[0][2]-C[2][0])/q[0];
    q[3]=0.25*(C[1][0]-C[0][1])/q[0];
  }
  else {
    if ((C[0][0]>C[1][1]) && (C[0][0]>C[2][2])) {
      if (C[2][1]<C[1][2])
        q[0]=-q[0];
      q[1]=sqrt(0.5*(1.0+C[0][0]));
      q[2]=0.25*(C[0][1]+C[1][0])/q[1];
      q[3]=0.25*(C[0][2]+C[2][0])/q[1];
    }
    else if ((C[1][1]>C[0][0]) && (C[1][1]>C[2][2])) {
      if (C[0][2]<C[2][0])
        q[0]=-q[0];
      q[2]=sqrt(0.5*(1.0+C[1][1]));
      q[1]=0.25*(C[0][1]+C[1][0])/q[2];
      q[3]=0.25*(C[1][2]+C[2][1])/q[2];
    }
    else {
      if (C[1][0]<C[0][1])
        q[0]=-q[0];
      q[3]=sqrt(0.5*(1.0+C[2][2]));
      q[1]=0.25*(C[0][2]+C[2][0])/q[3];
      q[2]=0.25*(C[1][2]+C[2][1])/q[3];
    }
  }
  return;
}
```

```
// QUPDT.CPP
// Function to update a quaternion using third order quaternion integration.

#include <math.h>


// *****
// Update a quaternion using third order quaternion integration.
// *****
//
void qupdt(const double Dangl[3], const double Dangl_L[3], double q[4])
{
  double L;
  double chi[4];
  double qq[4];

  L=pow(Dangl[0],2.0)+pow(Dangl[1],2.0)+pow(Dangl[2],2.0);
  chi[0]=1.0-0.125*L;
  chi[1]=(0.5-L/48.0)*Dangl[0]+(Dangl[2]*Dangl_L[1]-Dangl[1]*Dangl_L[2])/24.0;
  chi[2]=(0.5-L/48.0)*Dangl[1]+(Dangl[0]*Dangl_L[2]-Dangl[2]*Dangl_L[0])/24.0;
  chi[3]=(0.5-L/48.0)*Dangl[2]+(Dangl[1]*Dangl_L[0]-Dangl[0]*Dangl_L[1])/24.0;

  qq[0]=q[0]*chi[0]-q[1]*chi[1]-q[2]*chi[2]-q[3]*chi[3];
  qq[1]=q[0]*chi[1]+q[1]*chi[0]+q[2]*chi[3]-q[3]*chi[2];
  qq[2]=q[0]*chi[2]-q[1]*chi[3]+q[2]*chi[0]+q[3]*chi[1];
  qq[3]=q[0]*chi[3]+q[1]*chi[2]-q[2]*chi[1]+q[3]*chi[0];

  L=sqrt(pow(qq[0],2.0)+pow(qq[1],2.0)+pow(qq[2],2.0)+pow(qq[3],2.0));
  q[0]=qq[0]/L;
  q[1]=qq[1]/L;
  q[2]=qq[2]/L;
  q[3]=qq[3]/L;

  return;
}
```

```
// Q2DCM.CPP
// Function to compute a direction cosine matrix from a quaternion.

#include <math.h>


// *****
// Obtain a direction cosine matrix from a quaternion.
// *****
//
void q2dcm(const double q[4], double C[3][3])
{
  C[0][0]=1.0-2.0*(q[2]*q[2]+q[3]*q[3]);
  C[0][1]=2.0*(q[1]*q[2]-q[0]*q[3]);
  C[0][2]=2.0*(q[1]*q[3]+q[0]*q[2]);
  C[1][0]=2.0*(q[1]*q[2]+q[0]*q[3]);
  C[1][1]=1.0-2.0*(q[1]*q[1]+q[3]*q[3]);
  C[1][2]=2.0*(q[3]*q[2]-q[0]*q[1]);
  C[2][0]=2.0*(q[3]*q[1]-q[0]*q[2]);
  C[2][1]=2.0*(q[3]*q[2]+q[0]*q[1]);
  C[2][2]=1.0-2.0*(q[1]*q[1]+q[2]*q[2]);
  return;
}
```